Efficient Tracking Operation for Supply Chain Management based on Blockchain Technology

By

Lim Wei Ching

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS) DIGITAL ECONOMY TECHNOLOGY

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor Ts Dr. Ooi Joo On, who has given me this bright opportunity to engage in an IC design project. It is my first step to establish a career in IC design field. Besides that, they have given me a lot of guidance in order to complete this project. When I was facing problems in this project, the advice from them always assists me in overcoming the problems. Again, a million thanks to my supervisor and moderator.

Besides, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

The global supply chain for high-value goods is persistently challenged by issues of opacity, inefficiency, and susceptibility to counterfeiting, which significantly erode consumer trust and brand value. Traditional Supply Chain Management (SCM) systems often lack the requisite transparency and traceability to effectively combat these problems. This project is the development of a blockchain-based system for efficient tracking operations in luxury watch supply chain management. The luxury watches industry faces significant challenges from counterfeiting and a lack of transparency in traditional supply chains which lead to substantial financial losses and erosion of consumer trust. This project proposes a solution leveraging blockchain technology to create an immutable, transparent, and secure digital ledger for tracking luxury watches throughout their lifecycle. The system implements smart contracts on the Ethereum platform to manage raw material registration, raw material tracking, component registration, component tracking, certification, NFT watch, and ownership transfers, with a role-based access control mechanism to ensure appropriate permissions for different stakeholders. The proposed system architecture consists of interconnected smart contracts and a React-based frontend interface, designed to provide comprehensive tracking capabilities while addressing implementation challenges related to the physical-digital link, privacy, and scalability. Through the integration of artificial intelligence, the system can make watch price predictions and provide consulting services. Preliminary work demonstrates the feasibility of the approach through the implementation of core smart contracts and basic frontend functionality. This project contributes to the field by presenting an integrated approach to luxury watch tracking to implement granular access control, enable component-level traceability, and provide a practical blueprint for blockchain applications in luxury goods authentication.

Area of Study: Blockchain Technology, Anti-Counterfeiting

Keyword: Blockchain Implementation, Cryptography, Decentralized Application, Ethereum, Supply Chain Management, Luxury Goods Tracking, Smart Contracts,
Product Authentication, Role-Based Access Control, Traceability

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

%	Percent

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
BTC	Blockchain Technology
PoS	Proof of Stake
PoW	Proof of Work
SCM	Supply Chain Management
NFT	Non-Fungible Token
Dapps	Decentralized Applications
EVM	Ethereum Virtual Machine
HLF	Hyperledger Fabric
DLT	Distributed Ledger Technology
UTXO	Unspent Transaction Output
BSC	Binance Smart Chain
DeFi	Decentralized Finance
PoSA	Proof of Staked Authority
ekEVM	Zero-Knowledge Ethereum Virtual Machine
CorDapps	Corda Distributed Applications
PoA	Proof of Authority
DPoS	Delegated Proof of Stake

CHAPTER 1
Introduction

This chapter provides an overview of the project's background on BCT, SCM, smart contracts, objectives, project scope and direction, contributions, and report organization.

Blockchain Technology

First of all, a blockchain is a group of blocks that functions as a safe record for transactions and a data store. BTC is a distributed and decentralised ledger system that securely logs transactions across numerous machines [1]. This method removes the need for a central authority or middleman to validate transactions by offering security, transparency, and immutability.

Besides, BCT's decentralisation minimises single points of failure and lowers the possibility of data manipulation. All transactions are transparent and verifiable since each network participant has access to a copy of the complete blockchain. It is impossible to alter or delete once the data is recorded in a blockchain. Blockchain networks validate and agree on transactions using consensus techniques like PoW and Proof of Stake PoS. This process ensures that all network participants come to a common agreement before new data is added. This technology opens the door to a new economy where blockchain will rule smart contracts, digital assets, and currencies [2].

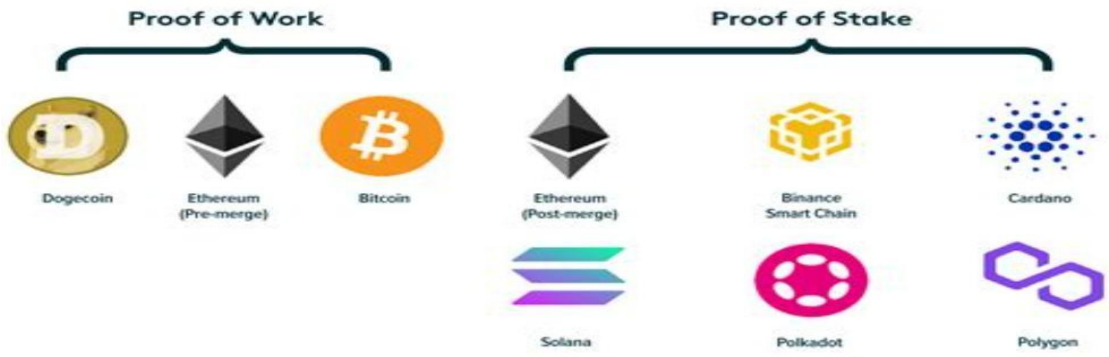

Figure 1.0.1 Blockchain Platforms Using PoW and PoS Consensus Mechanisms

Furthermore, PoW is a consensus algorithm that requires the involvement of people called miners, who must work through challenging cryptographic puzzles in order to validate transactions and add new blocks to the blockchain. This method uses a significant amount of energy and is quite computationally demanding [3]. The alternative consensus method known as Proof of Stake (PoS), on the other hand, chooses validators according to the number of bitcoin tokens they possess and are prepared to "stake" as collateral. Validators are selected to generate new blocks and validate transactions in direct proportion to their staked amounts, rather than solving cryptographic riddles [3].

In addition, the establishment of consensus within a decentralised framework is the goal of both PoW and PoS to ensure that all network participants concur on the legitimacy of transactions. The PoW is based on processing power and energy usage, whereas PoS relies on the economic stakes of its participants [4]. As depicted in Figure 1.0.1, PoW and PoS are two different consensus techniques that are applied by different blockchain systems, each of which uses a different approach to network security, transaction validation, and consensus-building throughout the distributed system.

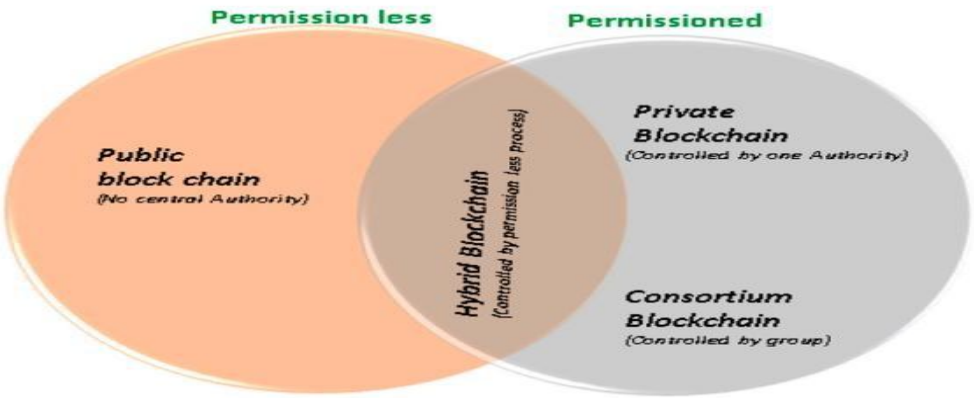

Figure 1.0.2 Types of Blockchain

According to Figure 1.0.2, there are four distinct types of blockchain technology, which can be categorized into two primary blockchain networks. The unrestricted accessibility of a permissionless blockchain means that anyone can join the network without requiring special authorisation or approvals [1]. Operating on a decentralised network of nodes, this kind reduces the possibility of censorship or manipulation by a single party and helps to disperse

power. This is due to the fact that every transaction on a permissionless blockchain is documented on a public ledger, transparency is made possible and anybody can confirm the integrity of the data and transaction histories. In contrast, a permissioned blockchain restricts participation and access to a small number of authorised individuals [5]. This structure frequently leads to improved privacy because it allows sensitive data to be shared only with authorised parties or kept offchain.

Moreover, the public blockchain represents one of the fundamental types of blockchain technology that completely embraces decentralisation is the public blockchain. It positions itself as a competitive alternative to established financial systems by utilising consensus methodologies like PoW and PoS. On the other hand, a private blockchain is less decentralised because it only allows a limited number of nodes to participate, which will lead to improved security. A hybrid blockchain incorporates features from both public and private blockchains to preserve some features as publicly available while permitting others to be managed by particular entities. This method offers a way to safely protect privacy while allowing some data to be publicly accessible. Finally, the consortium blockchain offers a cutting-edge solution that is suited to organisational requirements, making it easier for its members to validate and handle transactions.

Supply Chain Management

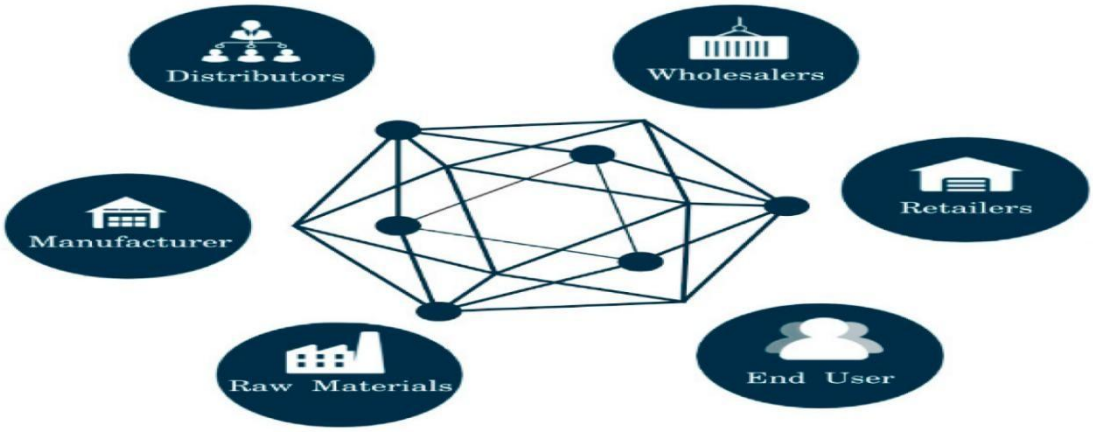

Figure 1.0.3 Decentralized Supply Chain Management

BCT has the potential to revolutionise SCM by improving efficiency, traceability, and transparency. Data silos, fraud, and a lack of transparency are common problems in traditional

supply chains that can result in inefficiencies and higher expenses. Based on Figure 1.0.3, businesses may use blockchain's decentralised and irreversible ledger to carefully monitor the flow of goods at every point of the supply chain, from manufacturing to final delivery. This promotes increased cooperation and confidence between all parties concerned by guaranteeing that all stakeholders have access to the same trustworthy and unchangeable information [6].

Smart Contract

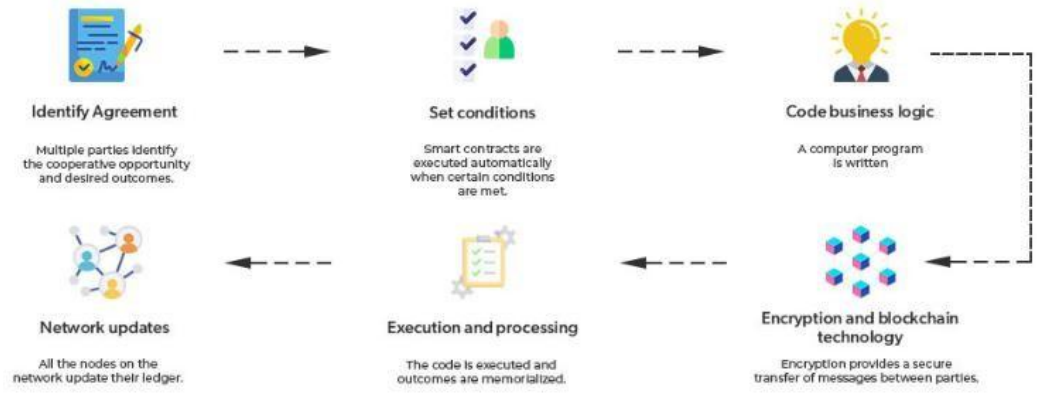

Figure 1.0.4 Process Flow of Smart Contract Execution in Blockchain

Furthermore, a smart contract is computer software created to automatically regulate the exchange of digital assets between parties in accordance with predetermined guidelines. A smart contract enforces agreements on its own and operates within predetermined bounds like a typical contract. These applications are run just as their creators intended. Smart contracts are enforced by their underlying code, as opposed to ordinary contracts, which are enforceable by law [7]. With predetermined rules inscribed on the blockchain, self-executing contracts can expedite a number of procedures, including confirming conditions, starting actions, and enabling safe ownership or information transfers [8].

Non-Fungible Token

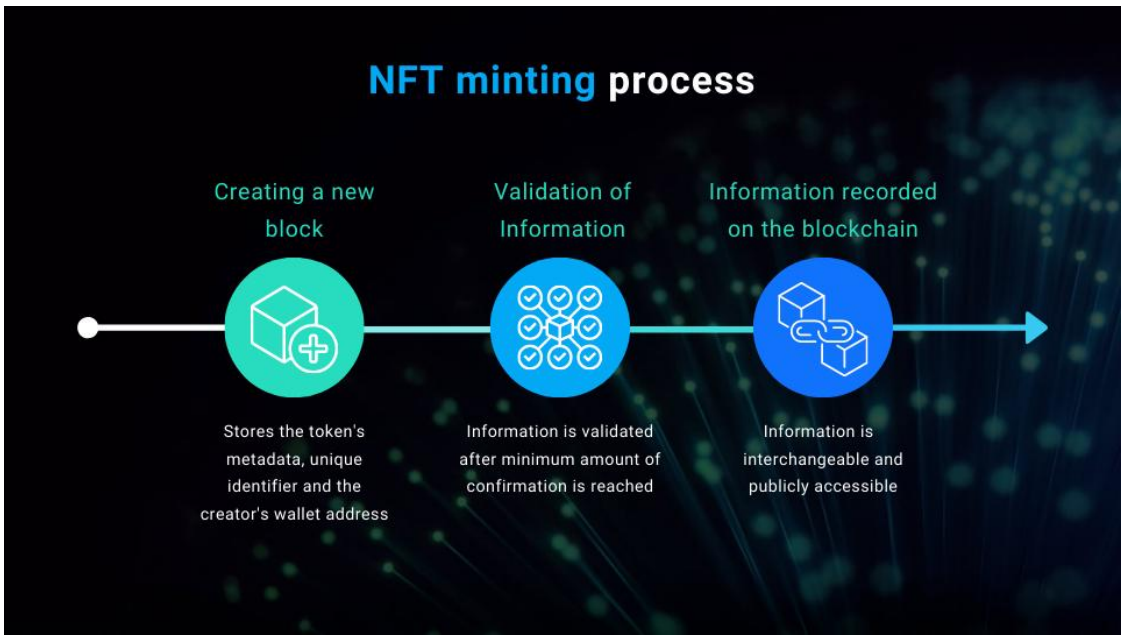

Figure 1.0.5 NFT Minting Process

In addition, Blockchain-based Non-Fungible Tokens (NFTs) offer a powerful solution for enhancing transparency and traceability in supply chains by providing unique, immutable digital representations of physical assets. A research investigation that developed a token-based blockchain architecture that captures all supply chain events through NFT "blueprints," ensuring comprehensive tracking from origin to transaction [39]. Another research introduced an AI-powered NFT minting platform on Solana to streamline minting, reduce costs, and automate provenance verification via smart contracts [40]. NFTs enable reliable certification of product origin and ownership, critical for sectors like food and pharmaceuticals, while also supporting sustainability by verifying ethical sourcing [41]. Together, these advances position NFT minting as a key innovation for efficient, transparent, and secure supply chain management.

1.1 Problem Statement and Motivation

Lack of Traceability in SCM

The flow of products and raw materials from suppliers to customers in traditional supply chains is frequently characterised by fragmentation across multiple businesses, each of which has its own systems for managing and tracking data. A lack of centralised and secure data may

arise from this decentralisation, which could make it harder to monitor products at every point in the supply chain and reduce visibility. This lack of traceability increases the possibility of fraud, counterfeiting, and non-compliance with established quality standards in the luxury goods industry, especially with regard to luxury timepieces. The authenticity and origin of the products are a major source of doubt for both businesses and customers in the absence of a safe way to track and validate each product's journey.

Difficulty in Verifying the Authenticity of Product

In the luxury watch industry, counterfeiting is a significant problem since expensive goods are regularly copied and passed off as genuine. Consumer confidence is eroded and luxury brands' reputations are negatively impacted by the difficulty of confirming the authenticity of pre-owned or grey market watches. Conventional authentication techniques, such serial numbers and certificates of authenticity, can be manipulated or faked. Both customers and producers face serious challenges in guaranteeing the authenticity of the product in this scenario, which could result in monetary losses and a deterioration in brand confidence.

Lack of Transparency in Traditional SCM

Multiple stakeholders, each running their own unique database, are commonly involved in traditional SCM systems. This compartmentalisation frequently results in a lack of transparency and makes it difficult for participants to provide data in real time. Inefficiencies, hold-ups, and disagreements about product origin, cost, and contract fulfilment can arise from this kind of fragmentation. Additionally, the lack of transparency makes it more difficult for customers to believe that goods, particularly expensive ones like luxury timepieces, have been sourced ethically and live up to the quality standards that have been promised.

1.2 Objectives

Implement Blockchain-Based Traceability for Luxury Watches

BCT has the potential to revolutionise the SCM of luxury timepieces because it allows for the recording of all transactions inside an irreversible, transparent, and decentralised ledger A distinct Non-Fungible Token (NFT), which functions as a digital proof of authenticity, can be linked to each watch. As the watch moves through the supply chain, from production to the point of sale, this NFT can be updated instantly. A blockchainbased solution like this ensures thorough traceability by providing a solid, auditable, and verifiable record of each transaction and ownership transfer. Additionally, blockchain's built-in security reduces the possibility of fraud and data manipulation, protecting the accuracy of the data exchanged across the supply chain.

Utilize Blockchain for Authenticity Verification Using NFTs

BCT offers a transparent and safe way to track timepieces over their whole lifecycle, which could help solve the authenticity verification issues in the luxury watch market. A distinct NFT, which is closely connected to crucial product details, manufacturing data, and an extensive ownership history, can be assigned to each watch. Any interested party can validate this NFT, which serves as a digital certificate of authenticity and reduces the possibility of counterfeiting. The blockchain permanently stores the watch's entire history, including repairs, ownership transfers, and authenticity checks, making it impenetrable and easily accessible. This creative method increases market trust and transparency by enabling buyers, sellers, and manufacturers to quickly confirm a watch's authenticity before buying or reselling it.

Implement Blockchain to Enhance Transparency and Collaboration in SCM

BCT offers a strong way to improve cooperation and transparency in conventional supply networks. Every transaction, from the procurement of raw materials to the ultimate sale, is documented on the blockchain through the use of a decentralised ledger that is open to all parties involved. This guarantees that the product's travel can be verified in real time by all

stakeholders. Furthermore, the use of smart contracts can help automate important procedures like delivery and payment. This method guarantees that everyone fulfils their responsibilities before a transaction is finalised. As a result, the system not only increases efficiency and reduces delays, but it also builds confidence by offering a high degree of supply chain transparency

1.3 Project Scope and Direction

The scope of this project involves a thorough design, development, and evaluation of a blockchain-enabled prototype system aimed at enhancing traceability and authenticity verification within the luxury watch supply chain. The primary deliverable is a functional proof-of-concept that demonstrates the essential capabilities of this system. The project commenced with the establishment of a blockchain infrastructure, focusing on the development and deployment of Solidity-based smart contracts on the Ethereum Sepolia test network. These contracts facilitate the digital representation and lifecycle management of watches and their critical components, encompassing activities such as registration, status updates, certification recording, warranty management, and ownership transfers. Additionally, the project necessitates the creation of an application layer through the development of a comprehensive full-stack web application. This application will feature a dynamic, role-aware frontend developed using React, designed to provide distinct user interfaces for key supply chain participants, including Administrators, Suppliers, Manufacturers, Retailers, Certifiers, Distributors, and Consumers, thereby ensuring secure interactions with the system. Furthermore, a robust backend will be constructed using Node.js and Express.js, which will oversee user authentication (utilizing JSON Web Tokens), authorization, business logic that is not suitable for on-chain execution and facilitate communication between both the blockchain and the relational database. Finally, data management will be enhanced through the integration of a PostgreSQL database to store user credentials, role information, and potentially off-chain application data or cached blockchain information to optimize performance. In addition, IPFS will be leveraged for decentralized storage of associated metadata, wherein immutable Content Identifiers (CIDs) will be stored on the blockchain.

1.4 Contributions

This project presents significant potential impact and contributes meaningfully to the academic understanding and practical application of blockchain technology within high-value supply chains. Its importance lies in addressing the critical and costly issues of counterfeiting and transparency in the multi-billion-dollar luxury goods market. By offering a robust, blockchain-based solution for authenticity verification and provenance tracking, this initiative delivers substantial value to various stakeholders. The project serves as a powerful instrument for safeguarding brand integrity, mitigating revenue loss due to counterfeiting, and enhancing brand reputation through demonstrable transparency. Furthermore, it has the potential to introduce new value added services grounded in verified product histories. In addition, it empowers consumers with the unprecedented ability to confirm the authenticity and provenance of high-value purchases, thereby fostering enhanced confidence and informed decisionmaking in both primary and secondary markets. This initiative also streamlines warranty and service claims. Moreover, the project enhances trust and efficiency among manufacturers, suppliers, distributors, and retailers by providing a shared, immutable source of truth. This reduces disputes and simplifies processes such as compliance verification and product recalls.

1.5 Report Organization

This report is systematically structured into six comprehensive chapters that provide a detailed examination of the blockchain-based luxury watch supply chain management system, encompassing theoretical foundations, technical implementation, and empirical evaluation.

Chapter 1 provides an introduction to the project's theoretical and practical foundations. It begins with an overview of blockchain technology, examining consensus mechanisms such as Proof of Work (PoW) and Proof of Stake (PoS), and categorizing blockchain types including public, private, hybrid, and consortium networks. The chapter establishes the context for supply chain management challenges, particularly in luxury goods authentication, and introduces smart contracts and Non-Fungible Tokens (NFTs) as technological solutions. It articulates the problem statement addressing traceability gaps, authenticity verification difficulties, and transparency limitations in traditional supply chain management. The chapter concludes by defining three primary objectives: implementing blockchain-based traceability, utilizing NFTs for authenticity verification, and enhancing transparency through decentralized collaboration.

Chapter 2 presents a comprehensive literature review examining blockchain applications across multiple domains. It begins with an in-depth analysis of blockchain platforms for product authentication, comparing Ethereum, Hyperledger Fabric, Corda, BNB Smart Chain (BSC), and Polygon across dimensions including consensus mechanisms, smart contract capabilities, scalability, and cost considerations. The chapter then explores NFT implementations for digital ownership and tracking across these same platforms, analyzing their respective strengths and limitations for luxury goods authentication. Finally, it examines real-world blockchain applications in supply chain management across diverse industries including agri-food, fisheries, healthcare, wood products, manufacturing, cross-border commerce, digital supply chains, and food crisis management, providing contextual understanding of blockchain's transformative potential.

Chapter 3 details the system methodology and approach, beginning with comprehensive system requirements specification including hardware specifications and extensive software tool requirements spanning blockchain development, backend frameworks, frontend libraries, and testing infrastructure. The chapter presents detailed system design diagrams including overall system architecture, use case diagrams illustrating seven stakeholder roles, activity diagrams mapping workflow processes from raw material registration through consumer ownership transfer, and class diagrams defining database entity relationships. The methodology emphasizes the integration of Secure Software Development Lifecycle (SSDLC) principles to ensure security and reliability throughout the development process.

Chapter 4 covers the system design architecture, presenting a comprehensive system block diagram illustrating the integration of blockchain smart contracts, backend services, frontend applications, and external storage systems. It details the blockchain-enabled supply chain management framework encompassing seven sequential stages from raw material sourcing through consumer acquisition. The chapter extensively covers database design using PostgreSQL for relational data management, smart contract development utilizing Solidity and OpenZeppelin standards for ERC-721 NFT functionality, backend API development using Node.js and Express.js frameworks, IPFS integration for decentralized storage through Pinata services, frontend development using React.js and Material-UI components, and comprehensive integration testing strategies ensuring system reliability and performance optimization.

Chapter 5 documents the complete system implementation process, beginning with detailed software setup procedures including MetaMask wallet configuration, Ethereum Sepolia testnet integration, Pinata IPFS account creation, and development environment establishment. The chapter provides comprehensive smart contract implementation details including inheritance structures, data models, and function implementations for raw material registration, component creation, certification processes, watch assembly with NFT minting, shipping management, retail operations, and consumer purchases. It extensively covers database schema implementation, backend API endpoint development across all stakeholder modules, and frontend implementation with role-based dashboard interfaces. The chapter concludes by documenting implementation challenges including blockchain integration complexities, multi-stakeholder workflow coordination, and IPFS storage reliability issues.

Chapter 6 evaluates the system through comprehensive testing methodologies and performance analysis. It details testing frameworks including Hardhat for smart contract validation, Postman for API endpoint verification, and Chai-Mocha for automated testing procedures. The chapter presents extensive testing results across smart contract functions, backend API operations, and frontend user interfaces, demonstrating system functionality and reliability. It discusses significant project challenges including IPFS connectivity issues, blockchain transaction management complexities, performance optimization requirements, and multi-browser compatibility considerations. The chapter evaluates achievement of the three primary objectives, demonstrating successful implementation of blockchain-based traceability, NFT-based authenticity verification, and enhanced supply chain transparency. It concludes with a comprehensive assessment of the project's contributions to luxury goods authentication and supply chain management innovation, establishing foundations for future research and commercial applications in decentralized authentication systems.

CHAPTER 2

Literature Reviews

2.1 Blockchain for Product Authentication

In a number of industries, including luxury goods, medicines, electronics, and food products, counterfeiting is a serious risk that can result in financial losses, harm to one's reputation, and even injury to consumers. Because of their lack of verifiable provenance and centralised vulnerabilities, traditional anti-counterfeiting techniques frequently fail. By offering a decentralised, unchangeable ledger to track products throughout their existence, blockchain technology presents a promising way to improve supply chain transparency and authenticity verification.

2.1.1 Ethereum for Product Authentication

Figure 2.1.1.1 Ethereum Logo

Due to its extensive decentralised network design and strong support for smart contracts, Ethereum, which was introduced in 2015, is one of the most well-known and frequently used public blockchain systems [11]. Although its primary purpose at first was to enable peer-to-peer digital currency transactions (such as those involving Ethereum or ETH), its capabilities go well beyond straightforward value transfer.

Ethereum functions as a distributed public ledger that is cooperatively managed by a worldwide network of nodes, offering a base upon which to construct DApps. The EVM, a Turing-complete execution environment that enables developers to implement intricate, self-executing contracts, automating agreements and business logic without the need for middlemen, is its main invention. Ethereum is a powerful, albeit complicated, platform for

tackling issues with product authenticity and preventing counterfeiting because of its programmability.

Figure 2.1.1.2 Key Components of the Ethereum Blockchain Ecosystem

Firstly, applications like product verification that demand high levels of trust and data integrity are well suited to Ethereum's core characteristics. By dispersing trust throughout the network and making the system intrinsically resistant to censorship or manipulation by any central authority, its decentralised architecture removes dependence on a single controlling institution [12]. Secondly, by offering a tamperproof history, the blockchain ledger's immutability guarantees that, once a record (a product's unique identifier) has been validated and added to the chain, it is nearly impossible to change or remove [12]. Thirdly, smart contracts make it possible to automate verification procedures, handle the transfer of product ownership, and apply complex logic to track items as they move through the supply chain [12].

Deploying product authentication systems on the public Ethereum main net has several obstacles and vulnerabilities, despite its potential. Scalability is arguably the most frequently mentioned issue. For high-volume sectors that need quick authentication checks, the network's fluctuating block confirmation durations and

restricted transaction throughput (around 15–30 transactions per second) may not be enough [13]. Second, because to network demand, transaction charges (gas fees) on Ethereum can be quite erratic and high. Frequent authentication procedures, including scanning a product several times throughout the supply chain, may become unaffordable due to these expenses [13]. Thirdly, because flaws can result in serious money loss or system compromise, the intricacy of creating, implementing, and maintaining secure smart contracts calls for specific knowledge.

To sum up, Ethereum offers a solid, programmable, decentralised base with high integrity guarantees that are appropriate for product authentication.

2.1.2 Hyperledger Fabric for Product Authentication

Figure 2.1.2.1 Hyperledger Fabric Logo

Hyperledger Fabric (HLF) is a well-known enterprise-grade, permissioned distributed ledger technology (DLT) platform that stands out from public blockchains like Ethereum. It is primarily made to serve as a basis for commercial applications that need privacy, secrecy, and performance [14]. Fabric, which has a modular architecture and functions on the tenet that network users are known, identified, and expressly given authorisation to communicate, was introduced in 2016 under the Linux Foundation's Hyperledger umbrella. One of HLF's primary differentiators is its permissioned model, which makes it ideal for sectors like SCM and product authentication in business networks or consortiums where data sensitivity and limited access are critical.

Additionally, HLF includes a number of unique features that are essential for strong product authentication systems. A permissioned network structure is one of its advantages, which guarantees that only authorised entities may take part, hence improving security and trust inside a certain business environment [14]. Second, Fabric makes use of channels, which serve as private subnets for communication between certain users. To meet important privacy needs for sensitive product or commercial

data, transactions and ledger state within a channel are separated and confidential, viewable only to authorised participants of that channel. Thirdly, the business logic is defined by smart contracts in Fabric, sometimes referred to as chaincode, which can be written in common programming languages like Java, Node.js, or Go. Chaincode facilitates the automation of provenance tracking, ownership transfers, and authentication rules by managing assets and regulating transactions.

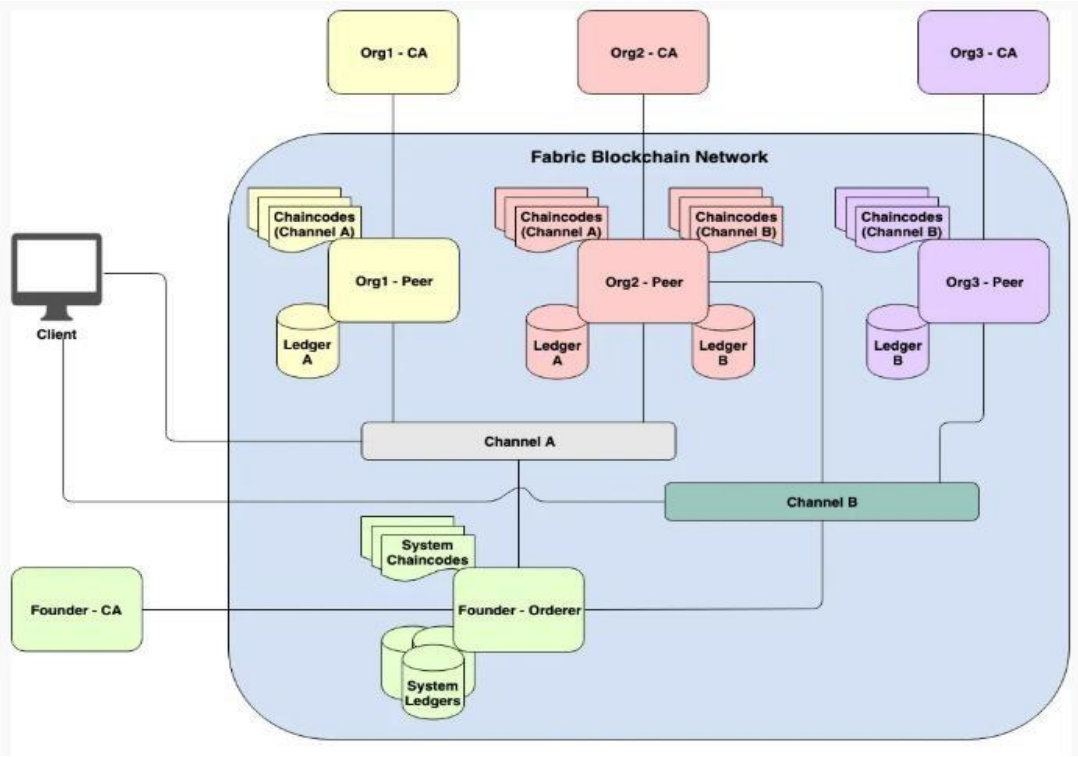

Figure 2.1.2.2 Fabric Network with Multiple Channels

HLF's enterprise focus directly contributes to its strengths for product authentication. The primary benefit, as illustrated in Figure 2.1.2.2, is privacy and secrecy, which are made possible by the channel architecture and permissioned design. This is crucial when handling sensitive supply chain logistics or proprietary product information. Second, in contrast to many public blockchains, Fabric is designed for greater scalability and performance. It is more practical for high-volume authentication scenarios because of its consensus algorithms, which enable higher transaction throughput and reduced latency [15]. Thirdly, companies using extensive tracking and verification might profit greatly from predictable operating expenses due to the lack of native cryptocurrency and related transaction fees (gas) [15]. However, implementing

HLF for product authentication also presents certain weaknesses and challenges. The primary drawback is often its complexity. Configuring channels and policies requires significant technical expertise compared to deploying applications on public blockchains [15]. Secondly, while being permissioned is a strength for privacy, it also means HLF is less decentralized than public networks. Trust is shifted from cryptographic consensus across unknown miners to the governance framework and trust relationships established among the participating organizations in the consortium. Thirdly, establishing effective governance models for a multi-party Fabric network to decide who controls permissions, manages upgrades, and resolves disputes can be organizationally complex.

In summary, HLF provides a strong, safe, and expandable platform designed for supply chain applications and enterprise product authentication where control, privacy, and performance are critical needs.

2.1.3 Corda for Product Authentication

Figure 2.1.3.1 Corda Logo

Another major participant in the enterprise blockchain market is Corda, created by R3. It is frequently used in conjunction with HLF but has unique architectural options designed mainly for regulated sectors like healthcare, insurance, and finance [16]. It is a private, permissioned, open-source DLT platform made to help well-known companies trade value in a direct and safe manner. Corda places a strong emphasis on point-to-point communication and data privacy, disclosing transaction details with parties only when necessary, in contrast to standard blockchains that broadcast transactions globally. It is a pertinent platform for certain product authentication and

supply chain scenarios where secrecy is crucial because of its emphasis on privacy and interoperability.

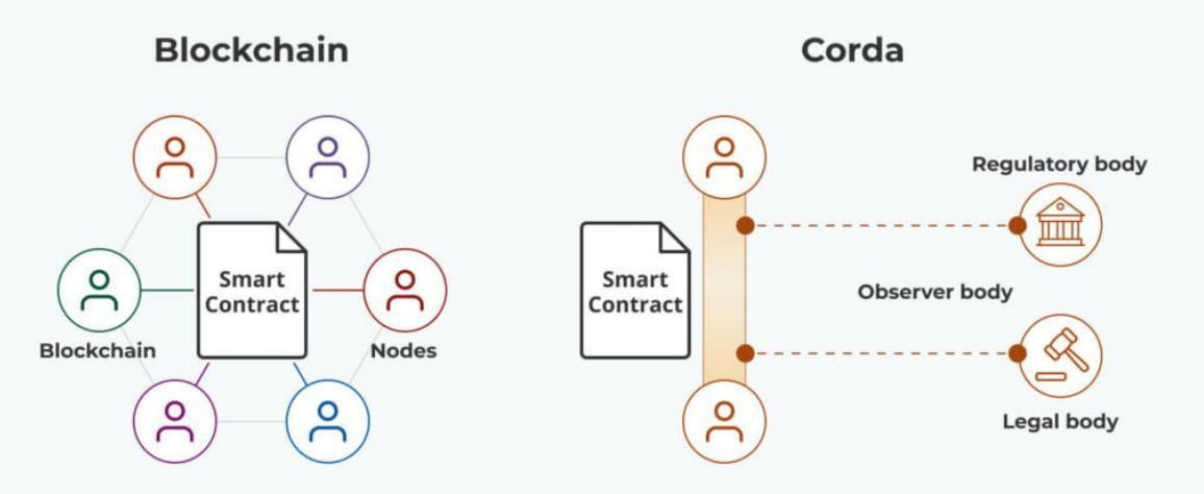

Figure 2.1.3.2 Blockchain vs Corda: Smart Contract Architecture Comparison

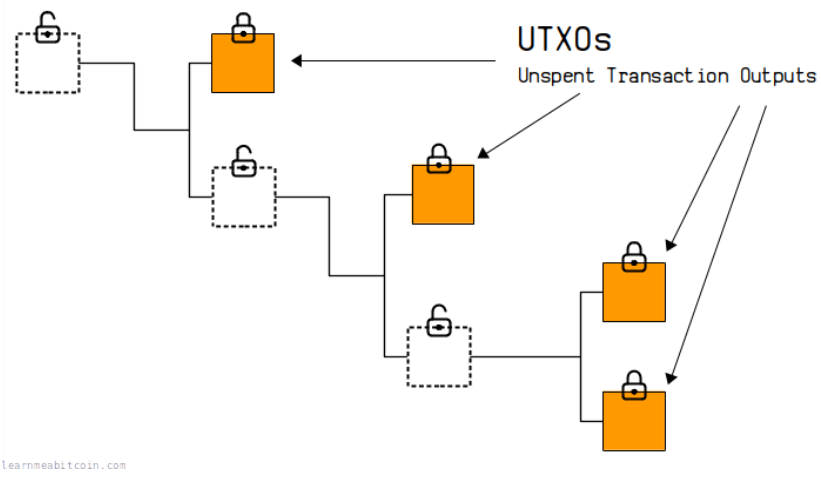

Figure 2.1.3.3 UTXO

Corda and its suitability for product authentication are defined by a few essential characteristics. First off, only recognised users who have been verified by a strong identity layer are permitted to join the network and conduct transactions due to its private and permissioned nature [16]. Second, transactions are not widely disseminated due to Corda's distinct data sharing methodology. By default, this offers strong data secrecy. Thirdly, Corda uses a concept similar to UTXO, in which transactions generate new states and consume current ones, giving tracked items a distinct and verifiable provenance.

One of Corda's main advantages for product authentication is its business-oriented design philosophy. Its point-to-point transaction model's inherent anonymity is a significant benefit for businesses who are hesitant to reveal private product or supply chain information, even on a permissioned ledger [17]. In contrast to conventional blockchain models, the Corda design seeks to increase scalability and performance by skipping worldwide broadcasting and consensus for each transaction. This could enable it to handle larger transaction volumes more effectively. Furthermore, the emphasis on network and CorDapp compatibility is a significant advantage for tying together diverse systems in a complicated value chain. Nevertheless, Corda has drawbacks for use cases involving product authentication. Building, implementing, and maintaining CorDapps and the supporting network infrastructure can be quite difficult, much like other corporate platforms [16] [17]. Second, the reviewed article pointed out that while Corda's smart contracts meet the fundamental business logic requirements, they may not be seen as "fully functioning" as platforms such as Ethereum [17]. Furthermore, although its ecosystem is robust in the financial sector, its development community and acceptance may be less than those of Ethereum or Hyperledger in more general supply chain or consumer-facing authentication domains. To sum up, Corda provides a distinctive, privacy-focused DLT solution that is ideal for product authentication in business networks or regulated industries where openness and communication between known parties are essential.

2.1.4 BSC (BNB Smart Chain) for Product Authentication

Figure 2.1.4.1 BSC Logo

In 2020, Binance introduced the BNB Smart Chain (BSC), a high-performance blockchain that functions in parallel to Binance [18]. By providing complete smart contract functionality and EVM compatibility, together with substantially lower transaction fees and quicker confirmation times than Ethereum's mainnet at the time, BSC soon gained interest, especially in the DeFi space [19]. In contrast to Ethereum and enterprise-focused platforms, its architecture, which strikes a balance between speed and cost-effectiveness, offers a unique set of trade-offs for product authentication applications.

BSC's potential function in product authentication is defined by a few essential characteristics. First and foremost, its EVM compatibility is a key selling point since it enables developers to quickly utilise the vast ecosystem of Ethereum development tools, libraries, and knowledge or port already-existing DApps with authentication based on Ethereum [19]. Second, BSC is compatible with smart contracts, which allow for the development of automated logic for managing ownership transfers (perhaps through NFTs utilising the BEP-721 standard, which is comparable to ERC-721), tracing provenance, confirming the authenticity of products, and putting anti-counterfeiting measures in place. Third, BSC utilises a consensus process known as Proof of Staked Authority (PoSA). In contrast to Ethereum's PoW (at the time) or even its current PoS, this hybrid approach uses a small group of validators that stake BNB tokens to protect the network, allowing for faster block times (around 3 seconds) and better transaction throughput [19].

For product authentication systems looking for speed and cost effectiveness, BSC has a number of advantages. The main benefit is its high transaction speed and performance, which are made possible by the PoSA consensus. This makes it appropriate for applications that need near real-time verification or frequent on-chain updates [18]. Second, it is more economically viable to record many authentication events or track specific items across the supply chain due to the historically lower transaction costs when compared to Ethereum L1 [20]. Thirdly, access to a vast developer community and well-established tools is made possible by the EVM compatibility, which also considerably reduces the development barrier [19].

Centralisation is the subject of the most important critique. In contrast to more decentralised networks like Ethereum, the PoSA consensus depends on a comparatively small, permissioned group of validators, which is heavily influenced by Binance. This raises questions regarding censorship resistance and the network's vulnerability to control or meddling by a central party [20]. Second, a lot of security problems, breaches, and attacks that target DApps and bridges developed on the platform have afflicted the BSC ecosystem. The regularity of these occurrences raises concerns about the ecosystem's apps' overall security posture or the ease with which potentially vulnerable contracts can be deployed, even though they are not necessarily defects in the core protocol itself [20].

In summary, BSC offers an EVM-compatible, low-cost, high-throughput platform that may be appealing for product authentication applications where budget and speed are top priorities.

2.1.5 Polygon for Product Authentication

Figure 2.1.5.1 Polygon Logo

The main goals of Polygon (formerly Matic Network), a prominent platform, are to scale Ethereum and improve its infrastructure [21]. By providing a comprehensive suite of Layer 2 scaling solutions and a foundation for creating interconnected blockchain networks, it overcomes Ethereum's intrinsic drawbacks, including high transaction fees (gas costs) and network congestion that causes long confirmation times [22]. Known as "Ethereum's Internet of Blockchains," Polygon is a very relevant

platform for implementing scalable product authentication applications since it leverages Ethereum's security and established ecosystem to increase transaction efficiency and lower costs.

Polygon's architecture offers a diverse set of features applicable to product authentication. Firstly, its core offering includes Layer 2 scaling solutions, most notably the Polygon PoS sidechain and newer advancements like Polygon zkEVM (ZeroKnowledge Ethereum Virtual Machine). These solutions process transactions off the main Ethereum chain, significantly increasing throughput and reducing fees, before potentially anchoring or proving transaction validity back to Ethereum L1 [21]. Secondly, Polygon maintains full EVM compatibility, meaning smart contracts written for Ethereum (typically in Solidity) can be deployed on Polygon with minimal or no modification, and developers can utilize familiar Ethereum tools and wallets like MetaMask [22]. Thirdly, the popular Polygon PoS chain uses a PoS consensus mechanism, which offers faster block times than Ethereum L1 [22]. To protect the network and confirm transactions, a decentralised group of validators stakes POL tokens (previously MATIC).

The main advantage of Polygon for product authentication is its capacity to efficiently grow Ethereum applications. High scalability and transaction speed, provided by both the PoS chain and zkEVM solutions, are the main advantages. They can process thousands of transactions per second, which is far faster than Ethereum L1's [22]. This is essential for applications that track a lot of items or require regular authentication checks. Second, on-chain interactions for product registration, verification, or ownership transfer are economically feasible, even at scale, because to Polygon's significantly lower transaction costs as compared to Ethereum L1 [23]. Thirdly, apps may take use of Ethereum's security thanks to programs like Polygon zkEVM. These solutions inherit the strong security guarantees of the main Ethereum network by applying validity proofs that have been validated on L1 [23].

Despite its benefits, Polygon has drawbacks and things to keep in mind such product authentication. First, the security paradigm differs for each of Polygon's solutions. By contrast, ZK-rollups derive greater security assurances directly from Ethereum L1 via mathematical proofs [23], whereas the PoS chain depends on its own validator set, whose security depends on the value pledged and the degree of decentralisation, but it

does checkpoint to Ethereum. Knowing these differences is essential when selecting a certain Polygon solution.

To sum up, Polygon provides an attractive set of options for creating scalable and affordable product authentication systems, especially for companies that are already a part of the Ethereum ecosystem.

2.1.6 Summary of Reviewed Blockchain Technologies

Feature	Ethereum	Hyperledger Fabric	Corda	BSC	Polygon
Platform Type	Public L1	Private/Consortium (Permissioned DLT)	Private/Consortium (Permissioned DLT)	Public L1 (Permissioned Validators)	L2 Scaling (PoS, zkEVM) & L1 (SDK Chains)
Consensus	PoS	Pluggable	Notary-based	PoSA	PoS
Smart Contract Language	Solidity	Go, Node.js, Java	Java, Kotlin	Solidity, Vyper	Solidity, Vyper
EVM Compatibility	Yes	No	No	Yes	Yes
Confidentiality	Low	High	Very High	Low	Low
Scalability	Low	High	High	High	Very High
Transaction Costs	High & Volatile	None	None	Low	Very Low
Key Strengths (Auth)	Decentralization, Immutability	Privacy, Performance	Privacy, Interoperability	Speed, Low Cost	Scalability, Low Cost,
Key Weaknesses (Auth)	Scalability (L1), Cost (L1), Public Privacy	Complexity, Less Decentralized, Governance	Complexity, Notary Trust	Centralization, Security Record	Security Varies (PoS vs zk), Bridge Risk, Complex

Table 2.1 Summary Reviewed

2.2 NFTs for Digital Ownership and Tracking

NFTs are a noteworthy use of BCT that provides a way to manage and express ownership of distinct digital or physical assets. Each NFT is unique and verifiable due to its unique properties that are recorded on the blockchain, unlike fungible tokens (such as cryptocurrencies), where each unit is interchangeable. This section examines how NFTs are used for digital ownership and tracking on five well-known blockchain platforms: Polygon, HLF, Corda, BSC, and Ethereum.

2.2.1 Ethereum NFTs for Digital Ownership

First, Ethereum is the most well-known and fundamental blockchain for NFT development and implementation, primarily because of its innovative role and the broad acceptance of the ERC-721 standard [24]. Most of the early and well-known NFT initiatives, such as digital art, gaming assets, and collectibles, started on the Ethereum network. A standard interface for distinct, non-interchangeable tokens is defined by the ERC-721 Standard, which is one of its features [24]. The Ethereum ecosystem's wallets, marketplaces, and DApps are all guaranteed to work together thanks to this standardisation. In addition, each ERC-721 token is uniquely identified by combining its smart contract address with a particular token ID within that contract, making it unique and identifiable. Individual digital products can now be clearly distinguished and their ownership can be verified. Next, developers can incorporate intricate logic into NFTs using Ethereum smart contracts. This can include fractional ownership systems, dynamic features that alter in response to interactions or events outside the system, or automated royalties paid to artists upon secondary sales. Because it has the largest and most liquid NFT market, the greatest high-value sales volume, and the widest variety of marketplaces and collectors, Ethereum's strengths are network effects and liquidity. There are notable network effects from this well-established ecosystem.

2.2.2 Hyperledger Fabric NFTs for Digital Ownership

This permissioned blockchain technology, which is mainly intended for commercial applications, provides an alternative method of handling NFTs in contrast to public chains like as Ethereum. Its main objective is to offer restricted, confidential, and safe settings for the management of digital assets in groups or particular corporate networks

[25]. One of HLF's characteristics is its permissioned architecture. Participants in Fabric networks, for instance, must be authorised and validated. Because of its built-in control, it works well in business settings where only known parties can own assets and conduct transactions. Unlike public chains like Ethereum, this permissioned blockchain platform is developed particularly for commercial applications and provides an alternative method for NFTs. Its main goal is to offer private, controlled, and safe settings for managing digital assets in groups or particular corporate networks [25]. HLF has a permissioned architecture as one of its characteristics. For instance, participants in Fabric networks must be authorised and validated. It works well in business settings where asset ownership and transactions must be limited to identified parties because of its built-in control.

2.2.3 Corda NFTs for Digital Ownership

This enterprise-focused DLT platform, which mainly targets regulated sectors like finance, offers tokenisation features, including NFTs, via its own Token SDK. In contrast to public chains, its strategy places a strong emphasis on identification, privacy, and integration with business and legal processes, providing a unique model for digital ownership [26]. It provides a specialised SDK made to make the development and administration of tokens on the ledger easier. Standard interfaces and processes are offered by this SDK for both fungible and non-fungible tokens. Additionally, Corda shows NFTs and other assets as states on the ledger.
Similar in principle to the UTXO model, but with more robust data capabilities, ownership is tracked by these states, which are created and consumed in transactions. Certain classes, such NonFungibleToken, are provided by the Token SDK to represent distinct assets. The characteristics and actions of these tokens are specified by developers in CorDapps. Corda's need-to-know data sharing architecture makes it ideal for showcasing sensitive asset ownership in situations where secrecy is crucial [26]. In comparison to static NFT representations, the EvolvableTokenType feature offers a more flexible way to depict real-world assets whose properties evolve.

2.2.4 BSC NFTs for Digital Ownership

Since its initial debut by Binance as Binance Smart Chain, it has quickly become a wellliked platform for decentralised apps (DApps), including a sizeable portion of the NFT industry. It is currently a part of the larger BNB Chain ecosystem. It is a desirable choice for developers and users, especially for high-volume or low-value NFT applications, because it offers EVM compatibility together with substantially reduced transaction fees and faster confirmation times than Ethereum Layer 1 [27]. The EVM and BSC are completely interoperable. This makes it simple for developers to use well-known development tools like MetaMask, Truffle, and Remix, as well as to port current Ethereum smart contracts, including NFT standards like ERC-721 and ERC-1155. BSC uses a hybrid consensus process called PoSA, which combines aspects of PoA and Delegated Proof of Stake DPoS. This entails a small group of validators who are selected according to their reputation and investment in BNB, the native token.
Compared to Ethereum L1, this architecture allows for reduced fees and faster block times (around 3 seconds). In order to ensure uniform interfaces for ownership, transfer, and metadata, BSC uses its own token standards, mainly BEP-20 for fungible tokens (which are frequently used for payments within NFT marketplaces) and BEP-721/BEP1155 for non-fungible tokens. These standards function similarly to those of Ethereum
ERC.

2.2.5 Polygon NFTs for Digital Ownership

With a diverse ecosystem that includes the well-known Polygon PoS sidechain and the more recent Polygon zkEVM Layer 2 rollup, Polygon has made a name for itself as a top platform for scaling Ethereum. With the promise of reduced fees, quicker transactions, and interoperability with the well-established Ethereum ecosystem, both are being used for NFT apps, drawing in developers and consumers [28]. The compatibility of Polygon PoS and Polygon zkEVM with the Ethereum Virtual Machine is one of its main advantages. This makes it possible for Ethereum NFT smart contracts (ERC-721, ERC-1155) to be deployed smoothly and for users to communicate with each other using tools and wallets that they are accustomed to using, such as MetaMask. With its own PoS consensus, Polygon PoS offers incredibly low transaction costs and high throughput. It has emerged as a leading platform for extensive collection projects and NFT games. In order to improve its security by using zero-knowledge proofs for state validation submitted to Ethereum, it is transitioning to become a zkEVM

Validium. A Layer 2 solution called Polygon zkEVM bundles transactions and posts validity proofs to Ethereum L1 via zero-knowledge rollups. It provides EVM equivalency and scalability, while inheriting more of Ethereum's security. Because Polygon's transaction costs are far lower than Ethereum L1, NFT trading, minting, and other interactions are very accessible and reasonably priced.

2.2.6 Summary of Reviewed Blockchain Technologies

Feature	Ethereum	Hyperledger Fabric	Corda	BSC	Polygon
Platform Type	Public L1	Private/Consortium (Permissioned DLT)	Private/Consortium (Permissioned DLT)	Public L1 (Permissioned Validators)	L2 Scaling (PoS, zkEVM) & L1 (SDK Chains)
Consensus	PoS	Pluggable	Notary-based	PoSA	PoS
NFT Standard(s)	ERC-721, ERC-1155	Chaincode implementation	Corda Token SDK (NonFungibleToken)	BEP-721, BEP-1155 (ERC compatible)	ERC-721, ERC-1155 (ERC compatible)
EVM Compatibility	Yes	No (Can integrate via bridges/oracles)	No	Yes	Yes
Typical Gas Fees	High (L1),	None	None	Low	Very Low (PoS), Low (zkEVM)
Transaction Speed	Moderate (L1)	Fast	Fast	Fast (~3s blocks)	Very Fast (PoS), Fast (zkEVM)
Privacy Features	Pseudonymous (L1),	High	High	Pseudonymous	Pseudonymous

Key Strengths (NFTs)	Decentralization, Security, Liquidity, Network Effects	Privacy, Control, No Gas Fees	Privacy, Evolvable Tokens, Regulatory Focus	Low Fees, Speed, Large Ecosystem	Low Fees (PoS), Scalability, EVM Compatibility
Key Weaknesses (NFTs)	High Fees (L1), Scalability (L1), User Complexity	Complexity, No Public Interoperability	No EVM/Public Interoperability, Complexity, Niche Focus	Centralization Concerns, Security Perception (Ecosystem)	Security Model (PoS), Centralization (PoS), Bridge Risks

Table 2.2 Summary Review II

2.3 Use Cases BCT in Different Supply Chain Management

2.3.1 Agri-Food Supply Chain: Improving Traceability and Safety

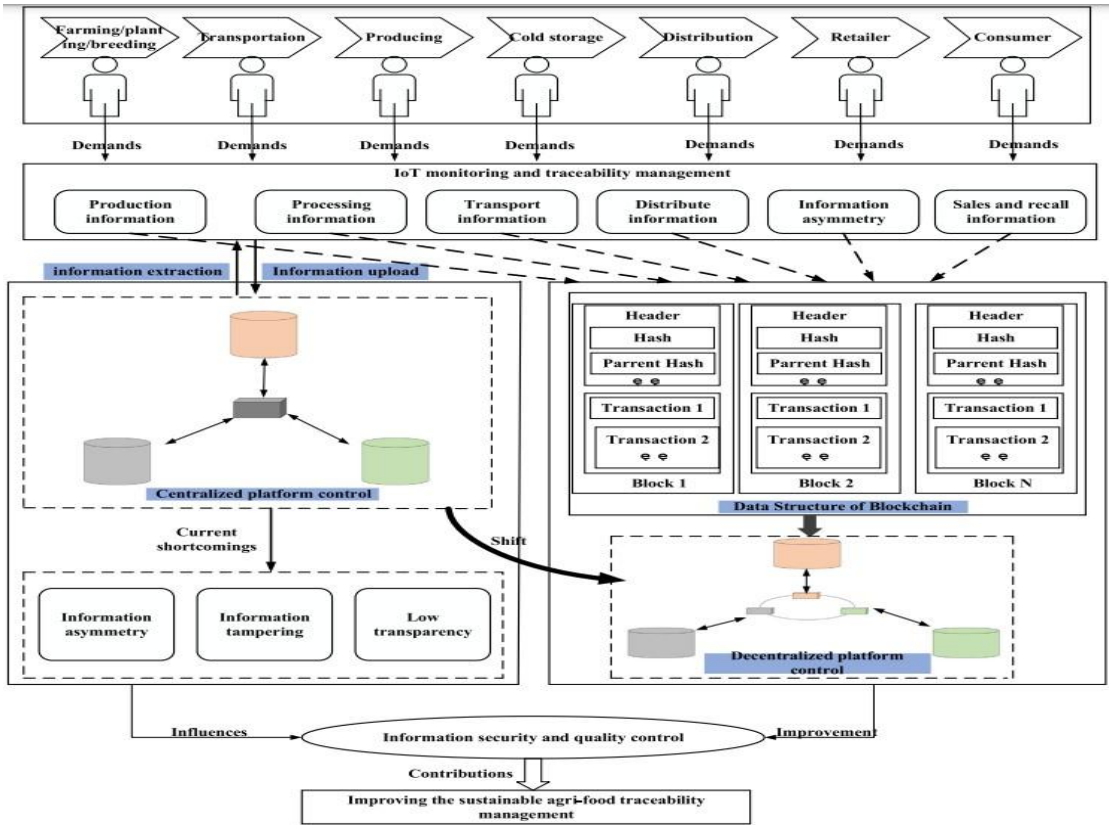

Figure 2.3.1.1 The traceability framework for blockchain-based operations

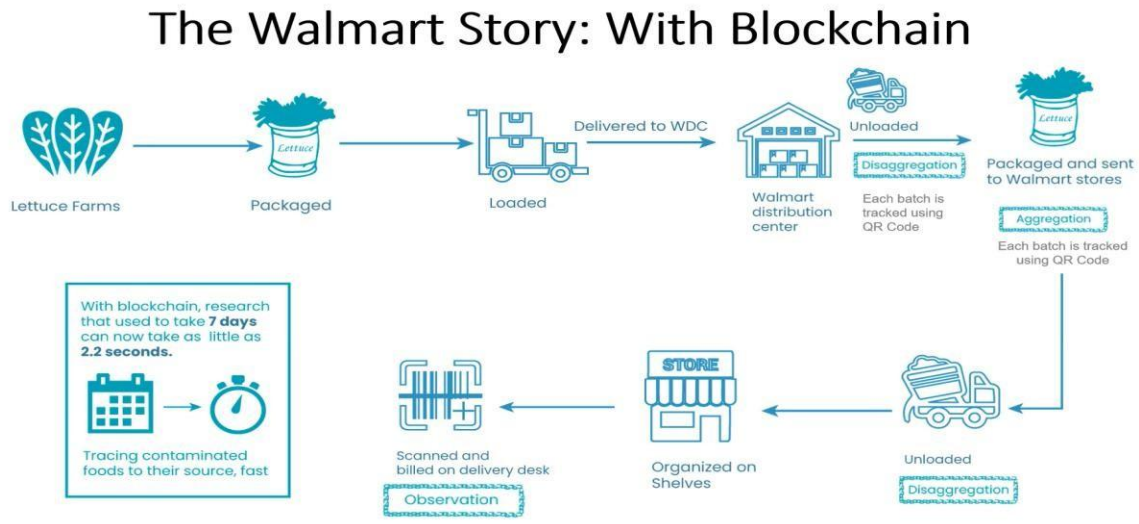

Figure 2.3.1.2 The Workflow of Walmart's Supply Chain with Blockchain

It is impossible to overestimate the significance of traceability in the agri-food industry. The concerns about food safety and quality of food have increased the demand for creative solutions to ensure transparency and accountability throughout the supply

chain from farm to fork. However, there are many problems in the traditional systems such as data fragmentation, delayed visibility, and the potential for fraud which will lead to jeopardized product integrity. The safety standards are stringent with the critical issues in the agri-food industry because any mistakes could harm public health. Thus, the implementation of BCT is crucial to create an immutable and decentralized ledger to track every transaction throughout the supply chain. The unique feature of BCT enhances trust and transparency across the whole supply chain and ensures that all stakeholders have access to a single and verified source of truth. Blockchain's immutability has managed to lower the possibility of data manipulation and ensure the integrity of shared information by preventing unauthorized changes. For example, the study describes the implementation of BCT to track and verify agricultural products such as fruits, vegetables, and grains throughout the journey of the supply chain [7]. Besides, the study offers detailed frameworks in Figure 2.3.1.1 for integrating BCT into SCM. These frameworks offer insightful recommendations to various companies looking to use BCT to enhance their tracking capabilities. The operational framework for a blockchain-based food traceability system contains specific mechanisms to ensure data accuracy, data security, and data accessibility throughout the supply chain. The framework also helps the companies to increase the efficiency and reliability of the supply chain operations by implementing BCT. The integration of BCT with IoT devices such as sensors and RFID tags will enable the stakeholders to access up-to-date information on the condition, location, and handling of products at every stage. This integration facilitates quicker responses to food safety incidents by reducing the risk of fraud, contamination, and foodborne illnesses. For instance, based on Figure 2.3.1.2, Walmart is a prominent corporation around the world that has implemented a blockchain-based system to track the leafy greens supply chain to reduce the time required to identify the source of contamination from weeks to seconds. Similarly, Carrefour aims to enhance customer trust and satisfaction which implementing BCT to enable the customer to trace the provenance of products like milk and chicken.

2.3.2 Fishery Supply Chain: Ensuring Sustainability and Reducing Carbon Footprints

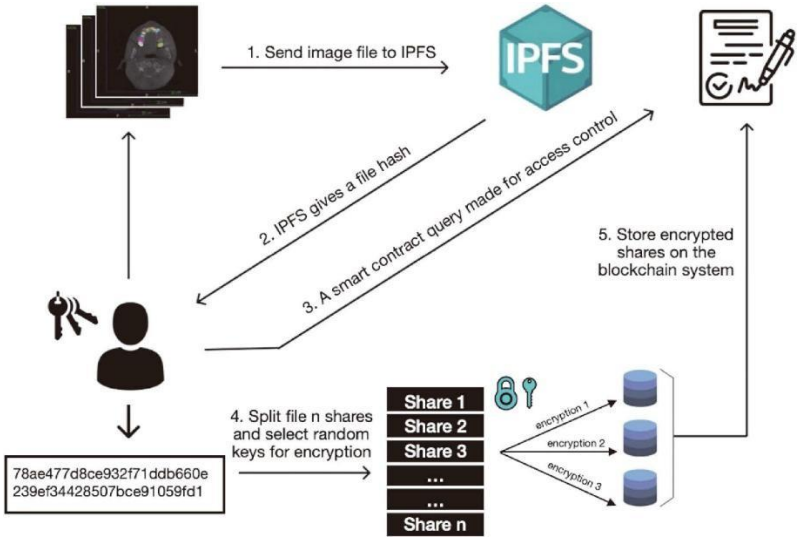

Figure 2.3.2.1 Blockchain and IPFS-Based Secure Data Management Architecture

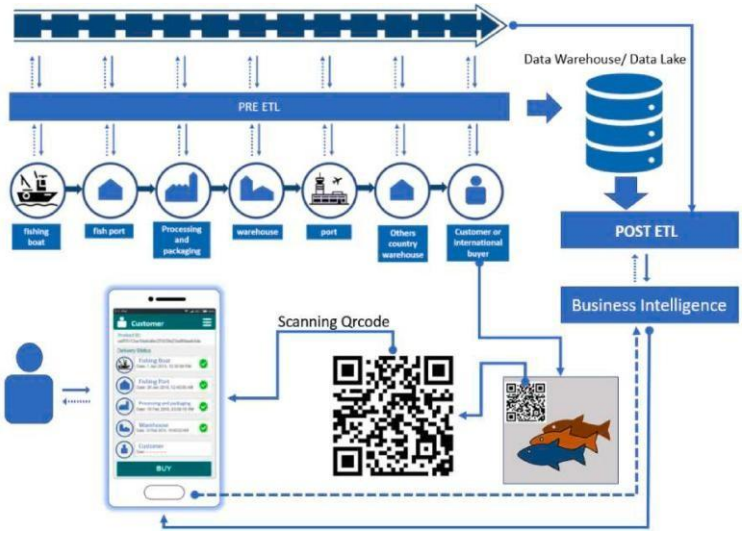

Figure 2.3.2.2 Proposed Integration of Big Data into the Blockchain

In the fishery supply chain, many obstacles endanger transparency, sustainability, and efficiency. This is because problems are quite common such as the lack of traceability, difficulties in managing carbon footprints, storage limitations on BCT, and the complexity of cross-border. All stakeholders including consumers, regulators, and suppliers lack confidence in the authenticity and quality of the products because they do not have a reliable way to trace the origin and journey of the fishery supply chain. It can be challenging to identify and effectively manage the high carbon emissions due to the complexity of the global supply chain. BCTs are not well-suited to store large amounts of datasets like images and videos which are crucial for detailed traceability

in the fishery industry. The problem is made more complicated by the intricacy of international trade and market expansion. Strong systems for data integration and administration across many countries are necessary for entering international markets. However, the current supply chain systems do not meet these requirements. The study proposed a method that integrates BCT with Big Data to improve sustainability, traceability, and transparency throughout the whole fishery supply chain [28]. The method is to combine big data architecture and blockchain by connecting it with other storage systems like the Interplanetary File System (IPFS) to overcome the storage limitations of BCT. Based the Figure 2.3.2.1, IPFS allows for the secure and efficient storage of large datasets such as photos and videos to provide detailed and accurate traceability information. In order to improve the functionality of the system, the study proposes new data management protocols such as Pre-ETL (Extraction, Transformation, and Loading) and Post-ETL protocols. Based on Figure 2.3.2.2, the protocols ensure consistent and simultaneous data processing between BCT and BD environments to optimize data management and integration. It leads to more efficient handling the complex data flows which is important in a dynamic environment like a fishery supply chain.

2.3.3 Healthcare Supply Chain: Securing Medical Records and Managing Products

In the healthcare industry, there are challenges associated with counterfeit drugs, data breaches, and regulatory compliance. Based on my research, a study suggests the integration of BCT with IoT technologies to solve the challenges [29]. It describes an IoT-driven blockchain-based architecture that includes a permissioned blockchain which is Hyperledger Fabric to securely manage sensitive data throughout the healthcare supply chain. This architecture allows for real-time tracking of medical goods such as vaccines and medications throughout the supply chain. The system ensures that the goods are transported and stored under the required conditions by utilizing BCT's immutable ledger. This will help to maintain their effectiveness and safety. The framework also improves security by using smart contracts to automate access control that only authorized users to access or modify data. The use of Hyperledger Fabric as a permissioned blockchain provides a safe environment where data privacy and integrity are preserved. For example, it only allows the registered and authorized entities to access the network. In summary, the integration of IoT and BCT

offers a great solution to the complexities of healthcare SCM by combining real-time monitoring, secure data management and automated access control to improve traceability, security, and efficiency of the healthcare supply chain.

2.3.4 Wood Supply Chain: Combating Illegal Logging and Promoting Sustainability

Based on the research, the wood supply chain faces several critical challenges such as illegal logging, deforestation, and label fraud have threatened sustainable forest management and eroded stakeholders' trust [30]. Traditional methods for tracking and verifying the origin of forest products are inefficient and lack transparency about illegal activities and compromise the integrity of the supply chain. Thus, the study proposes the adoption of BCT to provide a secure, decentralized, and transparent method for the journey of the forest product. BCT creates a decentralized ledger to record all the transactions and data entries immutably. The tamper-proof ledger greatly improves transparency and fosters stakeholder confidence to ensure that all participants have access to the same and unaltered information. In addition, BCT also offers a transparent and safe way to capture and verify the information which helps with regulatory compliance. There are numerous countries that have strict regulations such as the Australian Illegal Logging Regulation, the US Lacey Act, and the EU Timber Regulation (EUTR). BCT can help businesses comply with these regulations by providing a reliable record of all transactions and data. Thus, this will help the companies to follow the regulations more easily in their business.

2.3.5 Manufacturing Supply Chain: Enhancing Cooperation Transparency among SMEs

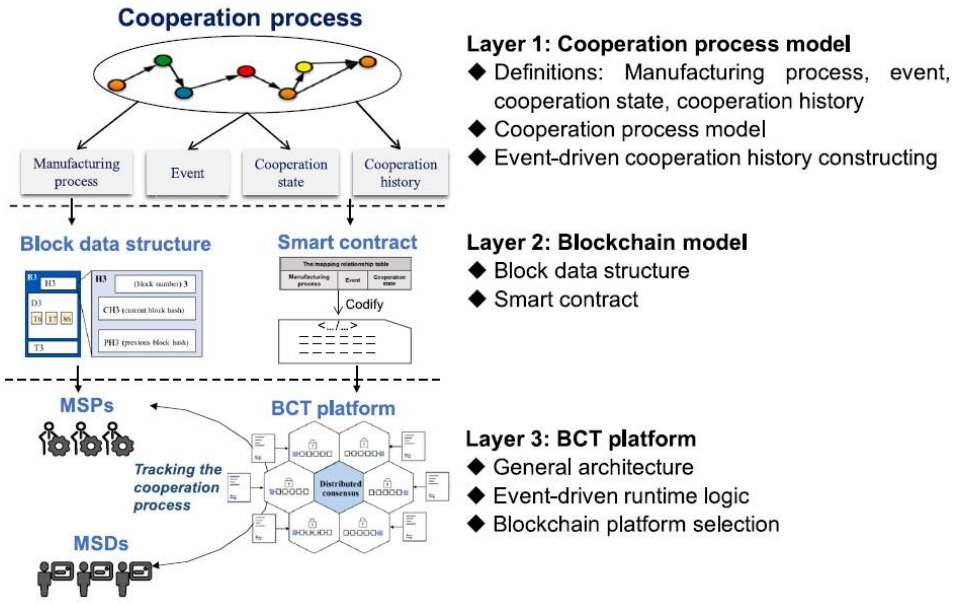

Figure 2.3.5.1 The BET framework's three-layered structure

In the manufacturing supply chain, SMEs frequently face challenges related to cooperation, transparency, and trust. The enterprises are not always able to fully trust each other which makes cooperative processes and data integrity become challenging. Based on the research, the study proposes a Blockchain-enabled and Event-driven Tracking (BET) framework that aims to improve transparency in the cooperation process among SMEs to solve the challenges [31]. Based on Figure 2.3.5.1, the BET framework combines an event-driven mechanism with BCT to provide an organized method of cooperation tracking. There are three main parts of the framework and the phases and activities involved in the cooperation process among SMEs are described in the cooperation process model. Then, the cooperation data can be managed and saved on a safe and decentralized platform established by the blockchain concept. The blockchain's event-driven method ensures that all data entries are precise, safe, and timely by causing specific actions to be taken based on predefined cooperation events. SMEs can securely capture, automate updates, and manage cooperation data by implementing the BET framework. This approach will minimize the need for manual intervention leading to reduce the errors and fostering greater collaboration between SMEs.

2.3.6 Cross-Border Supply Chain: Optimizing Tax and Pricing Strategies

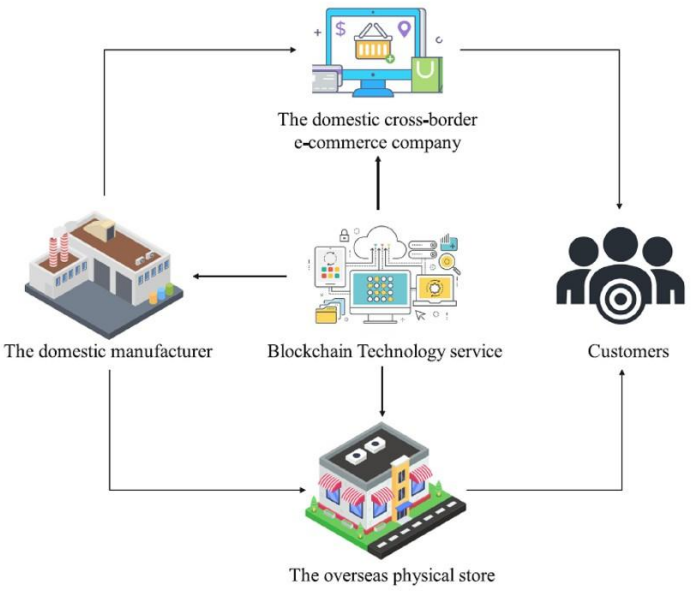

Figure 2.3.6.1 Dual-channel Supply Chain Model

In the cross-border supply chain, there are many challenges such as tax variations, competitive markets, and consumer sensitivity to blockchain attributes. Based on my research, the study explores a use case involving a cross-border dual-channel supply chain comprising a local manufacturer, a cross-border e-commerce company, and overseas physical stores [32]. It examines the different blockchain adoption scenarios to optimize the prices, investments, and strategic decisions. Thus, this will help the supply chain members to handle the complex regulatory environments more effectively. Moreover, BCT lowers uncertainty and gives a clearer picture of the supply chain performance and market demand in order to facilitate better investment decisions for investors. Based on Figure 2.3.6.1, the study indicates that in situations where a single member (domestic manufacturer) uses blockchain technology, other members may profit from free-riding, thereby increasing supply chain profitability overall without having to bear the associated costs of blockchain investment. On the other hand, domestic manufacturers are less inclined to utilize blockchain when the cost of blockchain investment is high and production variations are low. This suggests that weighing costs and advantages is crucial when making strategic decisions.

2.3.7 Digital Supply Chain: Investment and Carbon Reduction Strategy Based on the research, the study explores the integration of BCT with carbon reduction strategies in the digital supply chain by modeling the interactions between producers

and retailers in the supply chain and government interventions through the use of differential game theory [33]. It proposes a framework that uses BCT to record all carbon reduction activities and match compliance with regulatory standards. This is because BCT plays a critical role in this framework to enhance transparency and trust among supply chain members. This transparency helps all the supply chain members make informed decisions regarding investments in carbon reduction technologies and promotes better alignment with environmental and regulatory standards. The differential game-theoretic model used in the study analyses the strategic interactions among supply chain members over time, investigating different scenarios for BCT investments and government interventions. Thus, this model will lead to significant improvements in both economic efficiency and environmental performance. The government interventions such as incentives and penalties will play a crucial role in encouraging the adoption of BCT and green practices. In short, the model indicates that the best way to enhance the long-term economic and environmental advantages of digital supply chain is through a synergistic mechanism that involves bilateral involvement from both the private sector and government.

2.3.8 Food Supply Chain: Enhancing Resilience During Crises

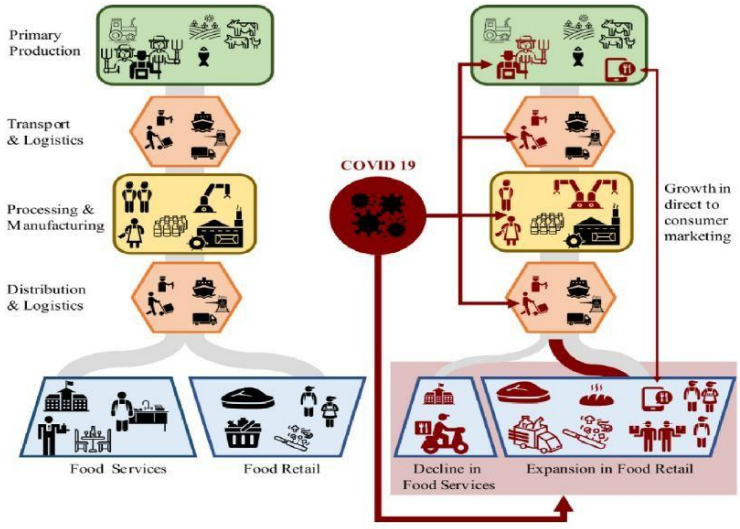

Figure 2.3.8.1 Impact of COVID-19 on Food Supply Chain Dynamics

Based on the research, the study investigates a use case in which a blockchain-based platform was utilized to establish a more effective and transparent food supply chain system during the COVID-19 pandemic [34]. BCT offers some advantages during the crisis such as providing a tamper-proof record of all transactions and enabling

stakeholders to monitor the locations and condition of commodities in real time. This level of transparency allows all participants to identify the bottlenecks and improve risk management by tracking the movements and storage conditions of food products across the supply chain. Apart from that, the use of smart contracts within a blockchain-based platform automates administrative tasks, reduces paperwork, and eliminates the need for intermediaries. The automation provided by smart contracts reduces the potential for human error and minimizes delays to ensure that the food products reach their destinations more quickly with fewer interruptions. Figure 2.3.8.1, shows that the COVID-19 pandemic has created a disruption point across all the stages which has affected the supply and demand of the food supply chain

2.3.9 Smart Contracts in Supply Chain Management: Reducing Transaction Costs and Improving Coordination

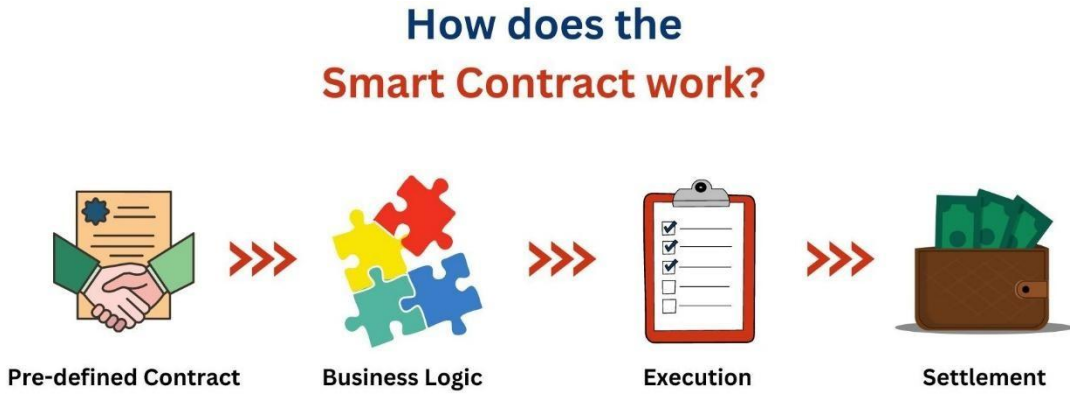

Figure 2.3.9.1 Key Benefits of Smart Contracts in Blockchain Technology

SCM can be revolutionized by BCT by emphasizing the creation of value rather than just the exchange of information. Businesses and people can safely exchange assets with BCT without the involvement of a third party, unlike with previous technologies. However, it can be challenging to find trustworthy partners and handle transactions effectively due to issues such as delivery management, last-mile logistics, and the bullwhip effect in a global supply chain. The study shows that there are two benefits of using BCT to reduce the costs which are the cost of value exchange without the middleman and the cost of independently verifying transaction properties [15]. These costs are inherent in traditional supply chains because the issues are quite common such as delays, discrepancies, and fraud. By using the distributed ledger of BCT to automate the verification process, it ensures the security and dependability of

transactions and vendors. Furthermore, BCT enhances settlement and reconciliation processes by ensuring that the transactions are finalized only once the specific conditions such as product quality and amount of capital are validated. Thus, this preserves a brand's value by preventing problems like conforming item delivery and ensuring the payments are issued only when the quality criteria are fulfilled. In short, automation eliminates the need for expense oversight to make for faster and more secure transactions.

CHAPTER 3

System Methodology/Approach

This chapter provides an outline of the suggested methodology and approach for the development of the smart contract framework project. The proposed approach for this project integrates blockchain technology into the luxury watch supply chain to enhance transparency, traceability, and authenticity verification. This solution employs a hybrid architecture involving smart contracts, backend systems, and a frontend user interface to provide a seamless, secure experience for all participants.

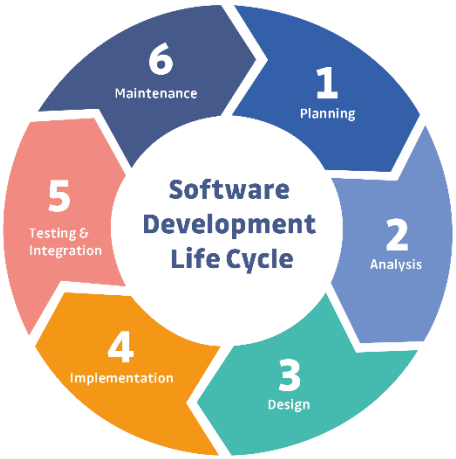

Figure 3.0 SSDLC Process

The integration of the Secure Software Development Lifecycle (SSDLC) into the "Efficient Tracking Operation for Supply Chain Management based on Blockchain Technology" project is crucial for ensuring that security is embedded throughout every phase of the system's development. SSDLC emphasizes the importance of building secure software from the outset, incorporating risk management, testing, and compliance strategies to mitigate potential vulnerabilities in the blockchain-based system [38].

This project revolves around the use of blockchain technology for tracking luxury goods within the supply chain, will benefit from SSDLC principles by addressing potential security issues in areas such as smart contract development, data storage, and user authentication. The system's reliance on Ethereum smart contracts and decentralized storage through IPFS requires rigorous

threat modeling to identify risks like unauthorized access, data tampering, or blockchain vulnerabilities. By applying SSDLC, the development process will integrate security practices, such as secure coding standards for Solidity contracts, regular code reviews, and continuous testing to detect any flaws early in the lifecycle.

Moreover, SSDLC ensures that data integrity is maintained across various stakeholders within the supply chain. Role-based access control (RBAC) and secure authentication mechanisms are integral to the system to ensure that only authorized participants can perform actions like registering watches or transferring ownership. With SSDLC, these security mechanisms are built and tested iteratively, ensuring that they meet high standards for robustness before the system goes into production.

By integrating SSDLC practices, this project aims not only to provide traceability and combat counterfeiting in the luxury watch market but also to establish a secure framework that can be trusted by all stakeholders, from manufacturers to consumers.

3.1 System Requirements

3.1.1 Hardware

The hardware used in this development is a laptop with a model of Dell Inspiron i5 3511, featuring an 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz processor and Intel(R) Iris(R) Xe Graphics Family graphics card. The operating system is Windows 11 with 8192MB of RAM and a 512GB SATA HDD.

Description	Specifications
Model	Dell Inspiron i5 3511
Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
Operating System	Windows 11
Graphic	Intel(R) Iris(R) Xe Graphics Family
Memory	8192MB RAM
Storage	512GB SATA HDD

Table 3.1 Specifications of Laptop

3.1.2 Software

Category	Tool/Library	Description
Languages	TypeScript	Static typing for frontend JavaScript.
	JavaScript	Language for backend (Node.js).
	Solidity	Language for writing smart contracts.
Blockchain & Smart Contracts	Ether.js	Library (FE/BE) to interact with the Ethereum blockchain.
	Hardhat	Development environment for Ethereum smart contracts.
	@nomicfoundation/hardhat-toolbox	Hardhat plugin bundle including common tools.
	@openzeppelin/contracts	Library of reusable and secure smart contract components.
	Smart Contract ABIs	JSON files defining contract interfaces (used by FE/BE)
Decentralized Storage	ipfs-http-client	Backend client library to interact with IPFS for file storage.
Backend Development	Express	Web application framework for Node.js API server.
	Sequelize	ORM for interacting with SQL databases.
	pg	Database drivers for PostgreSQL.
	bcrypt / bcryptjs	Libraries for hashing passwords securely.
	jsonwebtoken	Implements JSON Web Tokens for authentication/authorization.

	Helmet	Middleware for setting security-related HTTP headers.

	CORS	Middleware to enable Cross-Origin Resource Sharing.
	Compression	Middleware to compress HTTP responses.
	Morgan	HTTP request logger middleware.
	Multer	Middleware for handling file uploads (multipart/form-data).
	form-data	Creates multipart/form-data streams (for uploads).
	dotenv	Loads environment variables from a .env file.
	Nodemon	Automatically restarts the Node.js server during development.
	Mocha & Chai	Testing framework and assertion library for backend tests.
Frontend Development	React	Core library for building the user interface.
	Material UI (MUI)	React component library for UI design.
	React Router	Handles client-side navigation.
	Redux Toolkit	Manages global application state.
	Axios	HTTP client for making API requests.
	Formik & Yup	Libraries for building and validating forms.
	qrcode.react	Generates and displays QR codes.

	react-dropzone	Facilitates file uploads.
	Recharts	Library for creating charts and visualizations.
	Leaflet & react-leaflet	Libraries for interactive maps.
	Socket.IO Client	Enables real-time communication.
	Date-fns	Utility library for date manipulation.
	Web Vitals	Measures frontend performance metrics.
	React Scripts	Provides development scripts for Create React App.
	Testing Library	Used for testing React components.
	ESLint	Linter for code quality and style enforcement

Table 3.2 Software/Tools Required

3.2 System Design Diagram/Equation

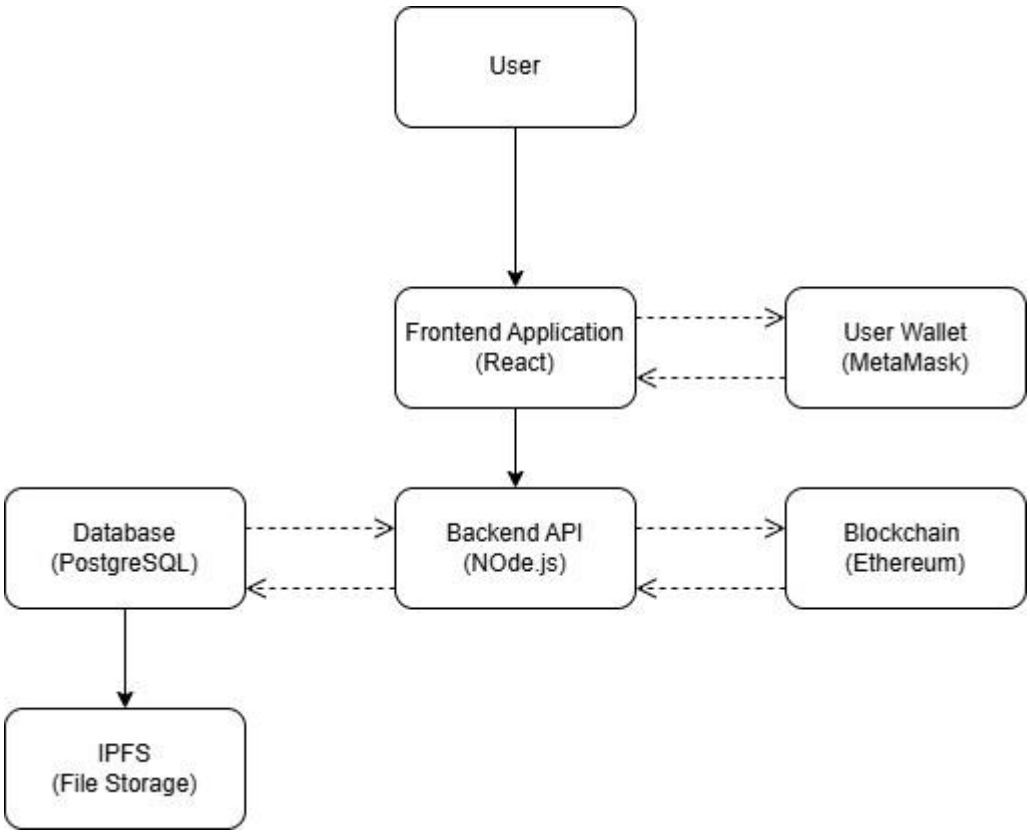

Figure 3.2.1 System Design Diagram

This diagram provides a high-level overview of the main components and their interactions within the supply chain management system.

1. User: Interacts with the system through the Frontend Application.
2. Frontend Application (React): The user interface built with React. It communicates with:
    a) Backend API: Sends requests for data processing, user management, and potentially blockchain interactions.
    b) User Wallet (MetaMask): Connects to the user's Ethereum wallet (like MetaMask) to sign blockchain transactions.
3. Backend API (Node.js): The server-side application. It interacts with:
    a) Database: Stores off-chain data like user information, potentially cached blockchain data (PostgreSQL Database)
    b) Blockchain: Sends transactions and reads data from the smart contracts.

c) IPFS: Stores larger files like images or metadata documents, linking them via URIs in the blockchain data.

4. User Wallet (MetaMask): Used by the user to authorize and sign transactions initiated by the Frontend Application before they are sent to the Blockchain.

5. Blockchain (Ethereum/Sepolia): The decentralized ledger where all supply chain data (watches, components, certifications, warranties, ownership) is immutably recorded through smart contracts. It interacts with both the Frontend (for reading data) and the Backend API (for reading and writing data).

6. Database: Centralized storage managed by the Backend API.

7. IPFS (InterPlanetary File System): Decentralized storage for off-chain metadata and files.

3.2.1 System Architecture Diagram

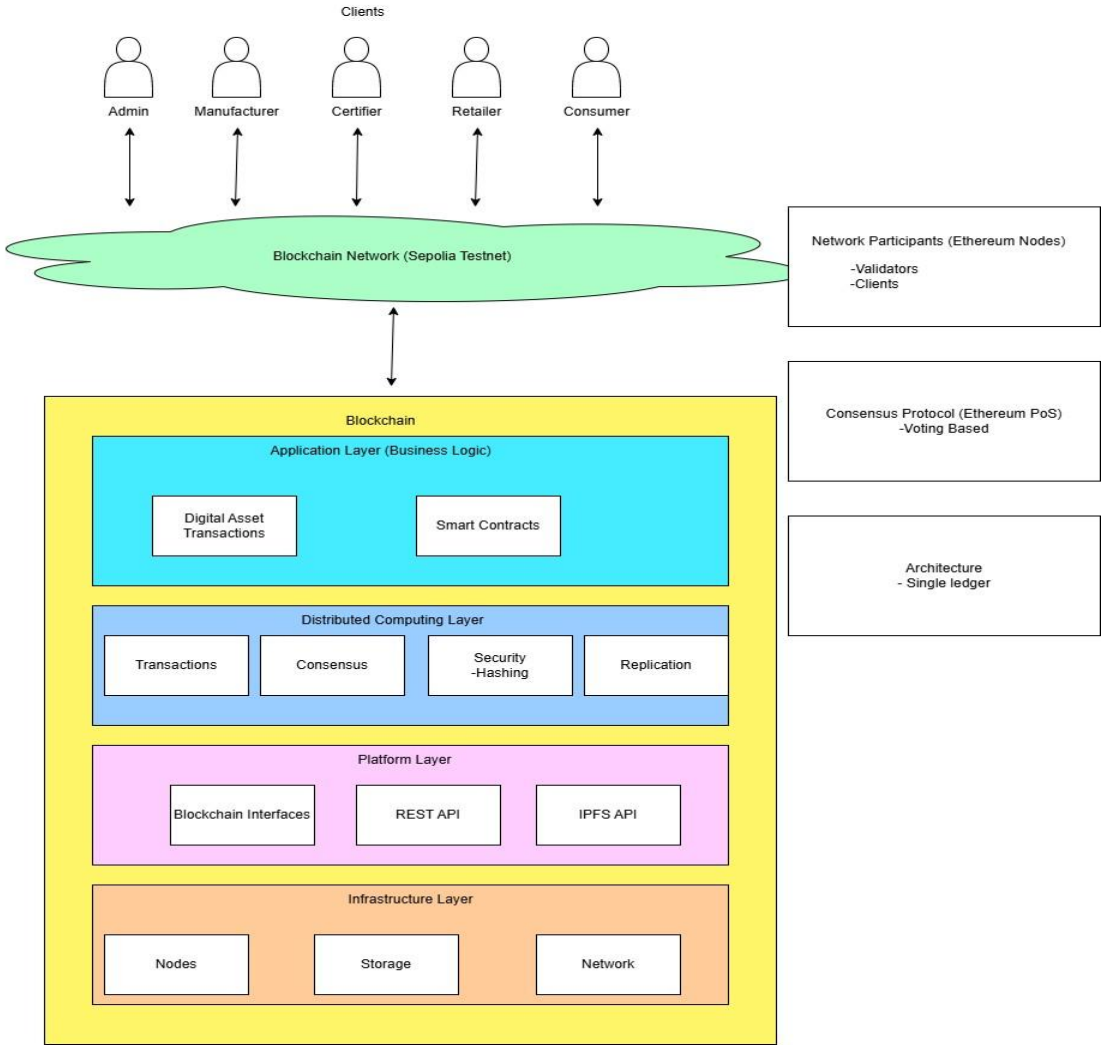

Figure 3.2.1.1 System Architecture Diagram

This architecture illustrates a layered approach for blockchain-based supply chain management system that running on the Sepolia Testnet. At the top, various Clients, including Admins, Manufacturers, Certifiers, Retailers, and Consumers, interact with the system. These users connect through applications (like your React frontend) to the underlying Blockchain Network (Sepolia Testnet), which serves as the communication backbone for accessing the decentralized ledger and its services.

The core Blockchain itself is structured into distinct layers. The uppermost is the Application Layer, which houses the specific Business Logic for your supply chain tracking. This includes the various Smart Contracts (like WatchRegistry, ComponentTracking, etc.) that define the rules for managing watches, components, certifications, and warranties, as well as handling Digital Asset Transactions such as the transfer of watch ownership. Below this lies the Distributed Computing Layer, which represents the fundamental operations provided by the Ethereum network itself. This layer manages the processing of Transactions, ensures network agreement through its Consensus mechanism (Ethereum's Proof-of-Stake), provides Security through cryptographic methods like Hashing, and ensures data integrity via Replication across participating nodes.

Further down, the Platform Layer provides the necessary interfaces for interaction. Blockchain Interfaces (like RPC used by ethers.js) allow applications to communicate directly with the blockchain nodes. Additionally, this layer includes interfaces for off-chain components, such as the REST API exposed by your backend system for user management and potentially orchestrating complex workflows, and the IPFS API used for storing and retrieving metadata files linked from the smart contracts. Finally, the Infrastructure Layer represents the foundational elements of the blockchain network, comprising the physical or virtual Nodes running the client software, the Storage systems holding the ledger data, and the underlying communication Network connecting everything.

Supporting this layered structure are key concepts defining the network's operation. The Network Participants consist of Validators, who run nodes and participate in the consensus process by staking ETH, and Clients, who use the network to read data or submit transactions. The Consensus Protocol employed is Ethereum's Proof-of-Stake (PoS), which is fundamentally Voting Based, relying on validators attesting to the validity of blocks. The

overall Architecture is Single-ledger based, meaning all participants operate on a single, consistent version of the distributed ledger maintained by the Ethereum network.

3.2.2 Use Case Diagram

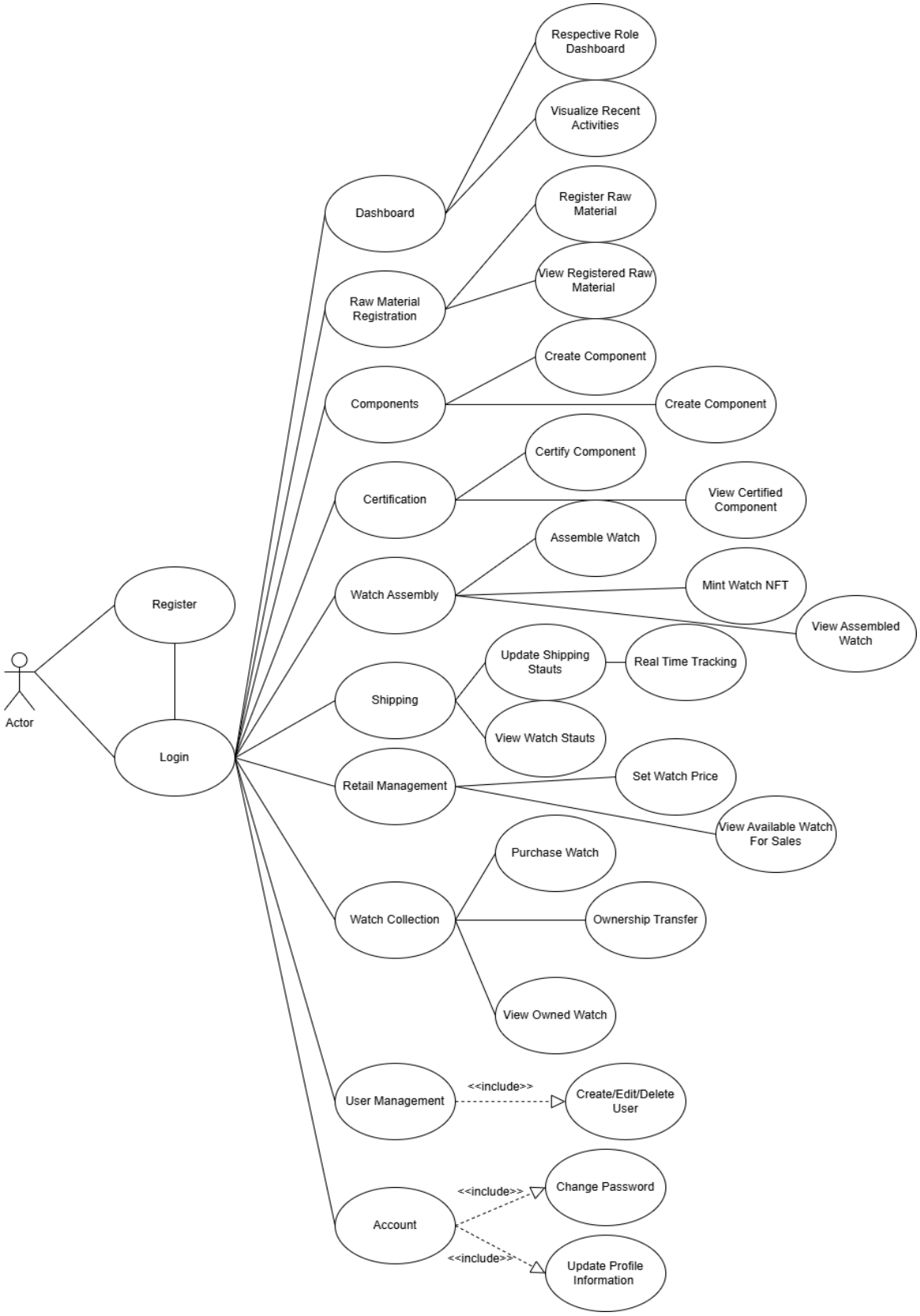

Figure 3.2.2.1 Use Case Diagram

The use case diagram illustrates the comprehensive functionality of the blockchain-based luxury watch supply chain management system with integrated NFT capabilities. The system demonstrates a complete end-to-end solution that encompasses all stakeholders in the luxury watch ecosystem, from raw material suppliers to end consumers.

Actor Analysis

Primary Actor: User

- Represents all system participants across the supply chain
- Single actor pattern with role-based access control implementation
- Multi-stakeholder engagement through differentiated dashboard interfaces

Core Use Case Categories

1. Authentication & Access Management

Register Use Case:

- **Purpose**: New user onboarding with role assignment
- **Backend Implementation**: POST /register endpoint in postgres.js
- **Database Integration**: User credentials stored in users table with wallet address verification
- **Security Features**: Password encryption, role validation, wallet integration

Login Use Case:

- **Purpose**: Secure system access with role-based routing
- **Backend Implementation**: POST /login endpoint with JWT token generation
- **Authentication Flow**: Username/password validation against PostgreSQL database
- **Session Management**: JWT-based stateless authentication with role-based authorization

2. Dashboard & Visualization System

Dashboard Use Case:

- **Respective Role Dashboard**: Dynamic interface generation based on user role
- **Implementation**: dashboardConfigs.js with 7 distinct role configurations
- **Visualize Recent Activities**: Real-time activity feeds and system status updates
- **Performance Metrics**: Role-specific KPIs and operational statistics

3. Supply Chain Management Use Cases

Raw Material Registration:

- **Register Raw Material**: Origin tracking with geological verification
- **Backend Endpoint**: POST /raw-material with blockchain integration

- **Smart Contract Function**: registerRawMaterial() in LuxuryWatchNFT.sol
- **View Registered Raw Material**: Complete material inventory with provenance data

Components Management:

- **Create Component**: Manufacturing workflow with material linkage
- **Backend Implementation**: POST /component with raw material validation
- **Smart Contract Integration**: createComponent() function with status tracking
- **Database Relations**: Foreign key relationships between components and raw materials

Certification Process:

- **Certify Component**: Independent quality assurance workflow
- **Backend Endpoint**: POST /certify-component with certifier validation
- **Smart Contract Function**: certifyComponent() with immutable attestation
- **View Certified Component**: Quality assurance audit trails and certification status

4. Watch Assembly & NFT Integration

Watch Assembly:

- **Assemble Watch**: Multi-component integration with validation
- **Backend Implementation**: POST /watch with comprehensive component verification
- **Smart Contract Function**: assembleWatch() with minimum component requirements
- **Mint Watch NFT**: Automatic ERC-721 token generation with metadata
- **IPFS Integration**: Dynamic metadata creation with supply chain attributes
- **View Assembled Watch**: Complete watch profile with component traceability

5. Distribution & Logistics

Shipping Management:

- **Update Shipping Status**: Real-time logistics tracking
- **Backend Endpoint**: POST /update-shipping with status validation
- **Smart Contract Function**: updateShippingStatus() with immutable tracking
- **Real Time Tracking**: GPS integration and delivery confirmation
- **View Watch Status**: Complete shipping history and current location

6. Retail & Sales Management

Retail Management:

- **Set Watch Price**: Dynamic pricing with market integration
- **Backend Implementation**: POST /mark-available with price validation
- **Smart Contract Function**: markAvailableForSale() with retail authorization

- **View Available Watch For Sales**: Marketplace integration with filtering capabilities
- **Purchase Watch**: Consumer transaction processing with ownership transfer

7. Ownership & Collection Management

Watch Collection:

- **Ownership Transfer**: Peer-to-peer transfer with blockchain verification
- **Smart Contract Function**: transferOwnership() with validation
- **NFT Integration**: Automatic token transfer with ownership synchronization
- **View Owned Watch**: Personal collection management with authentication history

8. Administrative Functions

User Management:

- **Create/Edit/Delete User**: Administrative user lifecycle management
- **Backend Implementation**: CRUD operations with role validation
- **Database Operations**: User table management with audit trails
- **Access Control**: Admin-only functionality with permission validation

Account Management:

- **Change Password**: Secure credential updates with validation
- **Update Profile Information**: User profile management with verification
- **Include Relationships**: Integrated functionality with user authentication system

3.2.3 Activity Diagram

Register and Login

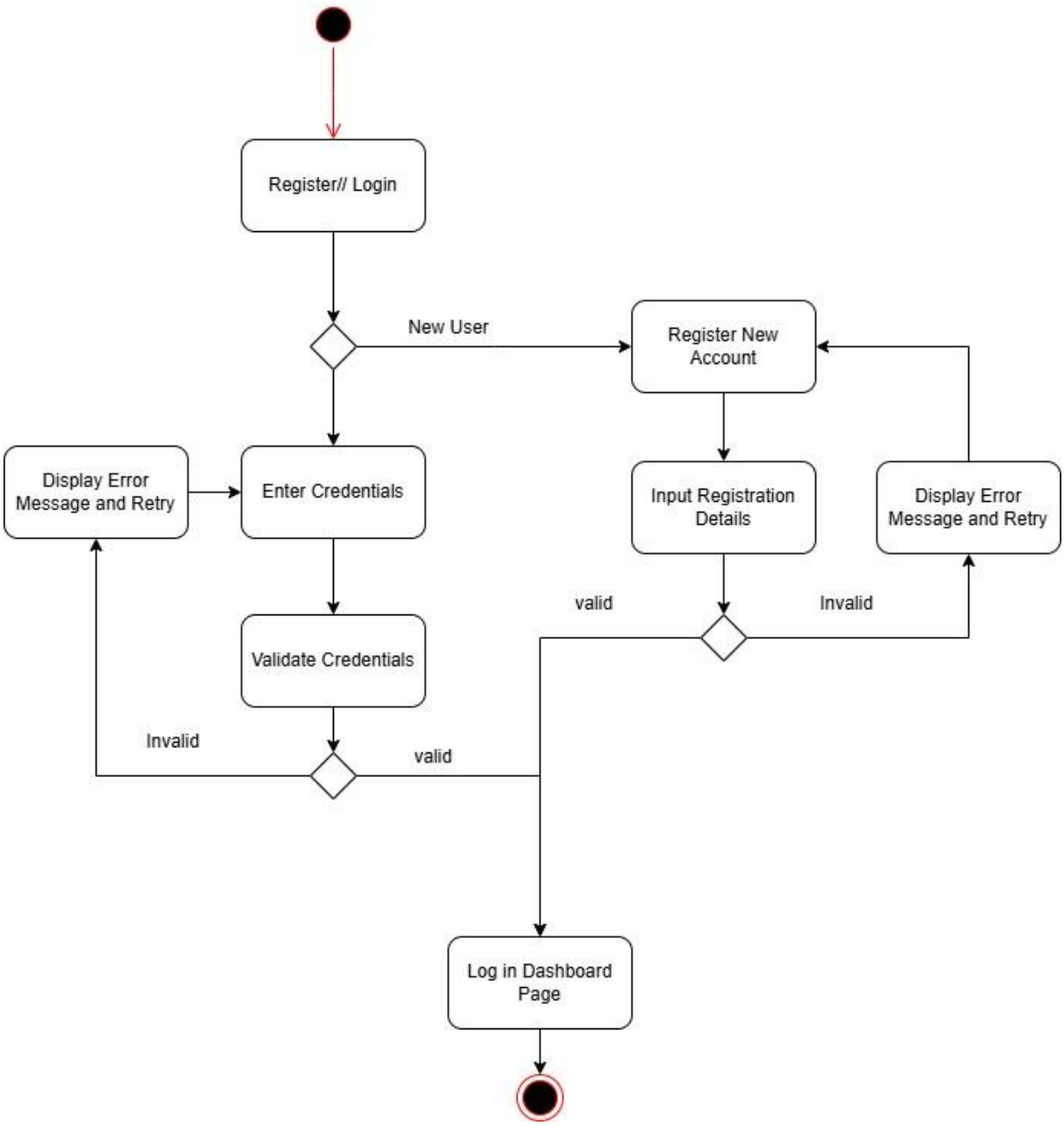

Figure 3.2.3.1 Activity Diagram of Register and User Login

This Activity Diagram represents the process flow for Register/Login functionality in a system. The process begins when the user is prompted with the option to either Register a new account or Login to an existing one. If the user selects to Login, they are directed to enter their credentials (such as username and password). Once entered, the system validates these credentials. If the credentials are valid, the user is successfully logged in and directed to the

Dashboard page. However, if the credentials are invalid, an error message is displayed, and the user is prompted to retry entering the correct information. On the other hand, if the user is a New User and selects to Register, they must input their registration details, including necessary fields like name, email, and password. Once the details are entered, the system checks whether the provided registration information is valid. If valid, the system proceeds to register the user and grants access to the Login screen, where the user can then log in. If the registration details are invalid, an error message is shown, and the user is asked to correct and retry the information. The process concludes when the user successfully logs in or completes the registration process, ultimately reaching the Dashboard page.

Profile Setting

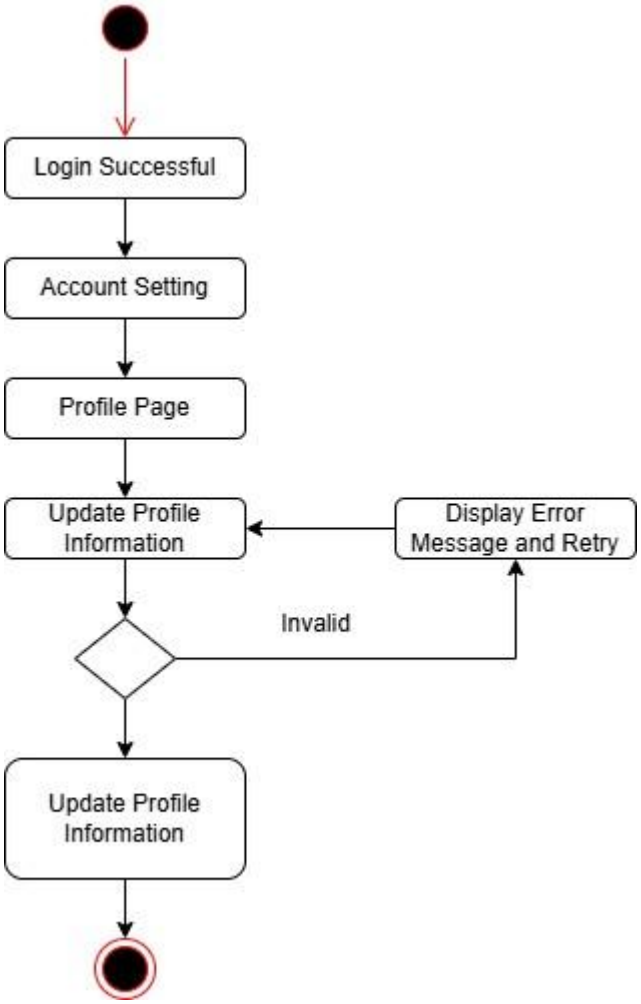

Figure 3.2.3.2 Activity Diagram of Profile Setting

This Activity Diagram illustrates the process flow for updating a user's profile information after a successful login. The process begins when the user logs in successfully, which leads them to the Account Setting page. From there, they are directed to the Profile Page, where they can choose to Update Profile Information. Once the user attempts to update their profile information, the system validates the input. If the information provided is valid, the user can successfully update their profile. However, if the information is invalid (e.g., missing required fields or incorrect data), an error message is displayed, prompting the user to retry the update process. This ensures that the user can only submit correct and complete information, maintaining the integrity of the profile data.

Raw Material Registration

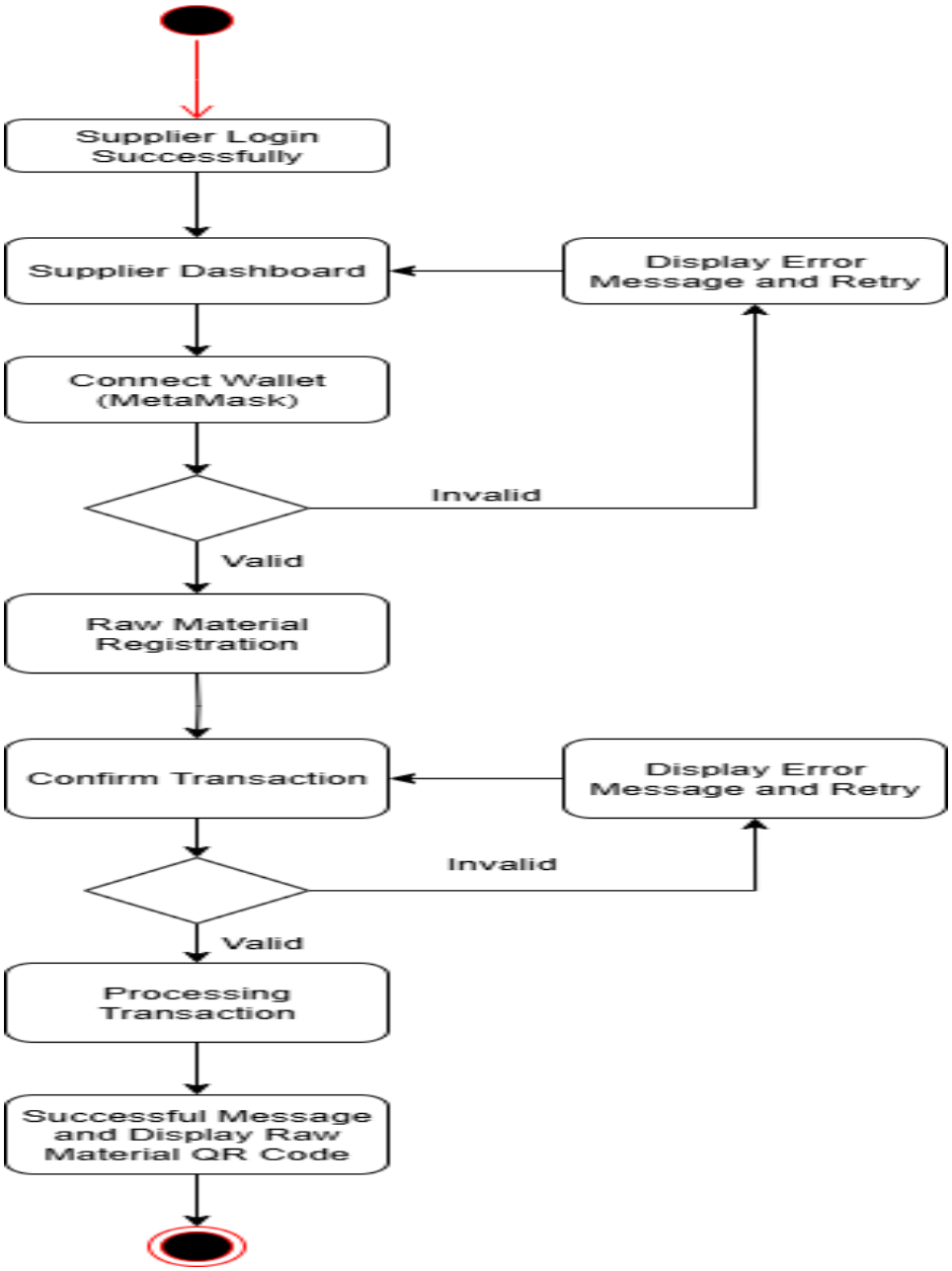

Figure 3.2.3.3 Activity Diagram of Raw Material Registration

This activity diagram shows the complete workflow for a **Supplier** to register raw materials in the luxury watch supply chain system.

Process Flow:

1. Authentication & Access

- Supplier logs in successfully and reaches their dashboard

- System validates their supplier role and credentials

2. Wallet Connection

- Supplier connects their MetaMask wallet to enable blockchain functionality
- System validates the wallet connection
- If invalid, shows error message and allows retry

3. Raw Material Registration

- Supplier fills out material details
- System prepares the registration data for blockchain submission

4. Transaction Processing

- System validates the transaction details
- If invalid, displays error and allows retry
- If valid, processes the blockchain transaction

5. Completion

- Successfully creates an immutable record on the blockchain
- Generates a QR code for the registered material
- Displays success message to the supplier

Create Component

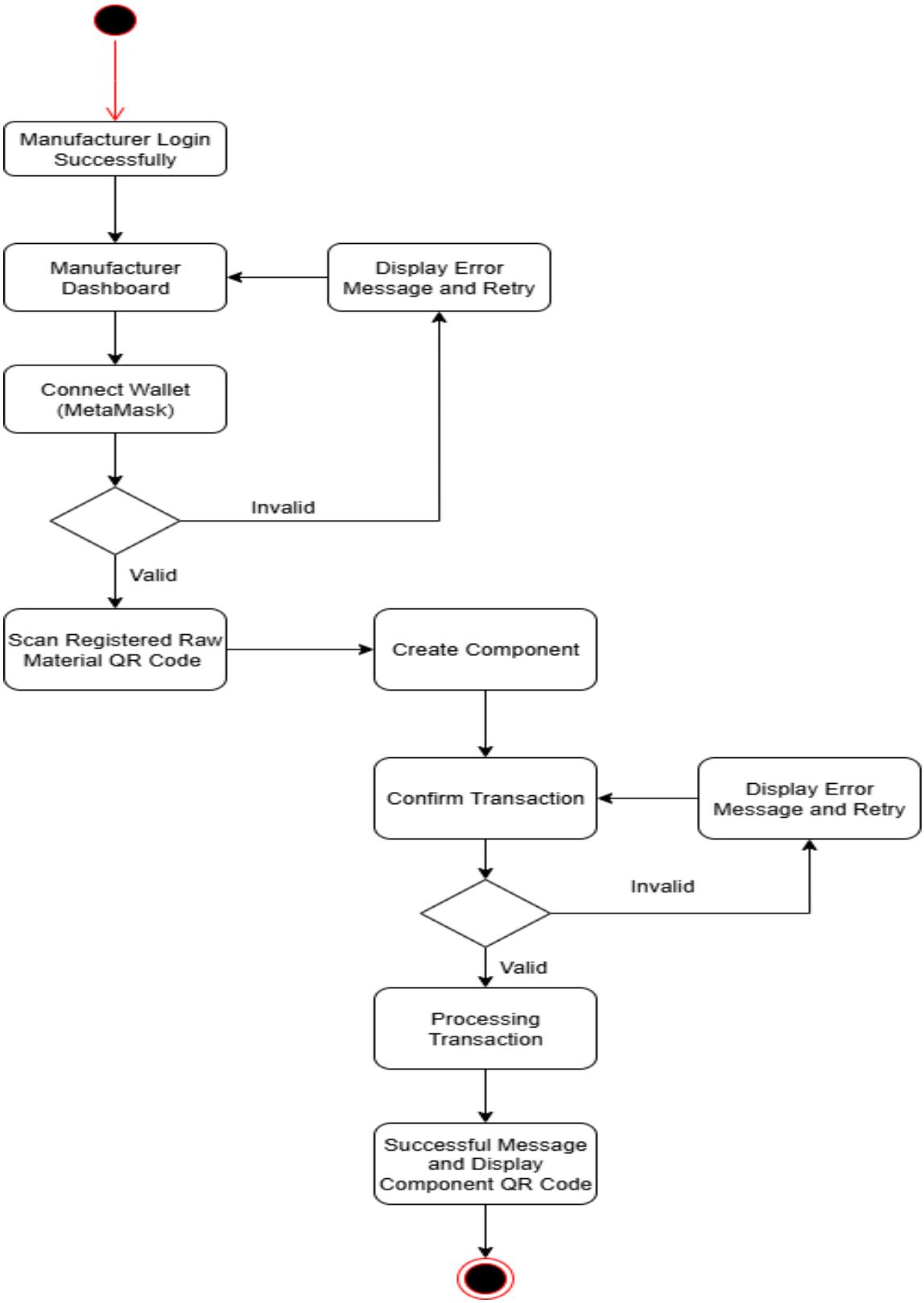

Figure 3.2.3.4 Activity Diagram of Create Component

This activity diagram shows the workflow for a **Manufacturer** to create components from registered raw materials in the supply chain system.

Process Flow:

1. Authentication & Access

- Manufacturer logs in successfully and accesses their dashboard
- System verifies manufacturer role and permissions

2. Wallet Connection

- Manufacturer connects MetaMask wallet for blockchain transactions
- System validates wallet connection
- Error handling with retry option if connection fails

3. Raw Material Verification

- Manufacturer scans a QR code from previously registered raw materials
- This links the new component to its source material for traceability

4. Component Creation

- Manufacturer enters component details
- System prepares component data for blockchain registration

5. Transaction Processing

- System validates transaction details
- Error handling with retry if validation fails
- Processes blockchain transaction when valid

6. Completion

- Creates immutable component record on blockchain
- Generates new QR code for the manufactured component
- Displays success confirmation to manufacturer

Certification Component

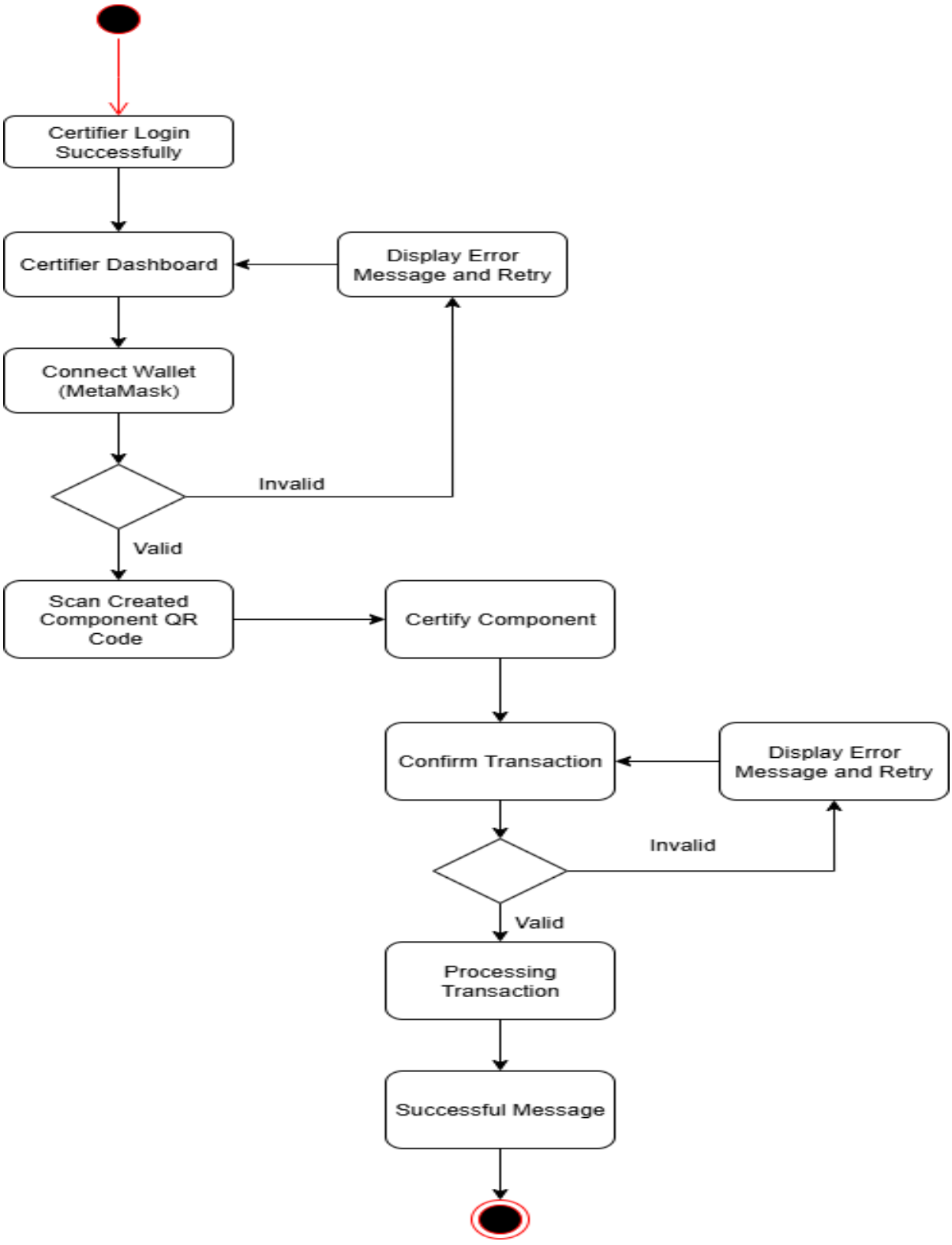

Figure 3.2.3.5 Activity Diagram of Certification Component

This activity diagram shows the workflow for a **Certifier** to perform quality assurance and certify manufactured components in the supply chain system.

Process Flow:

1. Authentication & Access

- Certifier logs in successfully and accesses their specialized dashboard
- System verifies certifier role and quality assurance permissions

2. Wallet Connection

- Certifier connects MetaMask wallet for blockchain certification transactions
- System validates wallet connection with error handling and retry option

3. Component Identification

- Certifier scans QR code of a created component (from previous manufacturer step)
- This identifies which component needs quality inspection and certification

4. Quality Certification Process

- Certifier performs quality inspection and enters certification details
- System prepares certification data for blockchain recording

5. Transaction Processing

- System validates certification transaction
- Error handling with retry mechanism if validation fails
- Processes blockchain transaction when validation passes

6. Completion

- Records immutable certification on blockchain
- Updates component status to "CERTIFIED"
- Displays success confirmation to certifier

Assembly Watch

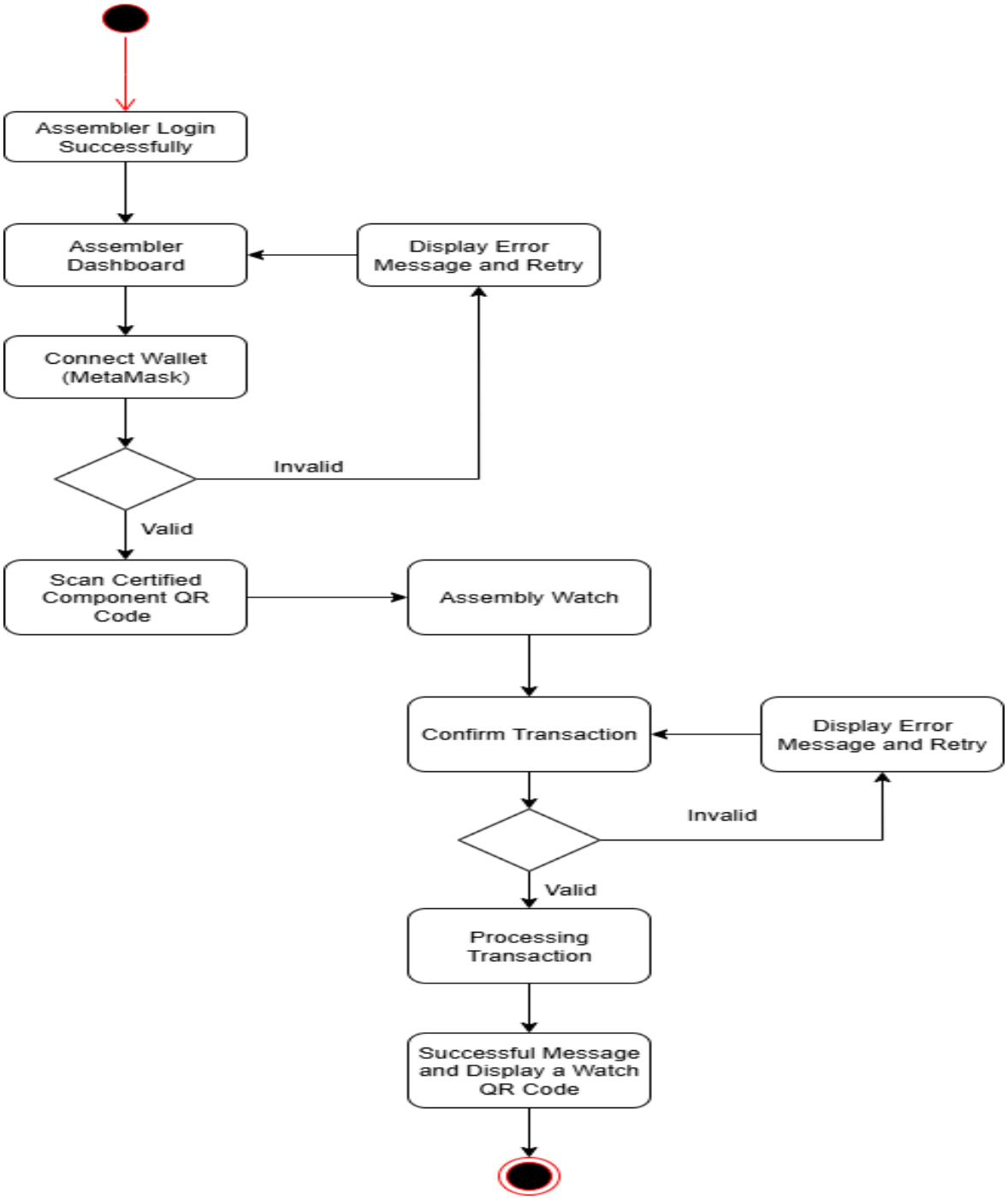

Figure 3.2.3.6 Activity Diagram of Assembly Watch

This activity diagram shows the workflow for an **Assembler** to create complete luxury watches from certified components and generate NFTs in the supply chain system.

Process Flow:

1. Authentication & Access

- Assembler logs in successfully and accesses their assembly dashboard
- System verifies assembler role and watch assembly permissions

2. Wallet Connection

- Assembler connects MetaMask wallet for blockchain transactions
- System validates wallet connection with error handling and retry capability

3. Component Verification

- Assembler scans QR codes of certified components (multiple components required)
- System verifies all components are properly certified and available for assembly

4. Watch Assembly Process

- Assembler combines certified components into a complete luxury watch
- System records assembly details and prepares for blockchain registration
- **NFT Generation:** Creates digital twin of the physical watch with metadata

5. Transaction Processing

- System validates assembly transaction and NFT minting
- Error handling with retry mechanism for failed transactions
- Processes blockchain transaction when validation passes

6. Completion

- Records immutable watch assembly on blockchain
- **Mints unique NFT** representing the luxury watch
- Displays success confirmation with watch and NFT details

Update Shipping Status

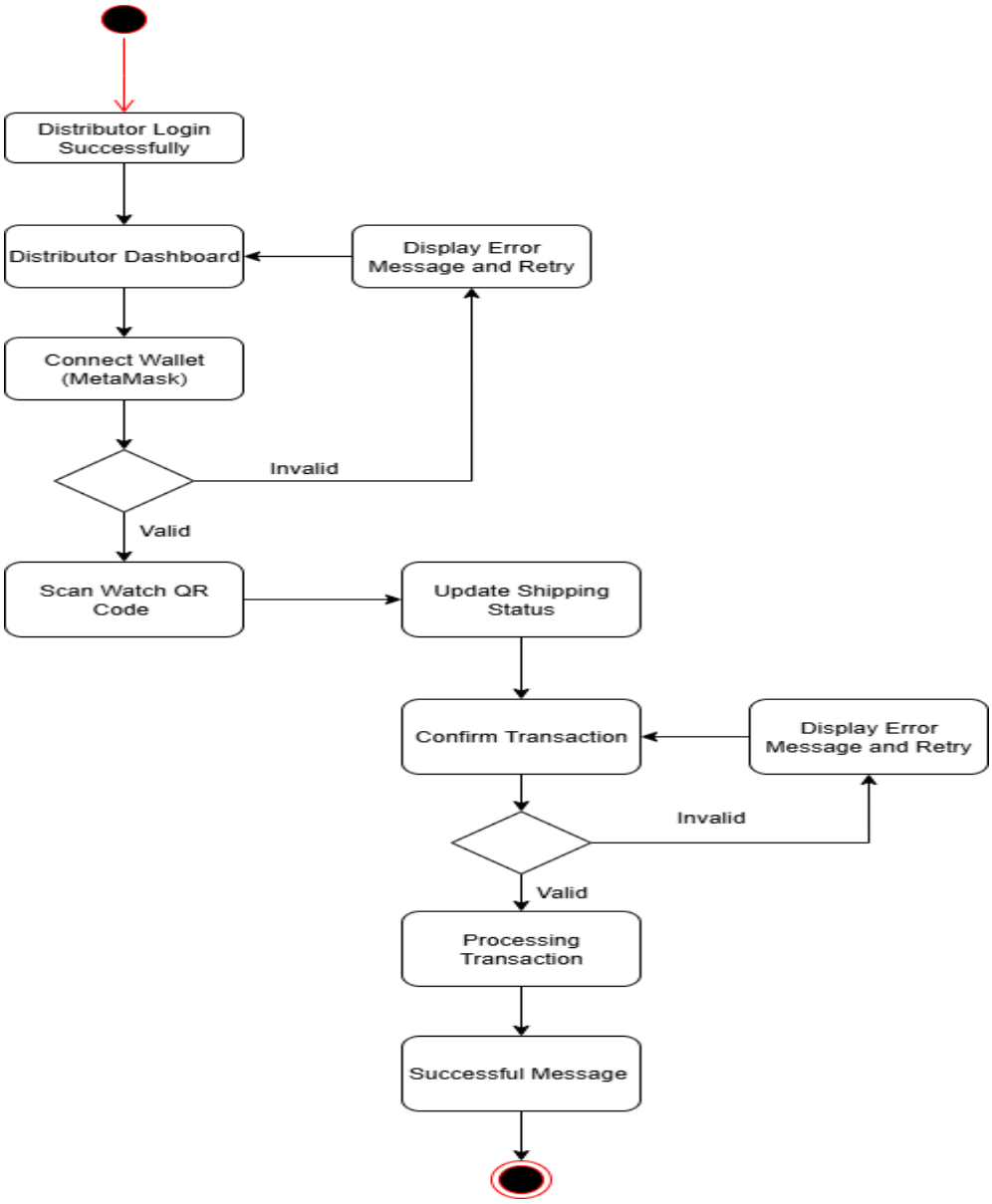

Figure 3.2.3.7 Activity Diagram of Update Shipping Status

This activity diagram shows the workflow for a **Distributor** to manage shipping and logistics tracking for assembled luxury watches in the supply chain system.

Process Flow:

1. Authentication & Access

- Distributor logs in successfully and accesses their logistics dashboard
- System verifies distributor role and shipping management permissions

2. Wallet Connection

- Distributor connects MetaMask wallet for blockchain shipping transactions

- System validates wallet connection with error handling and retry functionality

3. Watch Identification

- Distributor scans QR code of an assembled watch (from previous assembler step)
- System identifies which watch requires shipping status updates

4. Shipping Status Update

- Distributor enters shipping information (location, status, tracking details)
- System prepares shipping data for blockchain recording
- Updates could include: SHIPPED, IN_TRANSIT, DELIVERED

5. Transaction Processing

- System validates shipping transaction details
- Error handling with retry mechanism for failed validations
- Processes blockchain transaction when validation succeeds

6. Completion

- Records immutable shipping status on blockchain
- Updates watch location and delivery status
- Displays success confirmation with tracking information

Mark Watch Available For Sales

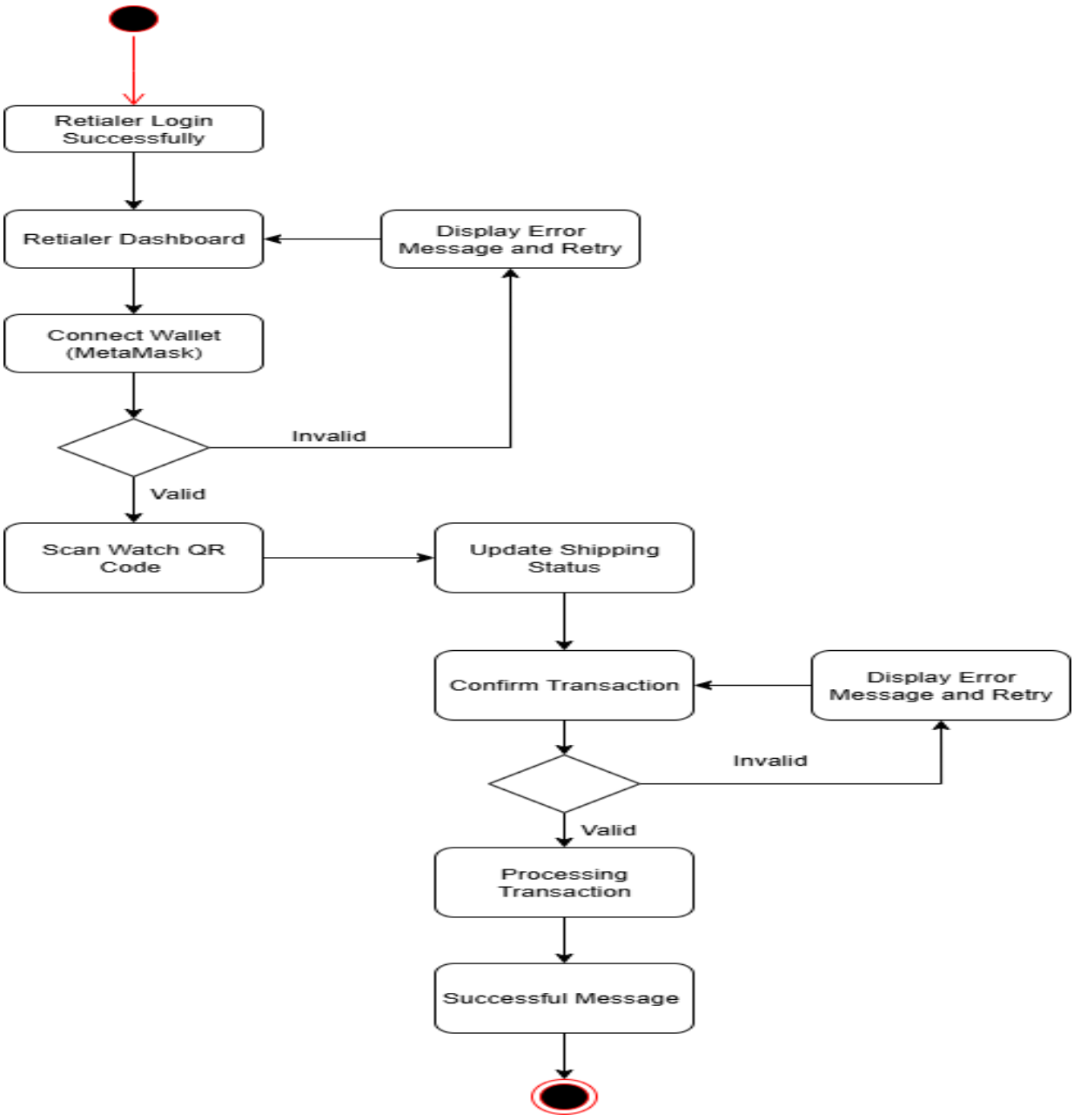

Figure 3.2.3.8 Activity Diagram of Mark Watch Available for Sales

This activity diagram shows the workflow for a **Retailer** to set pricing and make delivered luxury watches available for consumer purchase in the supply chain system.

Process Flow:

1. Authentication & Access

- Retailer logs in successfully and accesses their sales dashboard

- System verifies retailer role and sales management permissions

2. Wallet Connection

- Retailer connects MetaMask wallet for blockchain pricing transactions
- System validates wallet connection with error handling and retry option

3. Watch Identification

- Retailer scans QR code of a delivered watch (from previous distributor step)
- System identifies which watch will be priced and made available for sale

4. Pricing & Availability Setup

- Retailer sets the retail price for the luxury watch
- Marks the watch as "Available for Sales" in the marketplace
- System prepares pricing data for blockchain recording

5. Transaction Processing

- System validates pricing transaction details
- Error handling with retry mechanism for failed validations
- Processes blockchain transaction when validation passes

6. Completion

- Records immutable pricing information on blockchain
- Updates watch status to "Available for Sale"
- Displays success confirmation with marketplace listing details

Purchase Watch

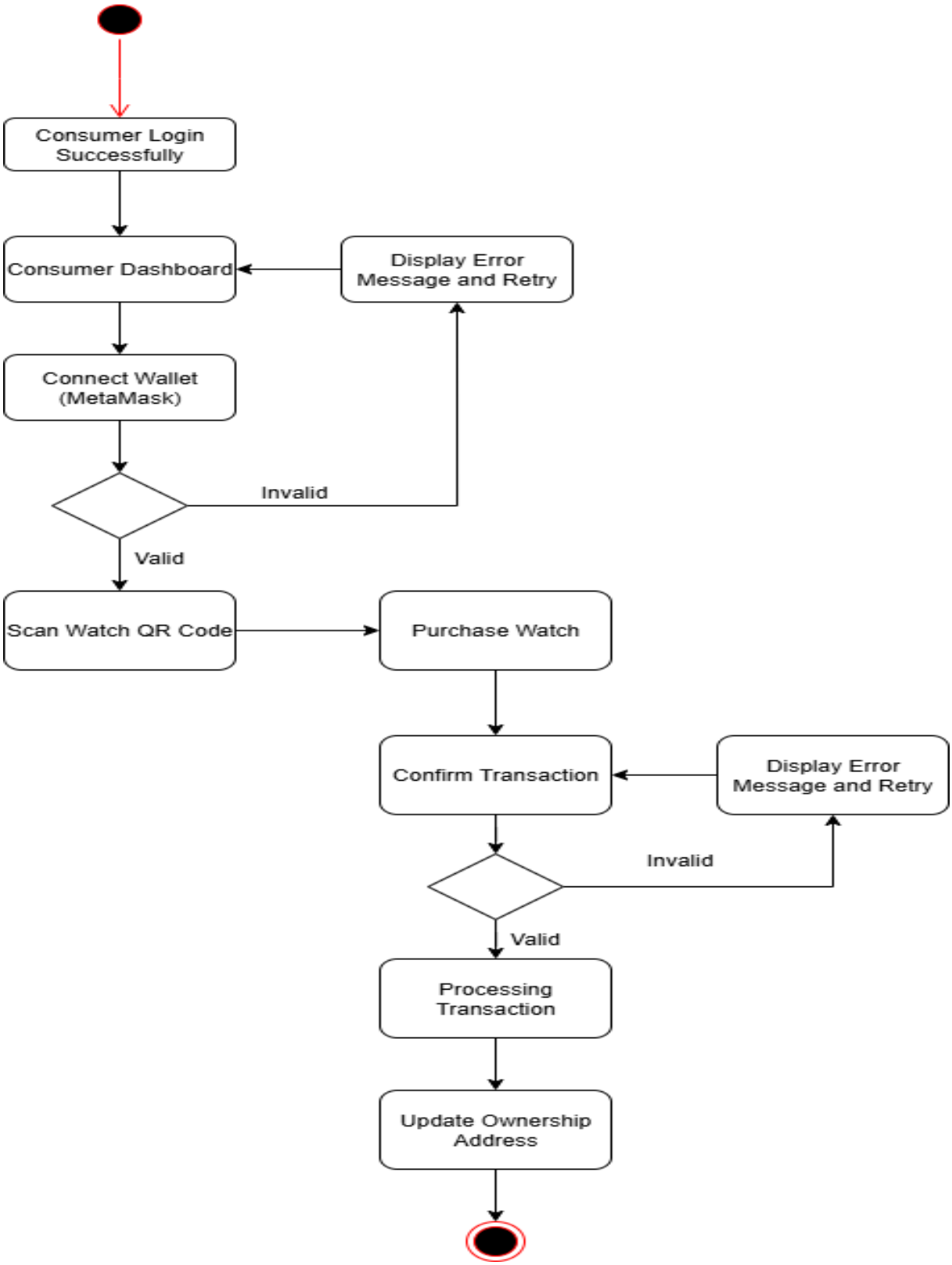

Figure 3.2.3.9 Activity Diagram of Purchase Watch

This activity diagram shows the workflow for a **Consumer** to purchase a luxury watch and complete the ownership transfer in the supply chain system.

Process Flow:

1. Authentication & Access

- Consumer logs in successfully and accesses their purchase dashboard
- System verifies consumer role and purchasing permissions

2. Wallet Connection

- Consumer connects MetaMask wallet for payment and NFT receipt
- System validates wallet connection with error handling and retry capability

3. Watch Selection

- Consumer scans QR code of an available watch (from previous retailer step)
- System identifies the specific watch available for purchase with pricing information

4. Purchase Initiation

- Consumer confirms purchase of the selected luxury watch
- System processes payment and prepares ownership transfer transaction

5. Transaction Processing

- System validates purchase transaction details
- Error handling with retry mechanism for failed payment or transfer
- Processes blockchain transaction when validation succeeds

6. Ownership Transfer

- Updates ownership address to consumer's wallet address
- **NFT Transfer:** Automatically transfers the watch's NFT to consumer
- Records immutable ownership change on blockchain

7. Completion

- Consumer now owns both the physical watch and its digital NFT certificate
- Complete supply chain journey ends with verified consumer ownership

Ownership Transfer

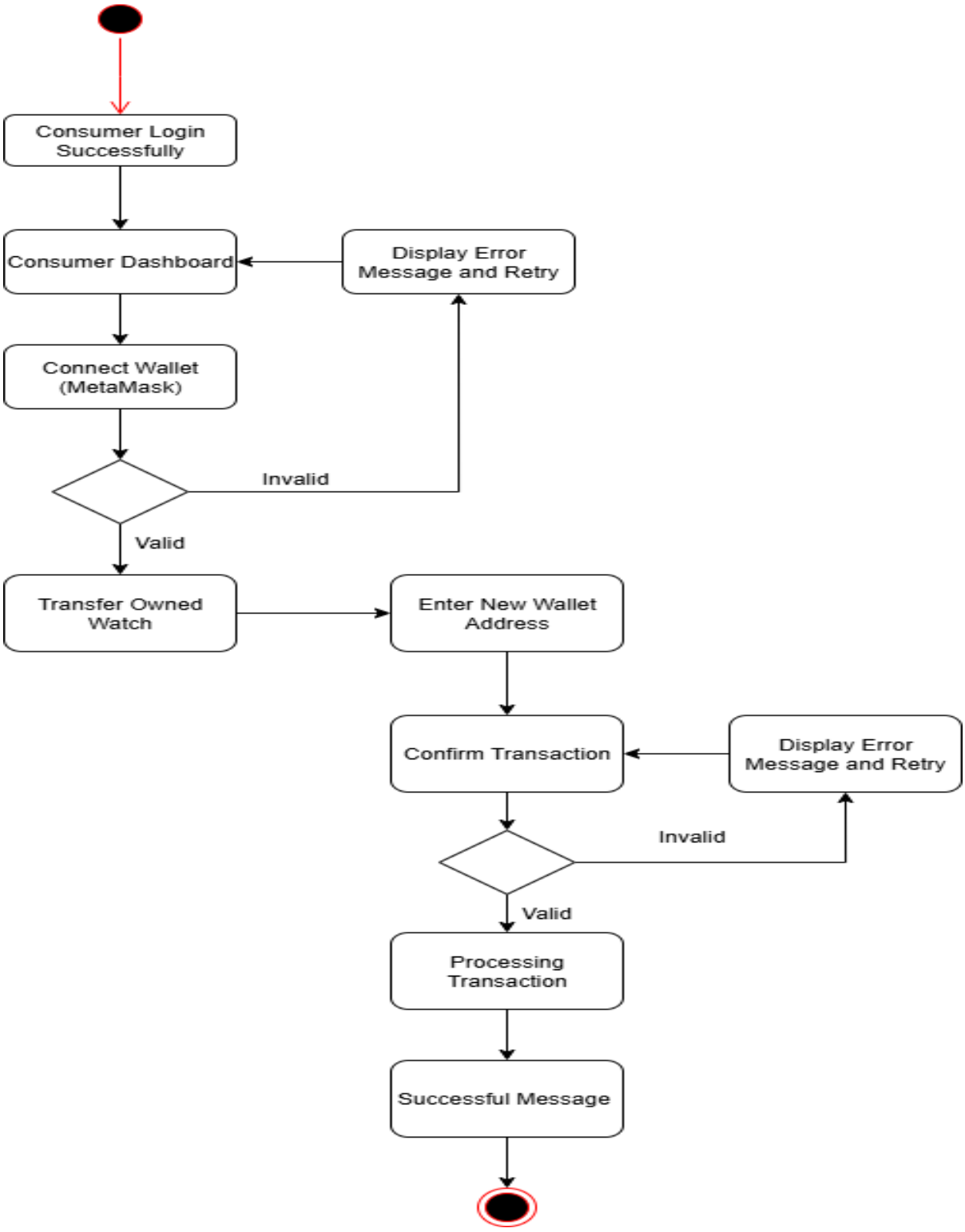

Figure 3.2.3.10 Activity Diagram of Ownership Transfer

This activity diagram shows the workflow for a **Consumer** to transfer ownership of their luxury watch to another person in the supply chain system.

Process Flow:

1. Authentication & Access

- Consumer (current owner) logs in successfully and accesses their dashboard
- System verifies consumer ownership and transfer permissions

2. Wallet Connection

- Consumer connects MetaMask wallet containing their watch NFT
- System validates wallet connection with error handling and retry functionality

3. Transfer Initiation

- Consumer selects "Transfer Owned Watch" from their collection
- System displays watches available for transfer from their ownership

4. Recipient Information

- Consumer enters the new wallet address of the intended recipient
- System validates the recipient's wallet address format and accessibility

5. Transaction Processing

- System validates transfer transaction details
- Error handling with retry mechanism for invalid addresses or failed transactions
- Processes blockchain transaction when validation succeeds

6. Completion

- Successfully transfers both physical watch ownership and NFT to new owner
- Updates blockchain records with new ownership information
- Displays confirmation message to both parties

Analysis of AI Model

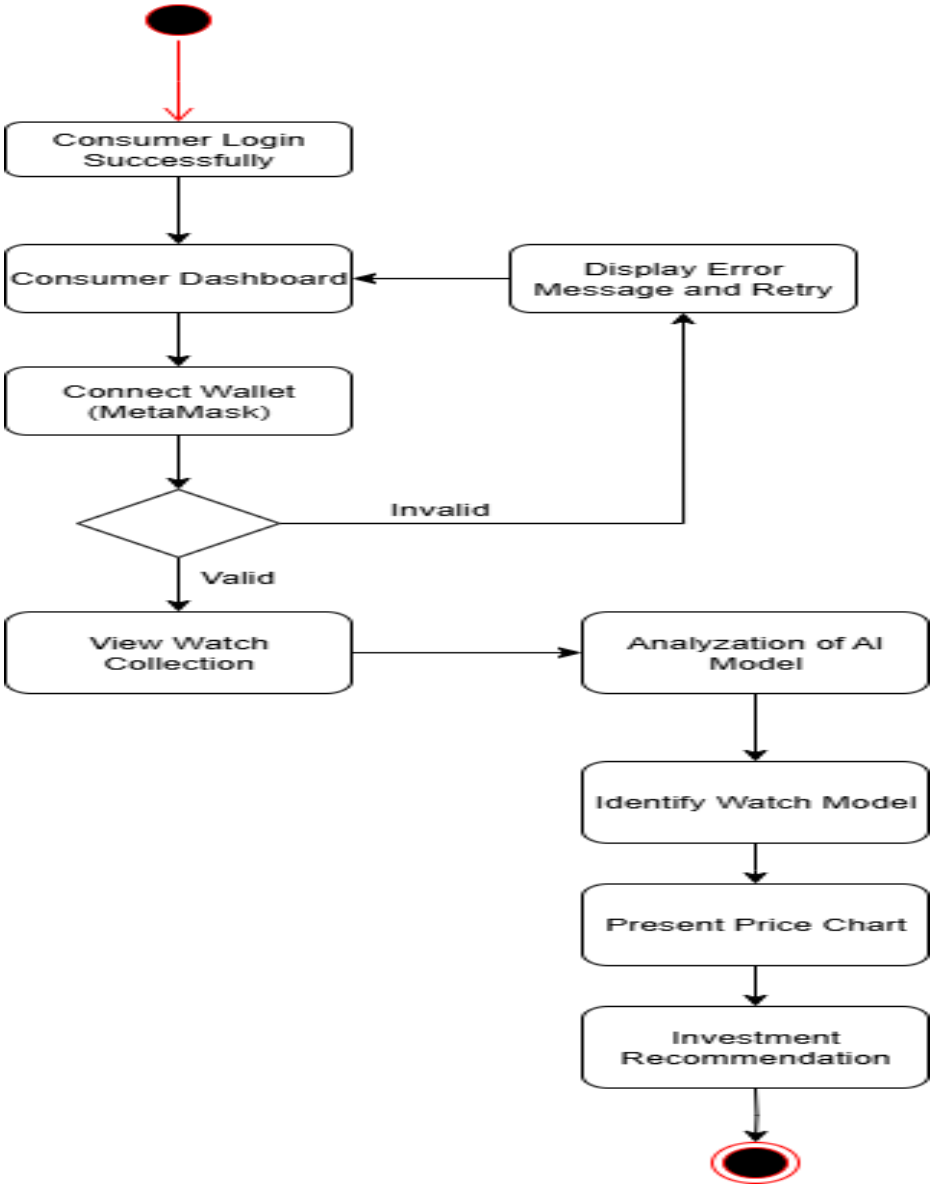

Figure 3.2.3.11 Activity Diagram of Analysis of AI Model

This activity diagram shows the workflow for a **Consumer** to view their luxury watch collection and receive AI-powered analysis and investment recommendations.

Process Flow:

1. Authentication & Access

- Consumer logs in successfully and accesses their dashboard
- System verifies consumer ownership and collection access permissions

2. Wallet Connection

- Consumer connects MetaMask wallet to access their NFT watch collection
- System validates wallet connection with error handling and retry capability

3. Collection Access

- Consumer selects "View Watch Collection" to see their owned luxury watches
- System displays all watches owned by the consumer's wallet address

4. AI Analysis Initiation

- System triggers "Analysis of AI Model" for the selected watches
- AI engine processes watch data and market information

5. Watch Identification

- AI system identifies specific watch models, brands, and characteristics
- Cross-references with watch databases and authentication records

6. Market Analysis

- System presents price charts showing historical and current market values
- Displays price trends, appreciation patterns, and market performance

7. Investment Insights

- AI generates personalized investment recommendations based on:
    - Watch rarity and market demand
    - Historical price performance
    - Market trends and forecasts
    - Portfolio diversification suggestions

3.2.4 Class Diagram

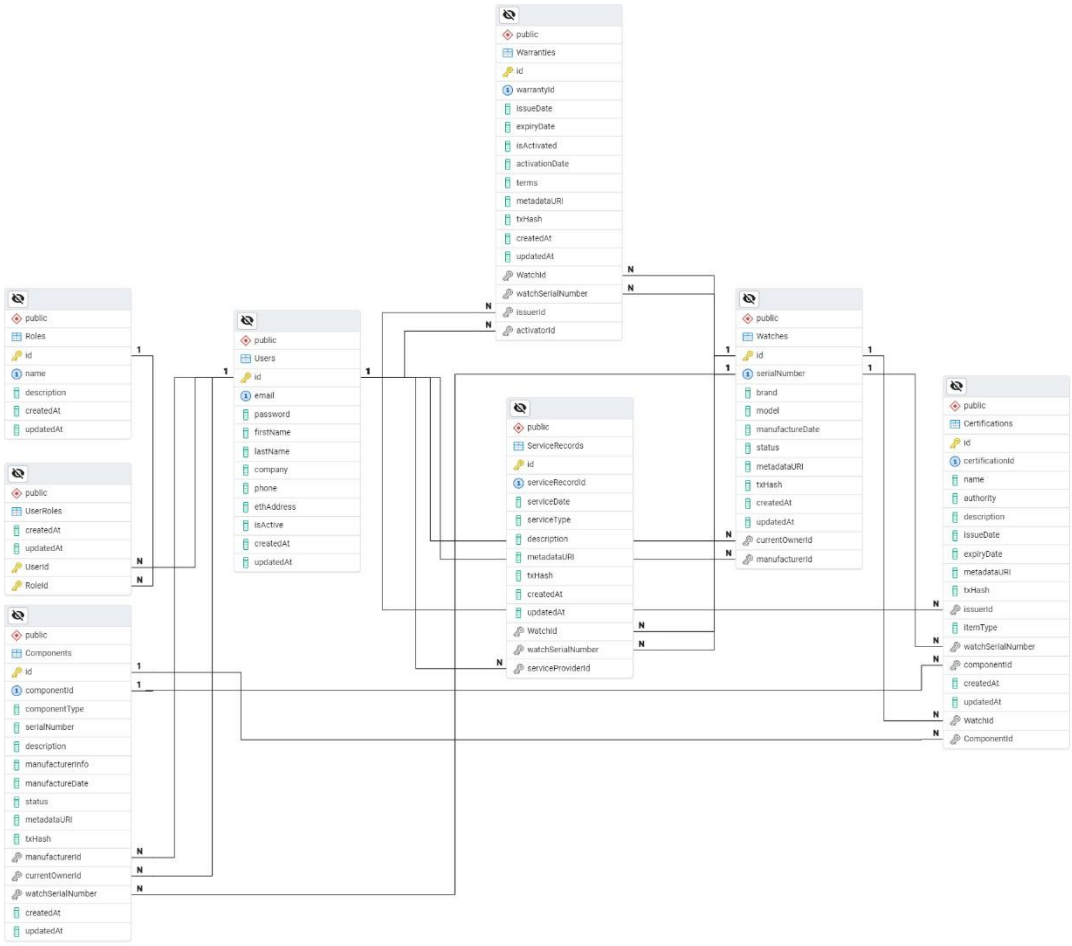

Figure 3.2.4.1 ERD Diagram

The Entity-Relationship Diagram (ERD) illustrates the structure of a relational database designed to track and manage watches, their components, service records, warranties, and certifications.

The Users table stores user-related information, such as email, password, and phone, and establishes relationships with the Roles table, defining the roles assigned to each user, such as admin or consumer. Each User can be associated with multiple Roles, representing the different levels of access and permissions within the system. The Watches table contains essential details about each watch, including serialNumber, brand, model, and currentOwnerId, which links a watch to its owner in the Users table. Watches are also associated with multiple Components, which are linked to the Watches table via a foreign key, detailing the individual parts of a watch, including their type, manufacturing details, and serialNumber.

The ServiceRecords table tracks maintenance or servicing events for each watch, capturing the type of service performed, the service date, and additional metadata. This table is related to both Watches and Users, identifying the watch serviced and the service provider. The Certifications table stores information about the official certification of the watch and its components, ensuring that the watch meets specific standards and is authentic. The Warranties table records warranty details for each watch, such as the warranty period and activation dates, and is linked to the Watches table.

Chapter 4: System Design

4.1 System Block Diagram

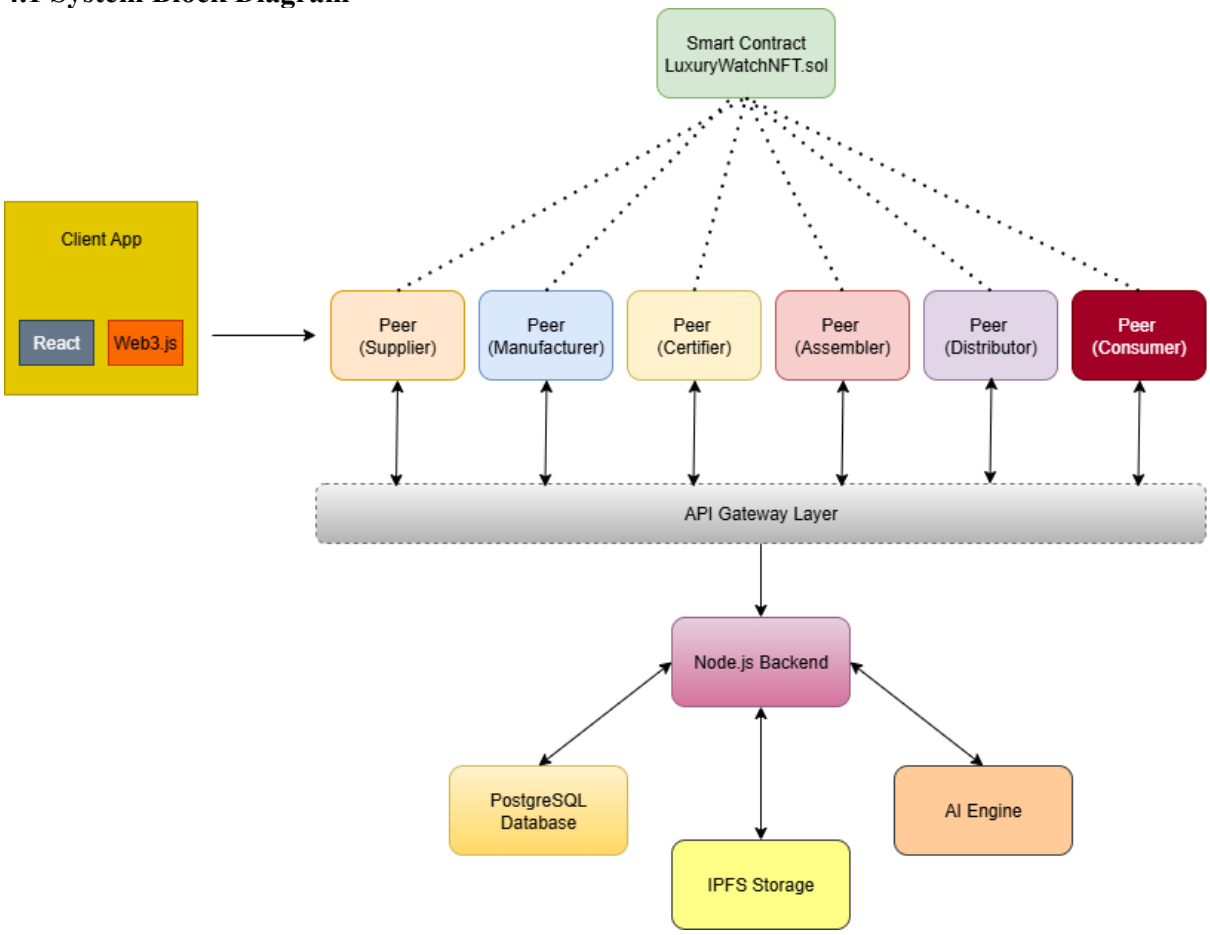

Figure 4.1.1 System Block Diagram

The presented system architecture diagram illustrates a comprehensive blockchain-based supply chain management solution specifically designed for luxury watch authentication and traceability. The architecture follows a distributed peer-to-peer model centered around the LuxuryWatchNFT.sol smart contract, which serves as the authoritative blockchain layer managing immutable records and NFT functionality. The smart contract maintains dotted connections to all stakeholder peers, indicating its role as the central truth source for supply chain data validation and NFT token management across the entire ecosystem.

The client application layer demonstrates a modern web-based approach, utilizing React.js for the user interface combined with Web3.js for blockchain connectivity. This configuration enables seamless interaction between traditional web applications and decentralized blockchain networks, allowing users to perform supply chain operations while maintaining cryptographic security through wallet integration. The client application serves as the primary

interface through which stakeholders interact with both the blockchain layer and backend services.

The peer network architecture encompasses six distinct stakeholder categories: Supplier, Manufacturer, Certifier, Assembler, Distributor, and Consumer. Each peer represents a specific role in the luxury watch supply chain, with color-coded differentiation indicating their unique functions and permissions within the system. The linear arrangement of these peers reflects the sequential nature of the supply chain process, where raw materials progress through manufacturing, certification, assembly, distribution, and ultimately reach consumers. The bidirectional arrows between peers and the API Gateway Layer indicate that each stakeholder can both submit data and retrieve information relevant to their role and authorization level.

The API Gateway Layer functions as a critical intermediary component, managing communication between the distributed peer network and the centralized backend services. This architectural pattern provides several advantages including load balancing, request routing, authentication management, and protocol translation between different system components. The gateway layer abstracts the complexity of backend service interactions while maintaining security through controlled access points and standardized communication protocols.

The Node.js Backend represents the core processing engine of the system, handling business logic, data validation, and orchestration of interactions between various system components. The backend architecture demonstrates integration with three essential services: PostgreSQL Database for structured data storage and complex relational queries, IPFS Storage for decentralized file storage and NFT metadata management, and an AI Engine for advanced analytics and watch authentication. This multi-service approach enables the system to leverage the strengths of different technologies while maintaining data consistency and operational efficiency.

The overall architecture exhibits characteristics of a hybrid blockchain system, combining the immutability and trust properties of blockchain technology with the performance and flexibility of traditional database systems. The integration of IPFS storage addresses the scalability limitations of on-chain data storage while maintaining decentralization principles. The inclusion of an AI Engine component suggests advanced capabilities for pattern recognition, fraud detection, and market analysis, enhancing the system's value proposition beyond basic supply chain tracking. This architectural design demonstrates a sophisticated understanding of enterprise blockchain implementation requirements, balancing decentralization benefits with practical operational needs.

4.2 Blockchain-Enabled Supply Chain Management and Digital Asset Tokenization Framework

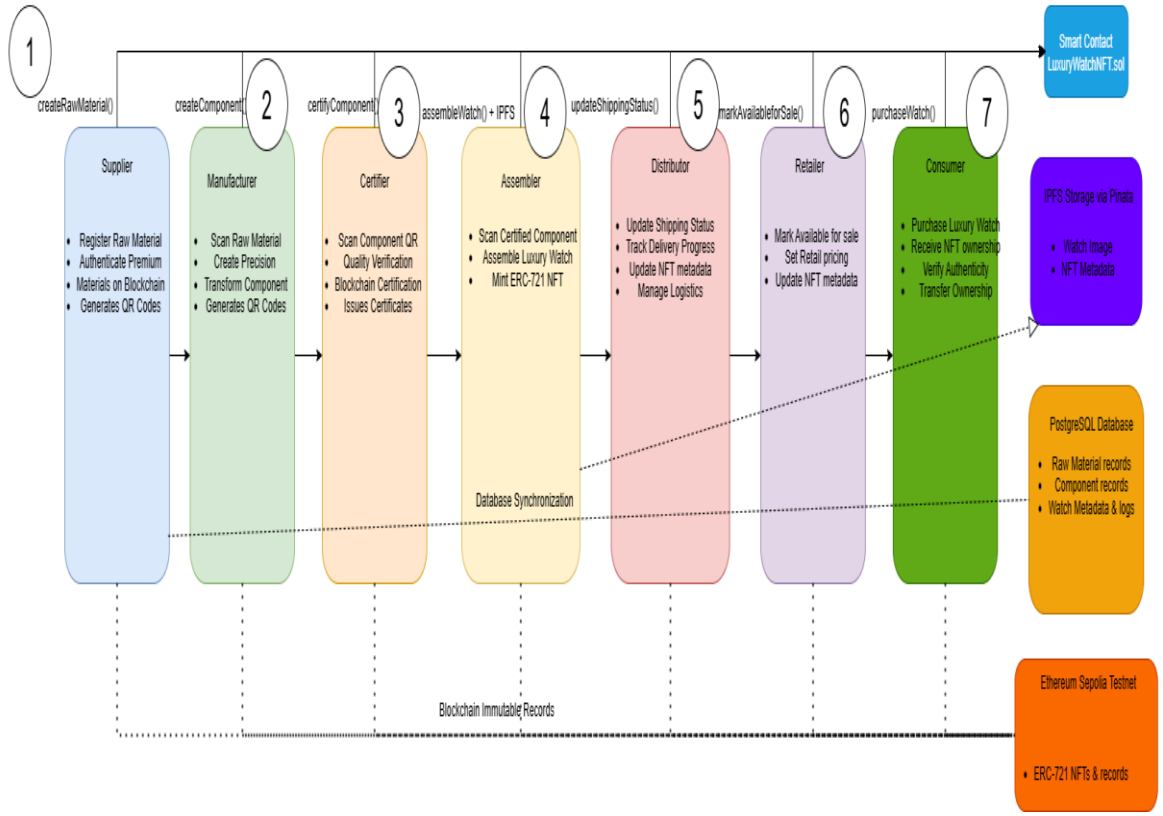

Figure 4.2.1 Project Workflow

This diagram illustrates a comprehensive digital transformation framework that integrates traditional supply chain management with distributed ledger technology and non-fungible token (NFT) protocols for luxury goods authentication. The system employs a seven-stage sequential workflow encompassing raw material sourcing, component manufacturing, quality certification, product assembly, distribution logistics, retail operations, and consumer acquisition. The architecture leverages a hybrid data management approach, utilizing PostgreSQL databases for operational efficiency while maintaining immutable records on the Ethereum blockchain infrastructure. The integration of ERC-721 NFT standards enables the creation of unique digital certificates that correspond to physical luxury watches, providing provenance verification and ownership authentication. This framework addresses critical challenges in luxury goods markets, including counterfeiting, supply chain opacity, and ownership verification, by creating an immutable digital thread that tracks each product's

journey from raw materials to final consumer purchase, thereby enhancing transparency, trust, and value proposition in high-value merchandise transactions.

4.3 Database Design and Implementation Strategy

The database architecture employs PostgreSQL as the primary relational database management system, chosen for its robust ACID compliance, advanced indexing capabilities, and excellent performance characteristics for complex query operations. The database schema design follows third normal form principles to minimize data redundancy while optimizing query performance for supply chain tracking operations. The schema comprises three primary entity tables representing raw materials, components, and assembled watches, with appropriate foreign key relationships maintaining referential integrity across the supply chain hierarchy.

The raw materials table serves as the foundational data structure, capturing essential information including material identification codes, material types, geographical origins, associated imagery, location data, timestamps, and supplier addresses. The components table extends this foundation by incorporating manufacturing details such as component types, serial numbers, raw material references, manufacturing locations, and quality certification status. The watches table represents the culmination of the supply chain process, aggregating component references, assembly information, shipping details, ownership records, and NFT-related metadata.

Database connection management utilizes connection pooling mechanisms to optimize performance and resource utilization. The implementation includes comprehensive error handling, transaction management, and automated backup procedures to ensure data integrity and system reliability. Additionally, the database design incorporates audit trail capabilities through timestamp tracking and modification logging, providing complete visibility into data changes throughout the system lifecycle.

4.4 Smart Contract Development and Blockchain Integration

The smart contract architecture represents a sophisticated implementation of supply chain management logic combined with NFT functionality on the Ethereum blockchain platform. The primary smart contract, LuxuryWatchNFT, inherits from OpenZeppelin's standardized ERC-721 implementation while extending functionality to support comprehensive supply chain operations. The contract design incorporates multiple data structures representing raw materials, components, and assembled watches, with state variables maintaining mappings between physical assets and their digital representations.

The smart contract implements a comprehensive set of functions corresponding to each stage of the supply chain process. Raw material registration functions enable suppliers to create immutable records of material origins and characteristics. Component creation functions allow manufacturers to establish digital representations of manufactured parts while maintaining linkages to their source materials. Quality certification functions provide mechanisms for third-party validators to attest to component quality and compliance standards. The watch assembly function represents the culmination of the supply chain process, automatically triggering NFT minting when components are successfully assembled into finished products.

The NFT implementation follows ERC-721 standards while incorporating custom metadata structures that capture comprehensive supply chain information. Token URI management enables dynamic metadata updates as watches progress through different ownership and shipping stages. The contract includes ownership transfer mechanisms that maintain supply chain integrity while enabling secondary market transactions. Gas optimization strategies have been implemented throughout the contract design to minimize transaction costs while maintaining functionality and security requirements.

4.5 Backend API Development and Service Architecture

The backend application programming interface serves as the central orchestration layer, coordinating interactions between frontend applications, database systems, blockchain networks, and external storage services. The API architecture follows RESTful design principles, implementing standard HTTP methods for resource manipulation while maintaining stateless operation characteristics. The Express.js framework provides the foundational server infrastructure, enhanced with middleware components for cross-origin resource sharing, request parsing, authentication, and error handling.

The API endpoint design encompasses all supply chain operations, from initial raw material registration through final consumer purchases. Each endpoint implements comprehensive input validation, business logic processing, and appropriate response formatting. File upload handling capabilities support image and document attachments at each supply chain stage, with automatic processing and storage orchestration. The API design includes robust error handling mechanisms, ensuring graceful degradation and informative error responses across all operational scenarios.

Integration services within the backend facilitate seamless communication with external systems including blockchain networks and IPFS storage providers. Asynchronous processing

capabilities enable efficient handling of blockchain transactions and IPFS uploads without blocking API response times. The service architecture includes comprehensive logging and monitoring capabilities, providing operational visibility and debugging support for production deployments.

4.6 Decentralized Storage Integration

The integration of InterPlanetary File System technology through Pinata service provision addresses the fundamental challenge of storing large binary data in blockchain applications. IPFS provides content-addressed storage capabilities, ensuring data integrity through cryptographic hashing while enabling distributed access and redundancy. The Pinata service integration offers managed IPFS infrastructure with guaranteed pinning services, ensuring persistent availability of uploaded content.

The IPFS integration strategy encompasses both image storage and metadata management functions. Product images captured at each supply chain stage are automatically uploaded to IPFS, with resulting content hashes stored in both the local database and blockchain smart contracts. Metadata files containing comprehensive product information, supply chain details, and NFT attributes are similarly uploaded to IPFS, creating immutable records of product provenance and characteristics.

The implementation includes fallback mechanisms to ensure system resilience in cases of IPFS service unavailability. Local storage systems provide temporary redundancy while automatic retry mechanisms attempt to restore IPFS connectivity. Content delivery optimization utilizes IPFS gateway services to minimize access latency while maintaining decentralization benefits. The integration also implements content verification procedures to ensure uploaded data integrity and prevent unauthorized modifications.

4.7 Frontend Development and User Interface Design

The frontend application employs React.js framework to deliver responsive, interactive user interfaces tailored to different stakeholder requirements within the supply chain ecosystem. The component-based architecture enables modular development and maintenance while ensuring consistent user experience patterns across different functional areas. Material-UI design system integration provides standardized visual components and responsive layout capabilities, ensuring optimal user experience across desktop and mobile device platforms.

The application structure implements role-based interface segregation, with distinct modules serving suppliers, manufacturers, certifiers, assemblers, distributors, retailers, and consumers. Each stakeholder interface provides specialized functionality relevant to their supply chain

responsibilities while maintaining access to shared features such as product tracking and authentication verification. The design emphasizes usability and accessibility, incorporating intuitive navigation patterns and clear visual hierarchies to minimize learning curves for non-technical users.

Advanced frontend capabilities include integrated QR code scanning functionality for rapid product identification and verification. Camera integration enables direct image capture for product documentation, while file upload interfaces support document and certification attachments. Real-time data updates ensure stakeholders receive immediate notifications of supply chain events relevant to their operations. The interface design includes comprehensive error handling and user feedback mechanisms, providing clear guidance for successful system interaction.

4.8 Integration Testing and Quality Assurance

The testing methodology encompasses multiple levels of verification to ensure system reliability, security, and performance under various operational conditions. Unit testing procedures validate individual component functionality, including database operations, API endpoints, smart contract functions, and frontend components. Integration testing focuses on inter-component communication and data flow verification, ensuring seamless operation across system boundaries and external service integrations.

End-to-end testing simulates complete supply chain workflows, validating the entire process from raw material registration through consumer purchase completion. These tests verify blockchain transaction processing, IPFS storage operations, database consistency, and user interface responsiveness under realistic usage scenarios. Performance testing evaluates system behavior under various load conditions, identifying potential bottlenecks and optimization opportunities.

Security testing procedures encompass vulnerability assessment, penetration testing, and compliance verification. Blockchain security testing validates smart contract logic, access control mechanisms, and transaction integrity. API security testing evaluates authentication, authorization, and input validation effectiveness. Database security testing ensures proper access controls, encryption implementation, and audit trail functionality.

4.9 Deployment Architecture and Configuration Management

The deployment architecture supports both development and production environments through containerization and automated deployment pipelines. The system deployment utilizes modern

DevOps practices, including infrastructure as code, continuous integration, and automated testing procedures. Environment configuration management ensures consistent deployment characteristics across different operational contexts while maintaining security and performance optimization.

Production deployment considerations include database optimization, API server scaling, load balancing, and content delivery network integration. Blockchain network configuration requires proper key management, transaction monitoring, and gas optimization strategies. IPFS integration deployment includes gateway configuration, pinning service setup, and redundancy planning to ensure content availability and performance.

Monitoring and logging infrastructure provides comprehensive operational visibility, including application performance metrics, database query analysis, blockchain transaction tracking, and user activity monitoring. Automated alerting systems notify operations teams of potential issues before they impact system availability or performance. Backup and disaster recovery procedures ensure business continuity and data protection across all system components.

Chapter 5 System Implementation

5.1 Setting Up

5.1.1 Software

Step 1: Add MetaMask Extension to Chrome

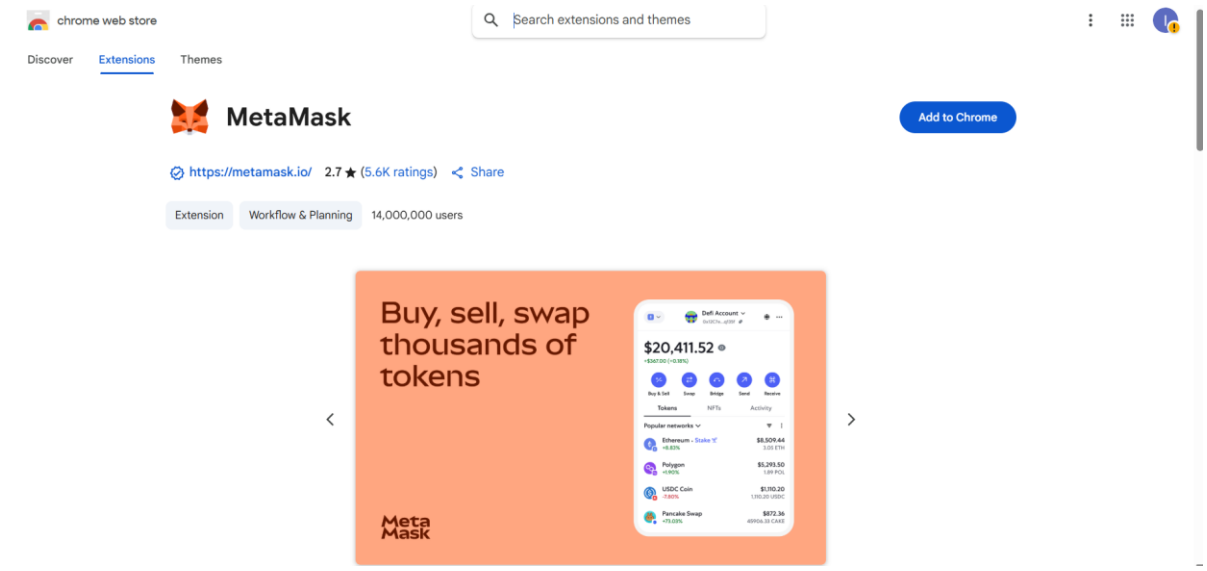

Figure 5.1.1.1 Add MetaMask Extension

Step 2: Create a new wallet on MetaMask

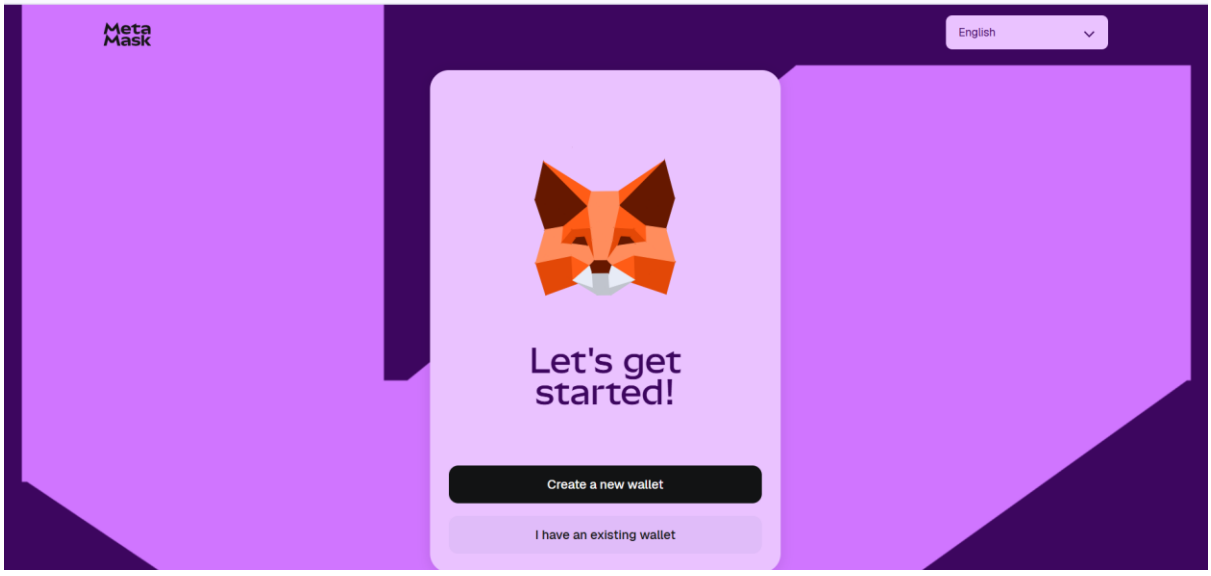

Figure 5.1.1.2 Create New Wallet

Step 3: Once successfully installed, the user can view the MetaMask extension in Chrome

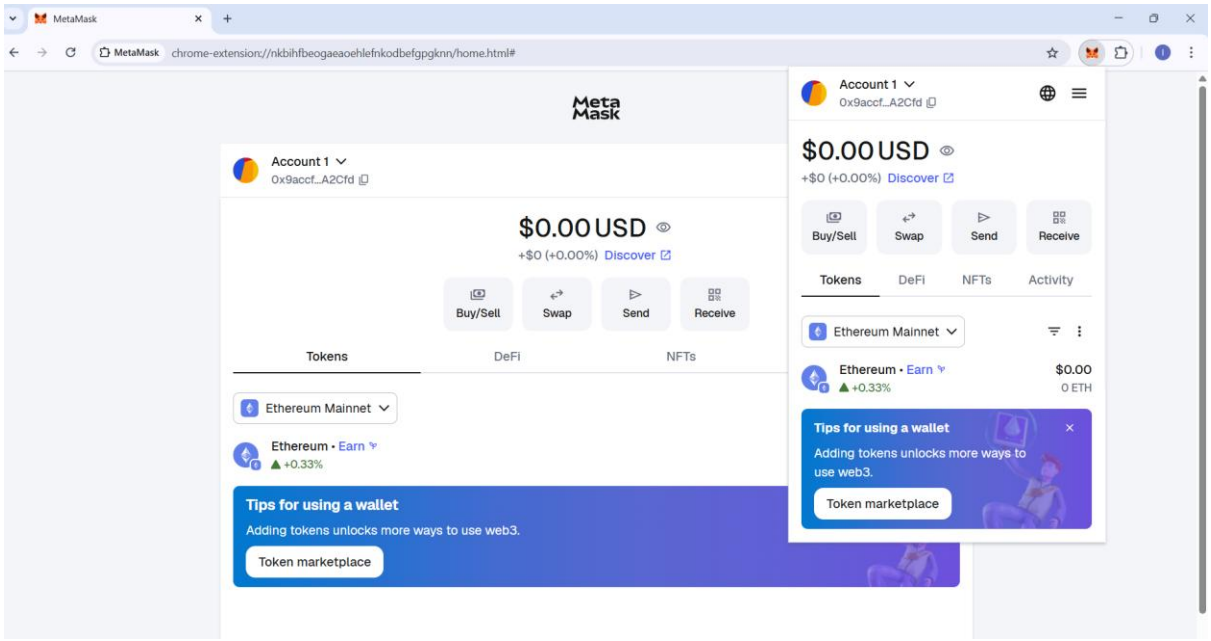

Figure 5.1.1.3 Successfully Output MetaMask Extension

Step 4: Copy the Wallet Address in MetaMask

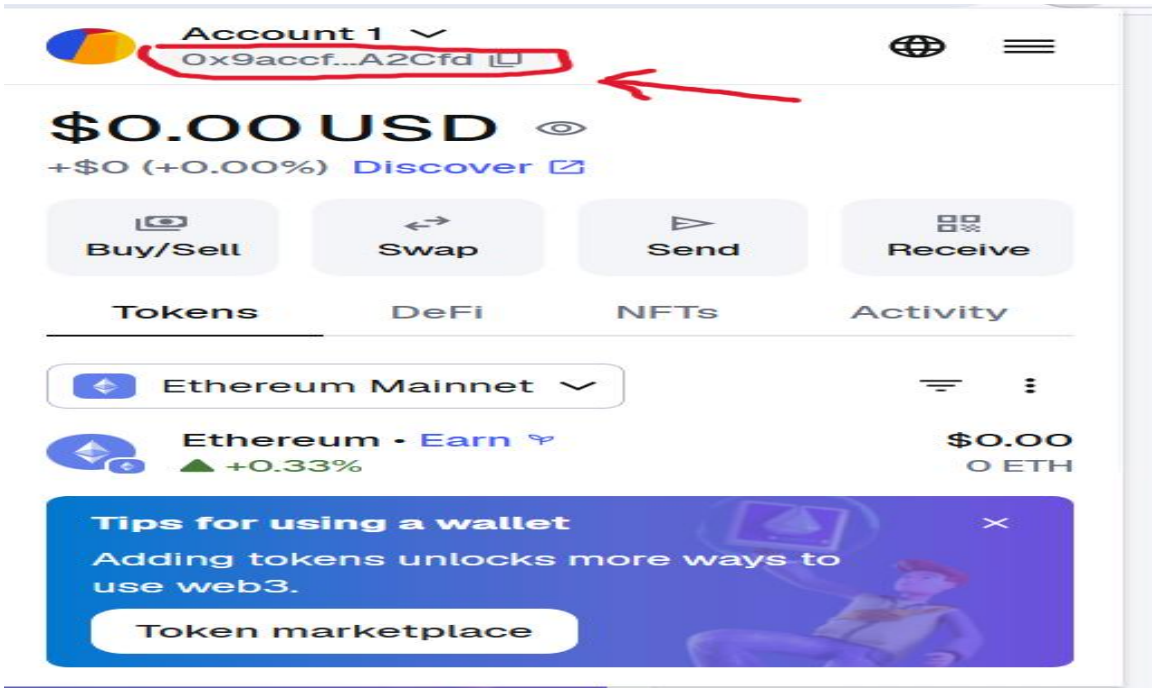

Figure 5.1.1.4 Copy MetaMask Wallet Address

Step 5: Visit https://www.alchemy.com/faucets/ethereum-sepolia to get free Sepolia ETH. This Sepolia ETH is used to make transactions through blockchain technology.

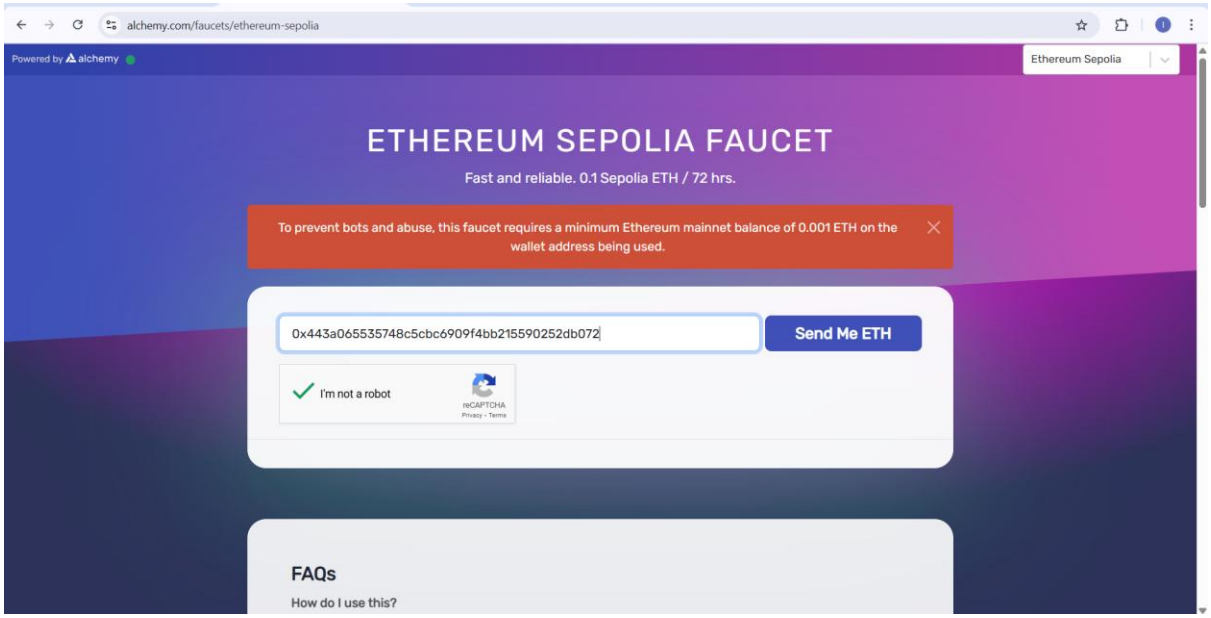

Figure 5.1.1.5 Get Free Sepolia ETH

Step 6: Click “Send Me ETH”, the user is able to receive 0.1 Sepolia ETH

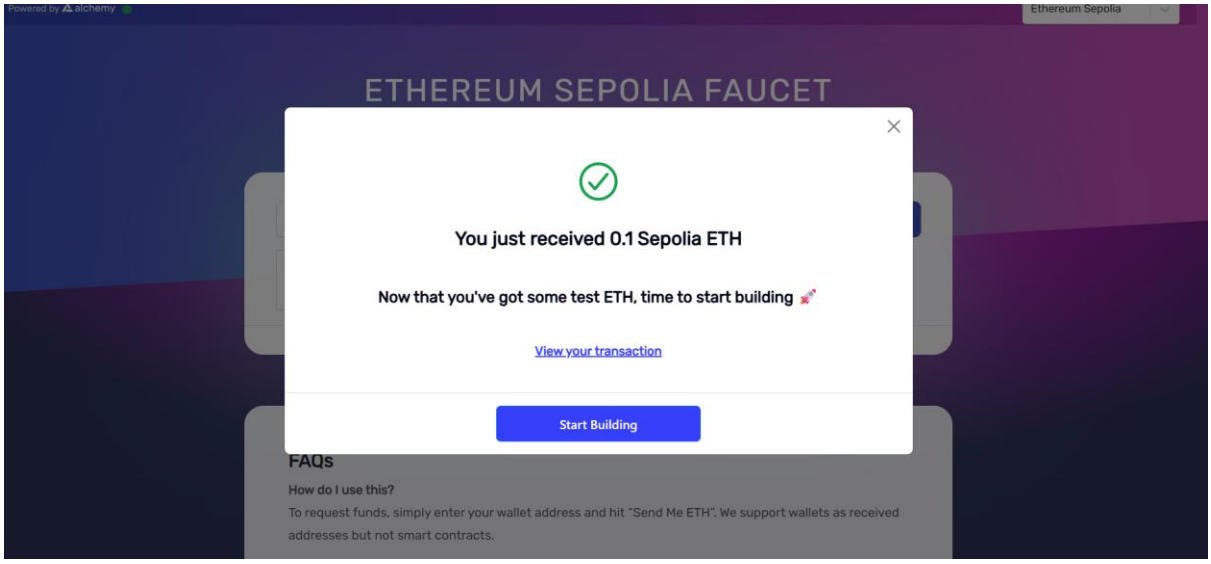

Figure 5.1.1.6 Successfully Get Free Sepolia ETH

Step 7: Visit https://pinata.cloud/ and sign up for a Pinata Account. Pinata is a crypto file storage that is used to store NFT image and metadata

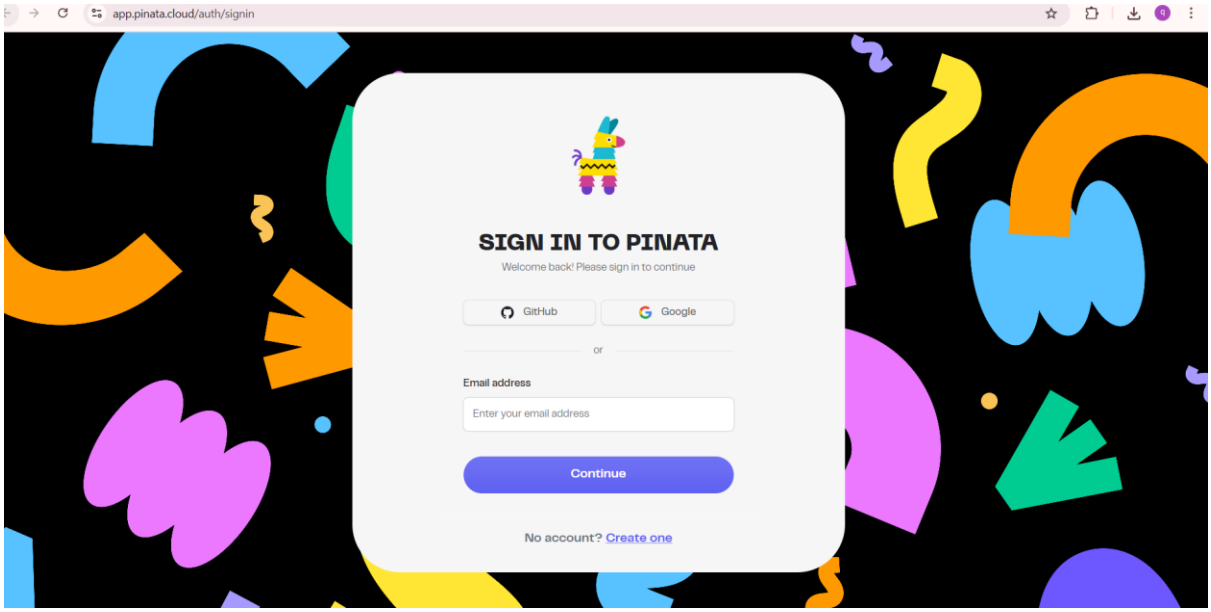

Figure 5.1.1.7 Sign Up Pinata Account

Step 8: Visit https://etherscan.io/ and sign up for an Etherscan Account. Etherscan is a Block Explorer and Analytics Platform for Ethereum, a decentralized smart contracts platform. All the blockchain transactions will be shown on Etherscan

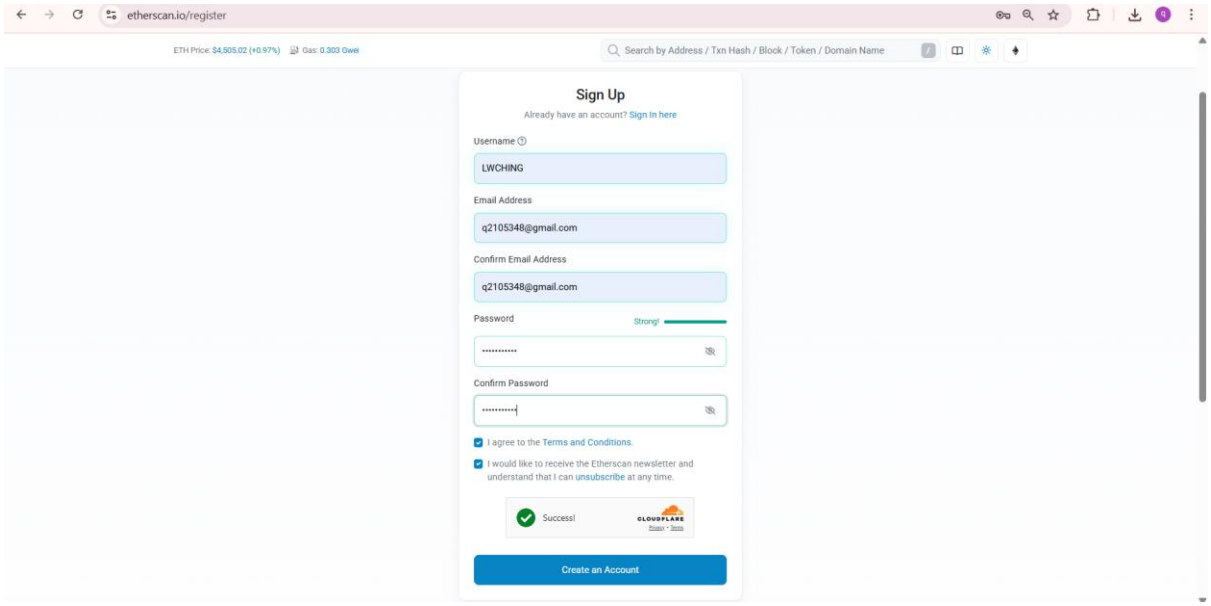

Figure 5.1.1.8 Sign Up Etherscan Account

Step 9: Open Visual Studio Code and install the Solidity extension by Juan Blanco

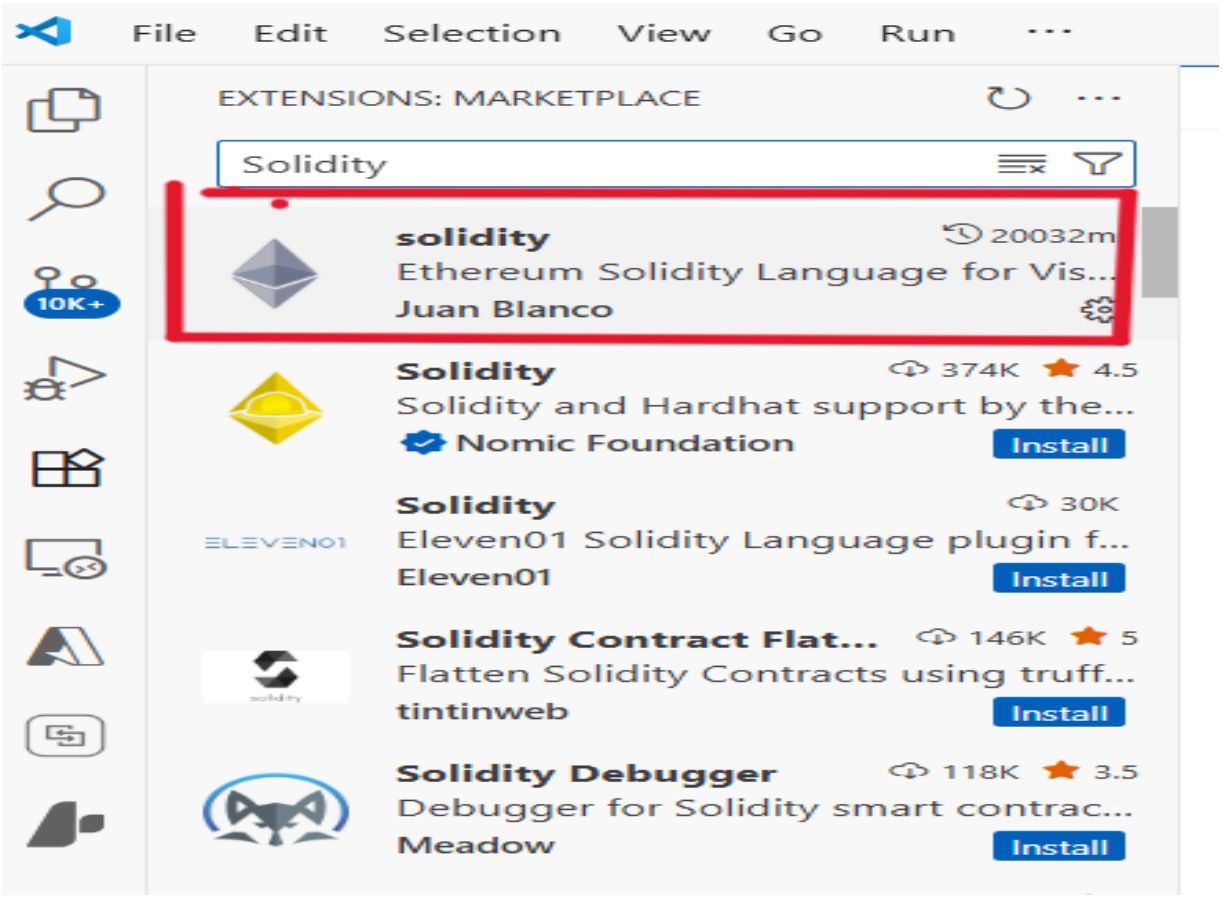

Figure 5.1.1.9 Install Solidity Extension

Step 10: Create LuxuryWatchNFT.sol in the contracts folder

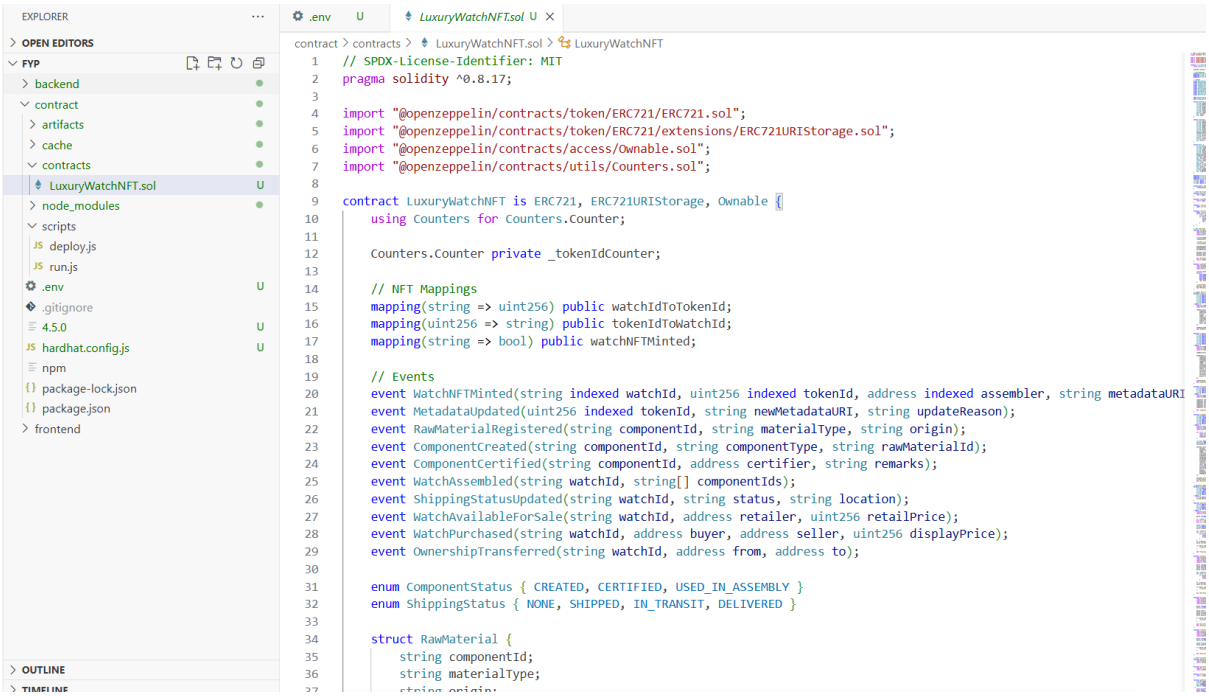

Figure 5.1.1.10 Complete Smart Contract Code

Step 11: Cretae .env to store MetaMask's private key and Ehterscan api key

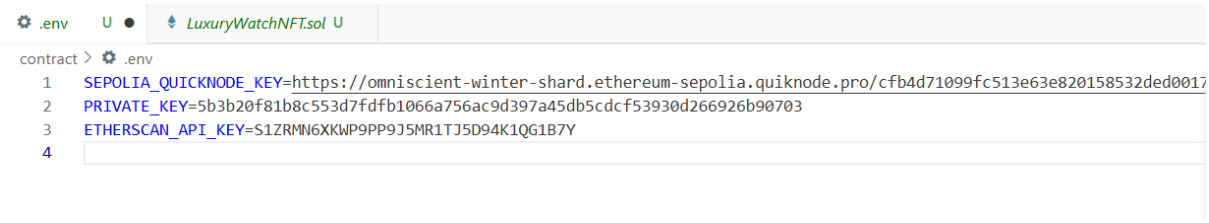

Figure 5.1.1.11 Set up .env in Smart Contract

Step 12: Deploy to Sepolia Testnet using the command "npx hard compile". The successful output has been shown in Figure 5.1.1.12.

```
C:\Users\USER\Desktop\system\FYP dEMO\contract>npx hardhat compile
Compiled 11 Solidity files successfully (evm target: paris).
```

Figure 5.1.1.12 Compile Smart Contract

Step 13: Deploy to Sepolia Testnet using the command “npx hardhat run scripts/deploy.js --network sepolia”. The successful output has been shown in Figure 5.1.1.13.

```
C:\Users\USER\Desktop\Testing\05082025 Latest\FYP\contract>npx hardhat run scripts/deploy.js --network sepolia
Deploying contracts with account: 0x8f6b3d3E26e4A7e79ab6Af0FD63DE43AA7B44A31
Account balance: 314796818395625523
Duplicate definition of OwnershipTransferred (OwnershipTransferred(string,address,address), OwnershipTransferred(address
,address))
LuxuryWatch contract deployed to: 0x3Fa6e449072ed3e357B3018B50Ab965d96d1Eabf
Waiting for 5 confirmations...
Verifying contract on Etherscan...
The contract 0x3Fa6e449072ed3e357B3018B50Ab965d96d1Eabf has already been verified
✅ Contract verified on Etherscan!
```

Figure 5.1.1.13 Deployed Contract Address

Step 14: Check the contract address through Etherscan

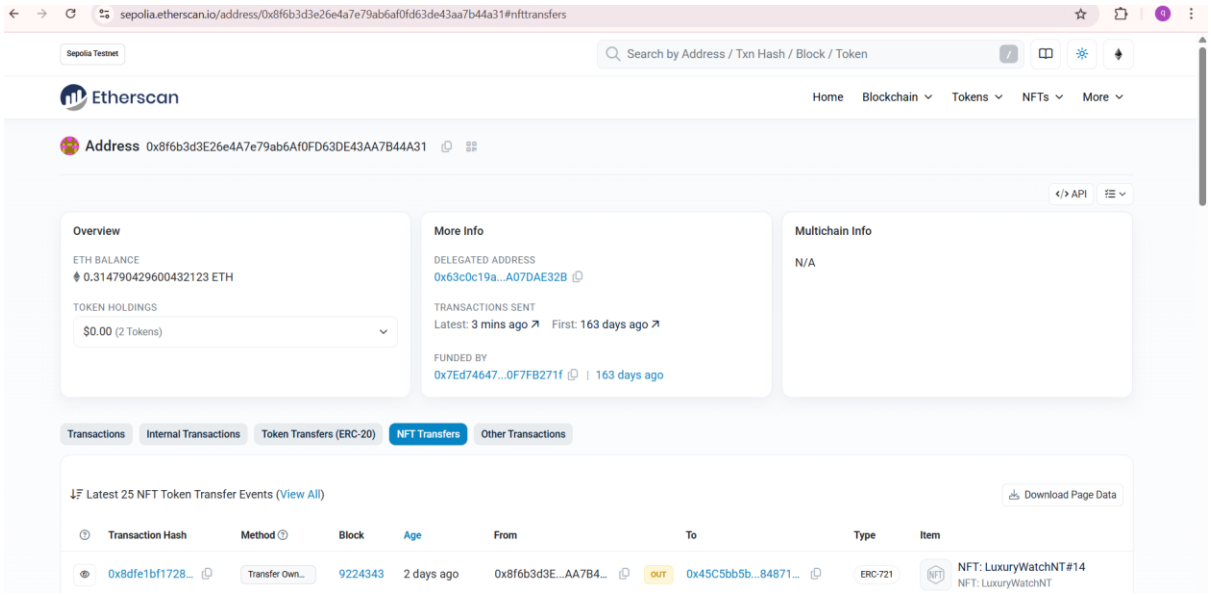

Figure 5.1.1.14 Deployed Contract Address in Etherscan

Step 15: Visit https://railway.com/ and sign up for backend deployment for further operation

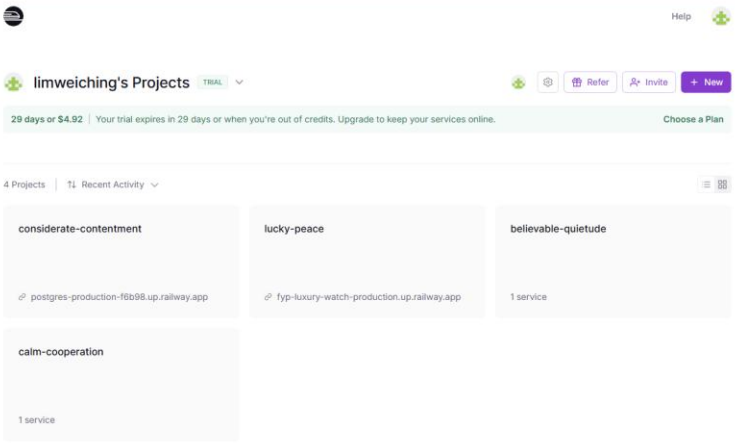

Figure 5.1.1.15 Backend Deployed in Railway

Step 16: Visit https://vercel.com/ and sign up for frontend deployment for further operation

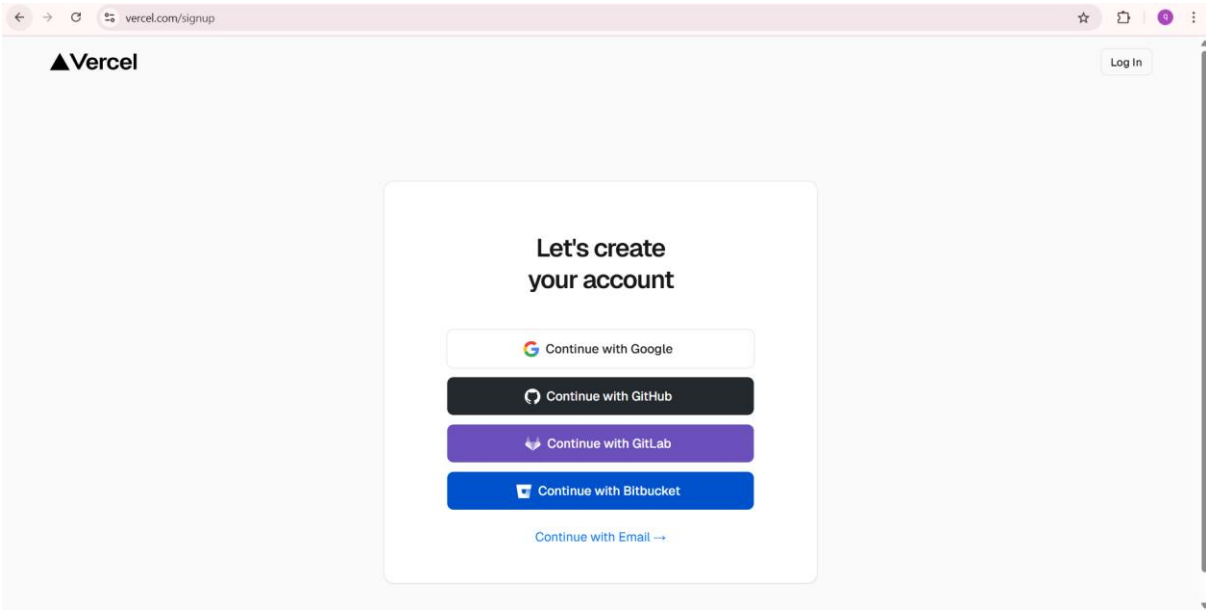

Figure 5.1.1.16 Frontend Deployed in Vercel

5.2 Smart Contract Implementation

1. Inheritance Structure

```
contract LuxuryWatchNFT is ERC721, ERC721URIStorage, Ownable
```

Figure 5.2.1 Inheritance Structure

The ERC-721 provides base NFT functionalities like minting, transferring, and ownership. Besides, the ERC721URIStorage adds metadata URI storage capabilities for NFTs, whereas Ownable implements access control with owner-only functions.

2. Data Structures

```
enum ComponentStatus { CREATED, CERTIFIED, USED_IN_ASSEMBLY }
enum ShippingStatus { NONE, SHIPPED, IN_TRANSIT, DELIVERED }
```

Figure 5.2.2 Data Structures

The ComponentStatus is used to track the component lifecycle stages, whereas the ShippingStatus is used to track the watch shipping progression.

3. Raw Material Structure

```
struct RawMaterial {
    string componentId;      // Unique identifier
    string materialType;     // Type of material (steel, gold, etc.)
    string origin;           // Geographic origin
    string image;            // IPFS image hash
    string location;         // Physical location
    string timestamp;        // Registration time
    address supplier;        // Supplier's wallet address
    bool exists;             // Existence flag
    bool used;               // Usage status
}
```

Figure 5.2.3 Raw Material Structure

The purpose is to represent raw materials, such as precious metals, crystals, or mechanical components, used in watch manufacturing.

4. Component Structure

```
struct Component {
    string componentId;         // Unique identifier
    string componentType;       // Type (movement, case, dial, etc.)
    string serialNumber;        // Manufacturing serial number
    string rawMaterialId;       // Reference to source material
    string image;               // Component image
    string location;            // Manufacturing location
    string timestamp;           // Creation timestamp
    address manufacturer;       // Manufacturer's address
    ComponentStatus status;     // Current status
    string certifierRemarks;    // Quality certification notes
    string certifierImage;      // Certification evidence image
    address certifier;          // Certifier's address
    string certifyTimestamp;    // Certification time
    bool exists;                // Existence flag
}
```

Figure 5.2.4 Component Structure

The purpose is to represent manufactured watch components with full traceability to raw materials and certification status.

5. Watch Structure

```
struct Watch {
    string watchId;              // Unique watch identifier
    string[] componentIds;       // Array of component IDs used
    string image;                // Final watch image
    string location;             // Assembly location
    string timestamp;            // Assembly timestamp
    address assembler;           // Assembler's address
    ShippingStatus shippingStatus; // Current shipping status
    string[] shippingTrail;      // Shipping history trail
    bool availableForSale;       // Sale availability flag
    address retailer;            // Retailer's address
    uint256 retailPrice;         // Retail price in wei
    bool sold;                   // Sale status
    address currentOwner;        // Current owner address
    address[] ownershipHistory;  // Complete ownership chain
    bool exists;                 // Existence flag
    uint256 tokenId;             // Associated NFT token ID
    string metadataURI;          // IPFS metadata URI
    bool isNFTMinted;            // NFT minting status
}
```

Figure 5.2.5 Watch Structure

The purpose is to represent the complete assembled watch, including its full supply chain history and NFT integration.

6. Raw Material Registration

```
function registerRawMaterial(
    string memory _componentId,
    string memory _materialType,
    string memory _origin,
    string memory _image,
    string memory _location,
    string memory _timestamp
) public
```

Figure 5.2.6 Raw Material Registration

The registerRawMaterial function allows suppliers to create blockchain records for premium materials used in watch manufacturing. The function validates unique component IDs, records material type and origin data, and associates the supplier's wallet address with the material. Each registration emits a tracking event and marks the material as available for component manufacturing.

7. Component Creation

```
function createComponent(
    string memory _componentId,
    string memory _componentType,
    string memory _serialNumber,
    string memory _rawMaterialId,
    string memory _image,
    string memory _location,
    string memory _timestamp
) public
```

Figure 5.2.7 Component Creation

The createComponent function enables manufacturers to transform registered raw materials into watch components while maintaining complete traceability. The function links each component to its source material, automatically marks the raw material as "used" to prevent reuse, and sets the component status to "CREATED" for subsequent certification processing.

8. Component Certification

```
function certifyComponent(
    string memory _componentId,
    string memory _remarks,
    string memory _certifierImage,
    string memory _timestamp
) public
```

Figure 5.2.8 Component Certification

The certifyComponent function requires third-party validators to assess component quality before assembly use. Certifiers provide detailed remarks and evidence images that become permanently recorded on the blockchain. Only components with "CERTIFIED" status can proceed to watch assembly, ensuring quality standards throughout the supply chain.

9. Watch Assembly with NFT Minting

```
function assembleWatch(
    string memory _watchId,
    string[] memory _componentIds,
    string memory _image,
    string memory _location,
    string memory _timestamp,
    string memory _metadataURI
) public
```

Figure 5.2.9 Watch Assembly with NFT Minting

The assembleWatch function combines certified components into complete luxury watches while automatically generating unique ERC-721 NFTs. The function requires minimum three certified components, validates all component statuses, and marks used components as "USED_IN_ASSEMBLY." Each successful assembly creates both a comprehensive watch record and a corresponding NFT with IPFS metadata.

10. Shipping Management

```
function updateShippingStatus(
    string memory _watchId,
    uint _status,
    string memory _location,
    string memory _timestamp
) public
```

Figure 5.2.10 Shipping Management

The updateShippingStatus function tracks watch movement through the distribution network with progressive status validation from shipped to in-transit to delivered. Each update creates detailed shipping trail entries and automatically transfers NFT ownership to the current handler, maintaining synchronization between physical custody and digital ownership.

11. Retail Operations

```
function markAvailableForSale(string memory _watchId, uint256 _retailPrice) public {
    require(watches[_watchId].exists, "Watch not found");
    require(watches[_watchId].shippingStatus == ShippingStatus.DELIVERED, "Not delivered");
    require(!watches[_watchId].availableForSale, "Already available");
    require(_retailPrice > 0, "Invalid price");
```

Figure 5.2.11 Retail Operations

The markAvailableForSale function enables retailers to list delivered watches for consumer purchase by setting retail prices and availability status. The function validates proper delivery status before enabling marketplace integration and automatically transfers NFT ownership to the retailer for legitimate sale authorization.

12. Consumer Purchase

```
function purchaseWatch(string memory _watchId) public {
    require(watches[_watchId].exists, "Watch not found");
    require(watches[_watchId].availableForSale, "Not available");
    require(!watches[_watchId].sold, "Already sold");
    require(watches[_watchId].currentOwner != msg.sender, "Already own");
```

Figure 5.2.12 Consumer Purchase

The purchaseWatch function allows consumers to acquire available watches through secure transactions that update ownership records and transfer NFTs. The function validates availability status, prevents duplicate ownership, and creates permanent ownership history

entries that document the complete chain of custody from assembly to final consumer acquisition.

13. NFT Minting (Internal)

```
// NFT Functions
function mintWatchNFT(string memory _watchId, address _assembler, string memory _metadataURI) internal returns (uint256) {
    require(watches[_watchId].exists, "Watch does not exist");
    require(!watchNFTMinted[_watchId], "NFT already minted");
    require(bytes(_metadataURI).length > 0, "Empty metadata URI");
```

Figure 5.2.13 NFT Minting (Internal)

The mintWatchNFT function automatically creates unique ERC-721 tokens when watches are successfully assembled. The function generates sequential token IDs, establishes bidirectional mappings between watch IDs and token IDs, and sets IPFS metadata URIs for each NFT. Each minting operation emits tracking events and marks the watch as having an associated NFT for ownership synchronization.

14. Metadata Updates

```
function updateNFTMetadata(string memory _watchId, string memory _newMetadataURI, string memory _updateReason) external {
    require(watches[_watchId].exists, "Watch does not exist");
    require(watchNFTMinted[_watchId], "NFT not minted");
    require(bytes(_newMetadataURI).length > 0, "Empty metadata URI");
```

Figure 5.2.14 Metadata Updates

The updateNFTMetadata function allows authorized parties to update NFT metadata as watches progress through the supply chain. Only the contract owner, assembler, current owner, or retailer can modify metadata to reflect shipping status, ownership changes, or additional certifications. Each update requires a valid IPFS URI and update reason that becomes permanently recorded on the blockchain.

15. Token Transfer Integration

```
function _beforeTokenTransfer(address from, address to, uint256 tokenId, uint256 batchSize) internal override {
    super._beforeTokenTransfer(from, to, tokenId, batchSize);
    if (from != address(0) && to != address(0)) {
        string memory watchId = tokenIdToWatchId[tokenId];
        if (bytes(watchId).length > 0) {
            watches[watchId].currentOwner = to;
            watches[watchId].ownershipHistory.push(to);
            emit OwnershipTransferred(watchId, from, to);
        }
    }
}
```

Figure 5.2.15 Token Transfer Integration

The _beforeTokenTransfer function automatically synchronizes NFT ownership with watch ownership records during any token transfer. The function updates the watch's current owner address and adds the new owner to the ownership history array. This ensures that physical watch ownership always mirrors digital NFT ownership regardless of how the token is transferred.

16. Individual Record Retrieval

```
function getRawMaterial(string memory _componentId) public view returns (RawMaterial memory) {
    require(rawMaterials[_componentId].exists, "Not found");
    return rawMaterials[_componentId];
}

function getComponent(string memory _componentId) public view returns (Component memory) {
    require(components[_componentId].exists, "Not found");
    return components[_componentId];
}

function getWatch(string memory _watchId) public view returns (Watch memory) {
    require(watches[_watchId].exists, "Not found");
    return watches[_watchId];
}
```

Figure 5.2.16 Individual Record Retrieval

It used to retrieve complete records for supply chain verification

17. NFT Information

```
function getWatchNFTInfo(string memory _watchId) public view returns (
    uint256 tokenId,
    string memory metadataURI,
    bool isNFTMinted,
    address nftOwner
)
```

Figure 5.2.17 NFT Information

The contract provides specialized view functions for NFT-related queries including getWatchNFTInfo for complete NFT details, getWatchTokenId for token ID lookup, and getTokenWatchId for reverse lookup functionality. The totalNFTsMinted function returns the current count of minted NFTs, enabling marketplace integration and collection analytics.

5.3 Database Design and Implementation

Figure 5.3.1 Logo PostgreSQL

The system utilizes PostgreSQL as the primary relational database management system, chosen for its robust ACID compliance, advanced indexing capabilities, and superior performance characteristics for complex supply chain operations. PostgreSQL's native support for array data types proves particularly valuable for storing ownership histories and shipping trails without requiring separate junction tables.

Core Database Schema

1. Users Table

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(50) NOT NULL,
    name VARCHAR(255),
    email VARCHAR(255),
    phone VARCHAR(20),
    description TEXT,
    website VARCHAR(255),
    location VARCHAR(255),
    image VARCHAR(255),
    wallet_address VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Figure 5.3.2 Database Scheme for Users Table

The users table implements a comprehensive stakeholder management system supporting multiple roles including suppliers, manufacturers, certifiers, assemblers, distributors, retailers, and consumers. The schema incorporates both authentication credentials and detailed profile information, enabling role-based access control throughout the supply chain.

2. Raw Materials Table

```
CREATE TABLE raw_materials (
    component_id VARCHAR(255) PRIMARY KEY,
    material_type VARCHAR(255) NOT NULL,
    origin VARCHAR(255) NOT NULL,
    image VARCHAR(255),
    location VARCHAR(255),
    timestamp VARCHAR(255),
    supplier_address VARCHAR(255) NOT NULL,
    used BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Figure 5.3.3 Database Scheme for Raw Material Table

The raw_materials table serves as the foundational supply chain entity, capturing comprehensive material provenance including geographical origin, material specifications, and supplier attribution. The boolean 'used' flag prevents double-spending of materials across multiple components.

3. Components Table

```
CREATE TABLE components (
    component_id VARCHAR(255) PRIMARY KEY,
    component_type VARCHAR(255) NOT NULL,
    serial_number VARCHAR(255),
    raw_material_id VARCHAR(255) REFERENCES raw_materials(component_id),
    image VARCHAR(255),
    location VARCHAR(255),
    timestamp VARCHAR(255),
    manufacturer_address VARCHAR(255) NOT NULL,
    status INTEGER DEFAULT 0, -- 0: Created, 1: Certified, 2: Used
    certifier_remarks TEXT,
    certifier_image VARCHAR(255),
    certifier_address VARCHAR(255),
    certify_timestamp VARCHAR(255),
    certifier_location VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Figure 5.3.4 Database Scheme for Components Table

The components table implements a sophisticated manufacturing tracking system with status enumeration supporting the progression from creation through certification to assembly utilization. Foreign key relationships maintain referential integrity to source raw materials while status fields enable precise supply chain state management.

4. Watches Table

```
CREATE TABLE watches (
    watch_id VARCHAR(255) PRIMARY KEY,
    component_ids TEXT[] NOT NULL,
    image VARCHAR(255),
    location VARCHAR(255),
    timestamp VARCHAR(255),
    assembler_address VARCHAR(255) NOT NULL,
    shipping_status INTEGER DEFAULT 0, -- 0: Not shipped, 1: Shipped, 2: In
    shipping_trail TEXT[],
    current_owner VARCHAR(255),
    ownership_history TEXT[],
    available_for_sale BOOLEAN DEFAULT FALSE,
    sold BOOLEAN DEFAULT FALSE,
    retail_price DECIMAL(15,2) DEFAULT 0.00,
    retailer_address VARCHAR(255),
    retailer_location VARCHAR(255),
    retailer_timestamp TIMESTAMP WITH TIME ZONE,
    consumer_timestamp TIMESTAMP WITH TIME ZONE,
    last_transfer_timestamp TIMESTAMP,
    distributor_address VARCHAR(255),
    nft_generated BOOLEAN DEFAULT FALSE,
    nft_metadata_uri TEXT,
    nft_image_uri TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Figure 5.3.5 Database Scheme for Watches Table

The watches table represents the culmination of the supply chain process, incorporating extensive metadata including component arrays, shipping status tracking, ownership histories, and NFT integration fields. The schema includes timestamp columns for detailed temporal tracking of each supply chain phase.

5. AI Model Table

```
CREATE TABLE aimodel (
    watch_id VARCHAR(255) PRIMARY KEY REFERENCES watches(watch_id),
    ai_analyzed BOOLEAN DEFAULT FALSE,
    ai_watch_model VARCHAR(255),
    ai_market_price DECIMAL(15,2) DEFAULT 0.00,
    ai_analysis_date TIMESTAMP,
    ai_confidence_score DECIMAL(3,2) DEFAULT 0.00,
    month1 DECIMAL(15,2), month2 DECIMAL(15,2), month3 DECIMAL(15,2),
    month4 DECIMAL(15,2), month5 DECIMAL(15,2), month6 DECIMAL(15,2),
    month7 DECIMAL(15,2), month8 DECIMAL(15,2), month9 DECIMAL(15,2),
    month10 DECIMAL(15,2), month11 DECIMAL(15,2), month12 DECIMAL(15,2),
    ai_trend_analysis TEXT,
    ai_recommendations TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Figure 5.3.6 Database Scheme for AI Model Table

The aimodel table provides specialized storage for artificial intelligence analysis results, including monthly price tracking arrays, trend analysis, and confidence scoring. This dedicated table enables sophisticated market analysis and watch valuation services.

6. Ownership Transfers Table

```
CREATE TABLE ownership_transfers (
    id SERIAL PRIMARY KEY,
    watch_id VARCHAR(255) REFERENCES watches(watch_id),
    from_address VARCHAR(255),
    to_address VARCHAR(255),
    transfer_type VARCHAR(100), -- 'initial_purchase', 'consumer_transfer'
    transfer_timestamp TIMESTAMP WITH TIME ZONE,
    transfer_location VARCHAR(255),
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Figure 5.3.7 Database Scheme for Ownership Transfers Table

The ownership_transfers table maintains a comprehensive audit trail of all watch ownership changes, recording detailed transaction information including source and destination addresses, transfer types, timestamps, and locations to ensure complete traceability and accountability throughout the secondary market lifecycle.

5.4 Backend API Endpoints Implementation

Authentication Module

1. POST /register

It is used to register new users with role-based access control for different supply chain stakeholders.

```
app.post("/register", async (req, res) => {
  const { username, password, role, fullName, email, phone, walletAddress } =

  try {
    const checkUser = await client.query("SELECT * FROM users WHERE username
    if (checkUser.rows.length > 0) {
      return res.status(400).send("Username already exists");
    }

    await client.query(
      "INSERT INTO users (username, password, role, name, email, phone, walle
      [username, password, role, fullName, email, phone, walletAddress]
    );

    res.status(201).send("User registered successfully");
  } catch (error) {
    res.status(500).send("Error registering user");
  }
});
```

Figure 5.4.1 Backend API Endpoints - POST /register

2. POST /addaccount

It is used to register new users with role-based access control for different supply chain stakeholders.

```
app.post("/addaccount", async (req, res) => {
  const { username, password, role, name, email, phone, description, website,

  if (!username || !password || !role) {
    return res.status(400).json({
      success: false,
      message: "Username, password, and role are required",
    });
  }

  try {
    const checkUser = await client.query("SELECT * FROM users WHERE username
    if (checkUser.rows.length > 0) {
      return res.status(400).json({ success: false, message: "Username alreac
    }

    const result = await client.query(
      `INSERT INTO users (username, password, role, name, email, phone, descr
       VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, CURRENT_TIMESTAMP, CU
      [username, password, role, name || "", email || "", phone || "", descri
    );

    res.json({ success: true, message: "Account created successfully", userId
  } catch (error) {
    res.status(500).json({ success: false, message: "Error creating account",
  }
});
```

Figure 5.4.2 Backend API Endpoints - POST /addaccount

3. POST /login

Authenticates users and returns user data including wallet address for blockchain interactions.

```
app.post("/login", async (req, res) => {
  const { username, password } = req.body;

  try {
    const result = await client.query("SELECT * FROM users WHERE usernam
    if (result.rows.length === 0) {
      return res.status(401).json({ message: "Invalid username or passw
    }

    const user = result.rows[0];
    if (password !== user.password) {
      return res.status(401).json({ message: "Invalid username or passw
    }

    const userData = {
      id: user.id,
      username: user.username,
      role: user.role,
      name: user.name,
      email: user.email,
      phone: user.phone,
      wallet_address: user.wallet_address,
      image: user.image,
    };

    res.json(userData);
  } catch (error) {
    res.status(500).json({ message: "Server error during login", error:
  }
```

Figure 5.4.3 Backend API Endpoints - POST /login

4. GET /users

Retrieves all registered users with their profile information.

```
app.get("/users", async (req, res) => {
  try {
    const data = await client.query(
      "SELECT id, username, role, name, description, website, location, image
    );
    res.send(data.rows);
  } catch (err) {
    console.log("Error fetching users:", err.message);
    res.status(500).send("Error fetching users");
  }
});
```

Figure 5.4.4 Backend API Endpoints - Get /users

5. GET /profile/:identifier

Retrieves user profile by ID or username.

```
app.get("/profile/:identifier", async (req, res) => {
  const { identifier } = req.params;

  try {
    let result;
    if (!isNaN(identifier)) {
      result = await client.query(
        "SELECT id, username, role, name, description, website, location, im
        [identifier]
      );
    } else {
      result = await client.query(
        "SELECT id, username, role, name, description, website, location, im
        [identifier]
      );
    }

    if (result.rows.length === 0) {
      return res.status(404).json({ success: false, message: "Profile not fo
    }

    res.json({ success: true, data: result.rows[0] });
  } catch (error) {
    res.status(500).json({ success: false, message: "Error fetching profile"
  }
```

Figure 5.4.5 Backend API Endpoints - Get / profile/:identifier

6. GET /profileAll

Retrieves all user profiles for management interfaces.

```
app.get("/profileAll", async (req, res) => {
  try {
    const result = await client.query(
      `SELECT id, username, role, name, description, website, location, image
       FROM users ORDER BY created_at DESC`
    );
    res.json(result.rows);
  } catch (error) {
    res.status(500).json({ success: false, message: "Error fetching profiles"
  }
});
```

Figure 5.4.6 Backend API Endpoints - Get / profileAll

7. PUT /user/:id

Updates user profile information.

```
app.put("/user/:id", async (req, res) => {
  const { id } = req.params;
  const { name, description, website, location, image, email, phone, walletAd

  try {
    await client.query(
      "UPDATE users SET name = $1, description = $2, website = $3, location =
      [name, description, website, location, image, email, phone, walletAddre
    );
    res.send("User updated successfully");
  } catch (error) {
    res.status(500).send("Error updating user");
  }
});
```

Figure 5.4.7 Backend API Endpoints - PUT /user/:id

8. DELETE /user/:id

Deletes user account.

```
app.delete("/user/:id", async (req, res) => {
  const { id } = req.params;
  try {
    const result = await client.query("DELETE FROM users WHERE id = $1", [id]
    if (result.rowCount === 0) {
      res.status(404).send("User not found");
    } else {
      res.send("User deleted successfully");
    }
  } catch (error) {
    res.status(500).send("Error deleting user");
  }
});
```

Figure 5.4.8 Backend API Endpoints - DELETE /user/:id

Supplier Module

1. POST /raw-material

Registers premium raw materials with origin verification and supplier attribution.

```
app.post("/raw-material", (req, res) => {
  const { componentId, materialType, origin, image, location, timestamp, supp

  client.query(
    "INSERT INTO raw_materials (component_id, material_type, origin, image,
    [componentId, materialType, origin, image, location, timestamp, supplier
    (err, result) => {
      if (err) {
        console.log("Error adding raw material:", err.message);
        res.status(500).send("Error adding raw material");
      } else {
        console.log("Raw material added successfully");
        res.send("Raw material added");
      }
    }
  );
});
```

Figure 5.4.9 Backend API Endpoints - POST /raw-material

2. GET /raw-materials

Retrieves all registered raw materials with usage status.

```
app.get("/raw-materials", async (req, res) => {
  try {
    const data = await client.query("SELECT * FROM raw_materials ORDER BY cre
    res.send(data.rows);
  } catch (err) {
    console.log("Error fetching raw materials:", err.message);
    res.status(500).send("Error fetching raw materials");
  }
});
```

Figure 5.4.10 Backend API Endpoints - Get /raw-material

3. GET /raw-material/:componentId

Retrieves specific raw material information by component ID.

```
app.get("/raw-material/:componentId", async (req, res) => {
  const { componentId } = req.params;

  try {
    const data = await client.query("SELECT * FROM raw_materials WHERE compo

    if (data.rows.length === 0) {
      return res.status(404).json({ success: false, message: "Raw material n
    }

    res.json(data.rows);
  } catch (err) {
    res.status(500).json({ success: false, message: "Error fetching raw mate
  }
});
```

Figure 5.4.11 Backend API Endpoints - Get /raw-material/:componentId

4. GET /raw-materials/supplier/:supplierAddress

Retrieves raw materials by specific supplier address.

```
app.get("/raw-materials/supplier/:supplierAddress", async (req, res) => {
  const { supplierAddress } = req.params;

  try {
    const data = await client.query(
      "SELECT * FROM raw_materials WHERE supplier_address = $1 ORDER BY creat
      [supplierAddress]
    );
    res.json(data.rows);
  } catch (err) {
    res.status(500).json({ success: false, message: "Error fetching raw mater
  }
});
```

Figure 5.4.12 Backend API Endpoints - Get / raw-materials/supplier/:supplierAddress

Manufacturer Module

1. POST /component

Creates components from registered raw materials with automatic usage tracking.

```
app.post("/component", async (req, res) => {
  const { componentId, componentType, serialNumber, rawMaterialId, image, loc

  try {
    const checkExisting = await client.query("SELECT component_id FROM compor
    if (checkExisting.rows.length > 0) {
      return res.status(400).send("Component ID already exists");
    }

    const checkRawMaterial = await client.query("SELECT component_id, used FR
    if (checkRawMaterial.rows.length === 0) {
      return res.status(400).send("Raw material not found");
    }
    if (checkRawMaterial.rows[0].used) {
      return res.status(400).send("Raw material has already been used");
    }

    await client.query(
      "INSERT INTO components (component_id, component_type, serial_number, r
      [componentId, componentType, serialNumber, rawMaterialId, image, locati
    );

    await client.query("UPDATE raw_materials SET used = TRUE WHERE component_
    res.send("Component added");
  } catch (err) {
    res.status(500).send(`Error adding component: ${err.message}`);
  }
});
```

Figure 5.4.13 Backend API Endpoints - POST /component

2. GET /components

Retrieves all manufactured components with certification status.

```
app.get("/components", async (req, res) => {
  try {
    const data = await client.query("SELECT * FROM components ORDER BY create
    res.send(data.rows);
  } catch (err) {
    res.status(500).send("Error fetching components");
  }
});
```

Figure 5.4.14 Backend API Endpoints - Get /components

3. GET /component/:componentId

Retrieves specific component information with raw material details.

```
app.get("/component/:componentId", async (req, res) => {
  const { componentId } = req.params;

  try {
    const data = await client.query(
      `SELECT c.*, rm.material_type, rm.origin
       FROM components c
       LEFT JOIN raw_materials rm ON c.raw_material_id = rm.component_id
       WHERE c.component_id = $1`,
      [componentId]
    );

    if (data.rows.length === 0) {
      return res.status(404).json({ success: false, message: "Component not f
    }

    res.json(data.rows);
  } catch (err) {
    res.status(500).json({ success: false, message: "Error fetching component
  }
});
```

Figure 5.4.15 Backend API Endpoints - Get/ component/:componentId

4. GET /components/manufacturer/:manufacturerAddress

Retrieves components by specific manufacturer address.

```
app.get("/components/manufacturer/:manufacturerAddress", async (req, res) =>
  const { manufacturerAddress } = req.params;

  try {
    const data = await client.query(
      "SELECT * FROM components WHERE manufacturer_address = $1 ORDER BY crea
      [manufacturerAddress]
    );
    res.json(data.rows);
  } catch (err) {
    res.status(500).json({ success: false, message: "Error fetching component
  }
});
```

Figure 5.4.16 Backend API Endpoints - GET /components/manufacturer/:manufacturerAddress

Certifier Module

1. POST /certify-component

Validates component quality and updates certification status with evidence.

```
app.post("/certify-component", (req, res) => {
  const { componentId, remarks, certifierImage, location, timestamp, certifie

  client.query(
    "UPDATE components SET status = '1', certifier_remarks = $1, certifier_im
    [remarks, certifierImage, timestamp, certifierAddress, location, componen
    (err, result) => {
      if (err) {
        res.status(500).send("Error certifying component");
      } else if (result.rowCount === 0) {
        res.status(400).send("Component not found or not in correct status fo
      } else {
        res.send("Component certified");
      }
    }
  );
});
```

Figure 5.4.17 Backend API Endpoints - POST/certify-component

2. GET /components/uncertified

Validates component quality and updates certification status with evidence.

```
app.post("/certify-component", (req, res) => {
  const { componentId, remarks, certifierImage, location, timestamp, certifie

  client.query(
    "UPDATE components SET status = '1', certifier_remarks = $1, certifier_im
    [remarks, certifierImage, timestamp, certifierAddress, location, componen
    (err, result) => {
      if (err) {
        res.status(500).send("Error certifying component");
      } else if (result.rowCount === 0) {
        res.status(400).send("Component not found or not in correct status fo
      } else {
        res.send("Component certified");
      }
    }
  );
});
```

Figure 5.4.18 Backend API Endpoints - GET /components/uncertified

3. GET /components/certified

Returns all quality-certified components available for watch assembly.

```
app.get("/components/certified", async (req, res) => {
  try {
    const data = await client.query("SELECT * FROM components WHERE status =
    res.send(data.rows);
  } catch (err) {
    res.status(500).send("Error fetching certified components");
  }
});
```

Figure 5.4.19 Backend API Endpoints - GET /components/certified

Assembler Module

1. POST /watch

Assembles watches from certified components while automatically generating NFT metadata.

```
app.post("/watch", async (req, res) => {
  const { watchId, componentIds, image, location, timestamp, assemblerAddress

  try {
    if (image && image.trim() !== "") {
      const imagePath = path.join(__dirname, "public/uploads/watch", image);
      if (!fs.existsSync(imagePath)) {
        return res.status(400).json({ success: false, message: `Image file nc
      }
    }

    let nftMetadata = null;
    if (generateNFT && image && image.trim() !== "") {
      try {
        nftMetadata = await generateWatchNFTMetadata({
          watchId, componentIds, image, location, timestamp, assemblerAddress
        });
      } catch (nftError) {
        console.error("NFT generation failed:", nftError);
        nftMetadata = null;
      }
    }
```

Figure 5.4.20 Backend API Endpoints - POST/watch

2. GET /watches

Retrieves all assembled watches with their status information.

```
app.get("/watches", async (req, res) => {
  try {
    const data = await client.query("SELECT * FROM watches ORDER BY created_a
    res.send(data.rows);
  } catch (err) {
    res.status(500).send("Error fetching watches");
  }
});
```

Figure 5.4.21 Backend API Endpoints - GET /watches

3. GET /watch/:watchId

Retrieves specific watch information with watch id.

```
app.get("/watch/:watchId", async (req, res) => {
  const { watchId } = req.params;

  try {
    const data = await client.query(
      `SELECT w.*,
       COALESCE(w.shipping_trail, ARRAY[]::text[]) as shipping_trail,
       COALESCE(w.ownership_history, ARRAY[]::text[]) as ownership_history,
       CASE WHEN ai.ai_analyzed IS TRUE THEN true ELSE false END as ai_analyz
       ai.ai_watch_model, ai.ai_market_price, ai.ai_analysis_date, ai.ai_conf
       ai.month1, ai.month2, ai.month3, ai.month4, ai.month5, ai.month6,
       ai.month7, ai.month8, ai.month9, ai.month10, ai.month11, ai.month12,
       ai.ai_trend_analysis, ai.ai_recommendations
       FROM watches w
       LEFT JOIN aimodel ai ON w.watch_id = ai.watch_id
       WHERE w.watch_id = $1`,
      [watchId]
    );

    if (data.rows.length === 0) {
```

Figure 5.4.22 Backend API Endpoints - GET /watch/:watchId

4. GET /watches/assembler/:assemblerAddress

Retrieves watches by specific assembler address.

```
app.get("/watches/assembler/:assemblerAddress", async (req, res) => {
  const { assemblerAddress } = req.params;

  try {
    const data = await client.query(
      "SELECT * FROM watches WHERE assembler_address = $1 ORDER BY created_at
      [assemblerAddress]
    );
    res.json(data.rows);
  } catch (err) {
    res.status(500).json({ success: false, message: "Error fetching watches",
  }
});
```

Figure 5.4.23 Backend API Endpoints - GET / watches/assembler/:assemblerAddress

5. GET /watches/available

Retrieves watches available for purchase.

```
app.get("/watches/available", async (req, res) => {
  try {
    const data = await client.query(
      `SELECT w.*,
       CASE WHEN ai.ai_analyzed IS TRUE THEN true ELSE false END as ai_analyz
       ai.ai_watch_model, ai.ai_market_price, ai.ai_confidence_score
       FROM watches w
       LEFT JOIN aimodel ai ON w.watch_id = ai.watch_id
       WHERE w.shipping_status = 3 AND w.available_for_sale = TRUE AND w.sold
       ORDER BY w.created_at DESC`
    );
    res.json(data.rows);
  } catch (err) {
    res.status(500).json({ success: false, message: "Error fetching available
  }
});
```

Figure 5.4.24 Backend API Endpoints - GET/watches/available

6. GET /watches/for-sale

Retrieves all watches currently for sale with pricing information.

```
app.get("/watches/for-sale", async (req, res) => {
  try {
    const data = await client.query(
      `SELECT w.*,
       CASE WHEN ai.ai_analyzed IS TRUE THEN true ELSE false END as ai_analyz
       ai.ai_watch_model, ai.ai_market_price, ai.ai_confidence_score
       FROM watches w
       LEFT JOIN aimodel ai ON w.watch_id = ai.watch_id
       WHERE w.available_for_sale = TRUE AND w.sold = FALSE AND w.retail_pric
       ORDER BY w.created_at DESC`
    );
    res.json(data.rows);
  } catch (err) {
    res.status(500).json({ success: false, message: "Error fetching watches f
  }
});
```

Figure 5.4.25 Backend API Endpoints - GET/watches/for-sale

7. POST /generate-nft-metadata

Generates NFT metadata for existing watches.

```
app.post("/generate-nft-metadata", async (req, res) => {
  const { watchId } = req.body;

  if (!watchId) {
    return res.status(400).json({ success: false, message: "Watch ID is requi
  }

  try {
    const watchResult = await client.query("SELECT * FROM watches WHERE watch
    if (watchResult.rows.length === 0) {
      return res.status(404).json({ success: false, message: "Watch not found
    }

    const watch = watchResult.rows[0];
    if (!watch.image) {
      return res.status(400).json({ success: false, message: "Watch must have
    }

    const nftData = await generateWatchNFTMetadata({
      watchId: watch.watch_id,
      componentIds: watch.component_ids,
      image: watch.image,
      location: watch.location,
      timestamp: watch.timestamp,
      assemblerAddress: watch.assembler_address,
    });
```

Figure 5.4.26 Backend API Endpoints - POST /generate-nft-metadata

8. POST /analyze-watch

Initiates AI-powered watch analysis using Gemini API.

```
app.post("/analyze-watch", async (req, res) => {
  const { watchId } = req.body;

  if (!watchId) {
    return res.status(400).json({ success: false, message: "Watch ID is requi
  }

  try {
    const watchResult = await client.query("SELECT watch_id, image FROM watch
    if (watchResult.rows.length === 0) {
      return res.status(404).json({ success: false, message: "Watch not found
    }

    const watch = watchResult.rows[0];
    if (!watch.image) {
      return res.status(400).json({ success: false, message: "Watch must have
    }

    const base64Image = await convertImageToBase64(watch.image);
    const analysisResult = await analyzeWatchWithAI(base64Image);

    await client.query(
      `INSERT INTO aimodel (watch_id, ai_analyzed, ai_watch_model, ai_market_
       ai_confidence_score, month1, month2, month3, month4, month5, month6,
       month7, month8, month9, month10, month11, month12, ai_trend_analysis,
       VALUES ($1, TRUE, $2, $3, CURRENT_TIMESTAMP, $4, $5, $6, $7, $8, $9, $
       ON CONFLICT (watch_id) DO UPDATE SET
         ai_analyzed = TRUE, ai_watch_model = $2, ai_market_price = $3,
         ai_analysis_date = CURRENT_TIMESTAMP, ai_confidence_score = $4,
```

Figure 5.4.27 Backend API Endpoints - POST /analyze-watch

9. GET /watch-analysis/:watchId

Retrieves AI analysis results for a specific watch.

```
app.get("/watch-analysis/:watchId", async (req, res) => {
  const { watchId } = req.params;

  try {
    const result = await client.query(
      `SELECT watch_id, ai_analyzed, ai_watch_model, ai_market_price, ai_anal
       ai_confidence_score, month1, month2, month3, month4, month5, month6,
       month7, month8, month9, month10, month11, month12, ai_trend_analysis,
       FROM aimodel WHERE watch_id = $1`,
      [watchId]
    );

    if (result.rows.length === 0) {
      return res.status(404).json({ success: false, message: "AI analysis not
    }

    const analysis = result.rows[0];
    const monthlyPrices = [
      parseFloat(analysis.month1 || 0), parseFloat(analysis.month2 || 0), par
      parseFloat(analysis.month4 || 0), parseFloat(analysis.month5 || 0), par
      parseFloat(analysis.month7 || 0), parseFloat(analysis.month8 || 0), par
      parseFloat(analysis.month10 || 0), parseFloat(analysis.month11 || 0), p
    ];

    const { months } = generateMonthRange();
```

Figure 5.4.28 Backend API Endpoints - GET /watch-analysis/:watchId

Distributor Module

1. POST /update-shipping

Updates shipping status with progressive validation through distribution network.

```
app.post("/update-shipping", async (req, res) => {
  const { watchId, shippingStatus, location, distributorAddress } = req.body;

  if (!watchId || !shippingStatus || !location || !distributorAddress) {
    return res.status(400).send("All fields are required");
  }

  const statusNum = parseInt(shippingStatus);
  if (isNaN(statusNum) || statusNum < 1 || statusNum > 3) {
    return res.status(400).send("Shipping status must be between 1 and 3");
  }

  try {
    const checkResult = await client.query(
      "SELECT watch_id, shipping_status, current_owner, ownership_history FR(
      [watchId]
    );

    if (checkResult.rows.length === 0) {
      return res.status(404).send("Watch not found");
    }

    const currentWatch = checkResult.rows[0];
    const currentStatus = parseInt(currentWatch.shipping_status) || 0;
```

Figure 5.4.29 Backend API Endpoints - POST /update-shipping

2. GET /watches/shipping

Retrieves watches with shipping status for logistics monitoring.

```
app.get("/watches/shipping", async (req, res) => {
  try {
    const data = await client.query(
      `SELECT watch_id, shipping_status, shipping_trail, distributor_address,
       FROM watches ORDER BY updated_at DESC`
    );

    res.json({ success: true, watches: data.rows, count: data.rows.length });
  } catch (err) {
    res.status(500).json({ success: false, message: "Error fetching watches",
  }
});
```

Figure 5.4.30 Backend API Endpoints - GET/watches/shipping

3. GET /shipping/stats

Provides shipping statistics across all status levels.

```
app.get("/shipping/stats", async (req, res) => {
  try {
    const stats = await client.query(`
      SELECT shipping_status, COUNT(*) as count
      FROM watches
      WHERE shipping_status IS NOT NULL
      GROUP BY shipping_status
      ORDER BY shipping_status
    `);

    const statusCounts = { 0: 0, 1: 0, 2: 0, 3: 0 };
    stats.rows.forEach((row) => {
      statusCounts[row.shipping_status] = parseInt(row.count);
    });

    res.json({
      success: true,
      statistics: statusCounts,
      labels: {
        0: "Not Shipped",
        1: "Shipped",
        2: "In Transit",
        3: "Delivered",
      },
    });
  } catch (err) {
    res.status(500).json({ success: false, message: "Error fetching shipping
  }
});
```

Figure 5.4.31 Backend API Endpoints - GET /shipping/stat

Retailer Module

1. POST /mark-available

Marks delivered watches as available for consumer purchase with pricing information.

```
app.post("/mark-available", async (req, res) => {
  const { watchId, retailerAddress, retailPrice, location, timestamp } = req.

  if (!watchId || !retailerAddress) {
    return res.status(400).json({ success: false, message: "Watch ID and reta
  }

  const priceValue = parseFloat(retailPrice);
  if (!retailPrice || isNaN(priceValue) || priceValue <= 0) {
    return res.status(400).json({ success: false, message: "Valid retail pric
  }

  try {
    const checkWatchResult = await client.query(
      `SELECT watch_id, shipping_status, available_for_sale, sold, retail_pri
       FROM watches WHERE watch_id = $1`,
      [watchId]
    );

    if (checkWatchResult.rows.length === 0) {
      return res.status(404).json({ success: false, message: "Watch not found
    }

    const watch = checkWatchResult.rows[0];
    const shippingStatus = parseInt(watch.shipping_status) || 0;
```

Figure 5.4.32 Backend API Endpoints - POST /mark-available

Consumer Module

1. POST /purchase-watch

Processes consumer purchases with atomic transactions ensuring data consistency.

```
app.post("/purchase-watch", async (req, res) => {
  const { watchId, buyerAddress, consumerTimestamp } = req.body;

  if (!watchId || !buyerAddress) {
    return res.status(400).json({ success: false, message: "Watch ID and buye
  }

  try {
    const checkResult = await client.query(
      `SELECT watch_id, available_for_sale, sold, retail_price, current_owner
       retailer_address, ownership_history, shipping_status
       FROM watches WHERE watch_id = $1`,
      [watchId]
    );

    if (checkResult.rows.length === 0) {
      return res.status(404).json({ success: false, message: "Watch not found
    }

    const watch = checkResult.rows[0];

    if (watch.sold === true) {
      return res.status(400).json({ success: false, message: "This watch has
    }
```

Figure 5.4.33 Backend API Endpoints - POST /purchase-watch

2. POST /transfer-ownership

Enables secondary market transfers between consumers with audit trail maintenance.

```
app.post("/transfer-ownership", async (req, res) => {
  const { watchId, newOwnerAddress, currentOwnerAddress, transferLocation, tr

  if (!watchId || !newOwnerAddress || !currentOwnerAddress) {
    return res.status(400).json({
      success: false,
      message: "Watch ID, current owner address, and new owner address are re
    });
  }

  if (newOwnerAddress.toLowerCase() === currentOwnerAddress.toLowerCase()) {
    return res.status(400).json({ success: false, message: "Cannot transfer c
  }

  try {
    await client.query("BEGIN");

    const checkResult = await client.query(
      `SELECT watch_id, current_owner, sold, ownership_history FROM watches W
      [watchId]
    );
```

Figure 5.4.34 Backend API Endpoints - POST /transfer-ownership

3. GET /watches/owner/:ownerAddress

Retrieves all watches owned by a specific address with AI analysis data.

```
app.get("/watches/owner/:ownerAddress", async (req, res) => {
  const { ownerAddress } = req.params;

  try {
    const data = await client.query(
      `SELECT w.*,
       CASE WHEN w.available_for_sale IS TRUE THEN true ELSE false END as availab
       CASE WHEN w.sold IS TRUE THEN true ELSE false END as sold,
       CASE WHEN ai.ai_analyzed IS TRUE THEN true ELSE false END as ai_analyzed,
       COALESCE(w.retail_price, 0.00) as retail_price,
       COALESCE(ai.ai_market_price, 0.00) as ai_market_price,
       COALESCE(ai.ai_confidence_score, 0.00) as ai_confidence_score,
       ai.ai_watch_model, ai.ai_analysis_date, w.last_transfer_timestamp,
       COALESCE(w.ownership_history, ARRAY[]::text[]) as ownership_history,
       ai.month1, ai.month2, ai.month3, ai.month4, ai.month5, ai.month6,
       ai.month7, ai.month8, ai.month9, ai.month10, ai.month11, ai.month12,
       ai.ai_trend_analysis, ai.ai_recommendations
       FROM watches w
       LEFT JOIN aimodel ai ON w.watch_id = ai.watch_id
       WHERE w.current_owner = $1
       ORDER BY w.last_transfer_timestamp DESC NULLS LAST, w.updated_at DESC, w.c
      [ownerAddress]
    );

    res.json({ success: true, watches: data.rows, count: data.rows.length, owner:
  } catch (err) {
    res.status(500).json({ success: false, message: "Error fetching owned watches
  }
});
```

Figure 5.4.35 Backend API Endpoints - GET/ watches/owner/:ownerAddress

4. GET /consumer/stats/:ownerAddress

Provides consumer statistics with timestamp analysis.

```
app.get("/consumer/stats/:ownerAddress", async (req, res) => {
  const { ownerAddress } = req.params;

  try {
    const consumerData = await client.query(
      `SELECT
         COUNT(*) as total_owned,
         COUNT(CASE WHEN w.consumer_timestamp IS NOT NULL THEN 1 END) as purchase
         COUNT(CASE WHEN w.last_transfer_timestamp IS NOT NULL THEN 1 END) as tra
         AVG(w.retail_price) as avg_purchase_price,
         MIN(w.consumer_timestamp) as first_purchase,
         MAX(w.consumer_timestamp) as latest_purchase,
         AVG(ai.ai_market_price) as avg_ai_value
       FROM watches w
       LEFT JOIN aimodel ai ON w.watch_id = ai.watch_id
       WHERE w.current_owner = $1`,
      [ownerAddress]
    );

    const stats = consumerData.rows[0];

    res.json({
      success: true,
      owner: ownerAddress,
      statistics: {
        totalOwnedWatches: parseInt(stats.total_owned || 0),
        purchasedByUser: parseInt(stats.purchased_by_user || 0),
        transferredWatches: parseInt(stats.transferred_watches || 0),
        averagePurchasePrice: parseFloat(stats.avg_purchase_price || 0),
        averageAiValue: parseFloat(stats.avg_ai_value || 0),
        firstPurchase: stats.first_purchase,
        latestPurchase: stats.latest_purchase,
```

Figure 5.4.36 Backend API Endpoints - GET/ consumer/stats/:ownerAddress

5. GET /watch/:watchId/transfer-history

Retrieves complete transfer audit trail for a specific watch.

```
app.get("/watch/:watchId/transfer-history", async (req, res) => {
  const { watchId } = req.params;

  try {
    const transferHistory = await client.query(
      `SELECT id, from_address, to_address, transfer_type, transfer_timestamp, tr
       FROM ownership_transfers WHERE watch_id = $1 ORDER BY transfer_timestamp D
      [watchId]
    );

    const watchData = await client.query(
      `SELECT watch_id, current_owner, ownership_history, last_transfer_timestamp
       distributor_address, retailer_address, retailer_timestamp, consumer_timest
       FROM watches WHERE watch_id = $1`,
      [watchId]
    );

    if (watchData.rows.length === 0) {
      return res.status(404).json({ success: false, message: "Watch not found" })
    }

    const watch = watchData.rows[0];

    res.json({
      success: true,
      data: {
```

Figure 5.4.37 Backend API Endpoints - GET/ watch/:watchId/transfer-history

File Upload Module

1. POST /upload/profile

Handles profile image uploads with validation.

```
app.post("/upload/profile", upload.profile.single("image"), (req, res) => {
  if (!req.file) {
    return res.status(400).json({ success: 0, message: "No file uploaded" });
  }
  res.json({
    success: 1,
    filename: req.file.filename,
    message: "Profile image uploaded successfully",
  });
});
```

Figure 5.4.38 Backend API Endpoints - POST/ upload/profile

2. POST /upload/raw-material

Handles raw material image uploads.

```
app.post("/upload/raw-material", upload.rawMaterial.single("image"), (req, res) =
  if (!req.file) {
    return res.status(400).send("No file uploaded");
  }
  res.json({
    filename: req.file.filename,
    message: "Raw material image uploaded successfully",
  });
});
```

Figure 5.4.39 Backend API Endpoints - POST/ upload/ raw-material

3. POST /upload/component

Handles component image uploads.

```
app.post("/upload/component", upload.component.single("image"), (req, res) => {
  if (!req.file) {
    return res.status(400).send("No file uploaded");
  }
  res.json({
    filename: req.file.filename,
    message: "Component image uploaded successfully",
  });
});
```

Figure 5.4.40 Backend API Endpoints - POST/ upload/ component

4. POST /upload/watch

Handles watch image uploads for NFT integration.

```
app.post("/upload/watch", upload.watch.single("image"), (req, res) => {
  if (!req.file) {
    return res.status(400).send("No file uploaded");
  }
  res.json({
    filename: req.file.filename,
    message: "Watch image uploaded successfully",
  });
});
```

Figure 5.4.41 Backend API Endpoints - POST/ upload/ watch

5. POST /upload/certifier

Handles certifier evidence image uploads.

```
app.post("/upload/certifier", upload.certifier.single("image"), (req, res) => {
  if (!req.file) {
    return res.status(400).send("No file uploaded");
  }
  res.json({
    filename: req.file.filename,
    message: "Certifier image uploaded successfully",
  });
});
```

Figure 5.4.42 Backend API Endpoints - POST/ upload/ certifier

6. GET /file/profile/:filename

Serves uploaded profile images.

```
app.get("/file/profile/:fileName", function (req, res) {
  const { fileName } = req.params;
  const filePath = path.join(__dirname, "public/uploads/profile", fileName);
  res.sendFile(filePath, (err) => {
    if (err) {
      res.status(404).send("Image not found");
    }
  });
});
```

Figure 5.4.43 Backend API Endpoints - GET/file/profile/:filename

7. GET /file/raw-material/:filename

Serves uploaded raw material images.

```
app.get("/file/raw-material/:fileName", function (req, res) => {
  const { fileName } = req.params;
  const filePath = path.join(__dirname, "public/uploads/raw-material", fileName);
  res.sendFile(filePath, (err) => {
    if (err) {
      res.status(404).send("Image not found");
    }
  });
});
```

Figure 5.4.44 Backend API Endpoints - GET/file/raw-material/:filename

8. GET /file/component/:filename

Serves uploaded component images.

```
app.get("/file/component/:fileName", function (req, res) {
  const { fileName } = req.params;
  const filePath = path.join(__dirname, "public/uploads/component", fileName);
  res.sendFile(filePath, (err) => {
    if (err) {
      res.status(404).send("Image not found");
    }
  });
});
```

Figure 5.4.45 Backend API Endpoints - GET/file/component/:filename

9. GET /file/watch/:filename

Serves uploaded watch images with enhanced validation.

```
app.get("/file/watch/:fileName", function (req, res) => {
  const { fileName } = req.params;

  if (!fileName || fileName.trim() === "" || fileName === "null" || fileName ===
    return res.status(400).json({ error: "Invalid filename", filename: fileName }
  }

  const filePath = path.join(__dirname, "public/uploads/watch", fileName);

  if (!fs.existsSync(filePath)) {
    return res.status(404).json({
      error: "Image not found",
      filename: fileName,
      suggestion: "Check if the file was uploaded correctly",
    });
  }

  res.sendFile(filePath, (err) => {
    if (err) {
      res.status(500).json({ error: "Error serving image", filename: fileName, de
    }
  });
});
```

Figure 5.4.46 Backend API Endpoints - GET/file/watch/:filename

10. GET /file/certifier/:filename

Serves uploaded certifier evidence images.

```
app.get("/file/certifier/:fileName", function (req, res) => {
  const { fileName } = req.params;
  const filePath = path.join(__dirname, "public/uploads/certifier", fileName);
  res.sendFile(filePath, (err) => {
    if (err) {
      res.status(404).send("Image not found");
    }
  });
});
```

Figure 5.4.47 Backend API Endpoints - GET/file/certifier/:filename

Search and Analytics Module

1. GET /search/raw-materials

Provides advanced search functionality for raw materials.

```
app.get("/search/raw-materials", async (req, res) => {
  const { query, type, status, supplier } = req.query;

  try {
    let sqlQuery = "SELECT * FROM raw_materials WHERE 1=1";
    const params = [];
    let paramCount = 0;

    if (query) {
      paramCount++;
      sqlQuery += ` AND (component_id ILIKE $${paramCount} OR material_type ILIKE
      params.push(`%${query}%`);
    }
```

Figure 5.4.48 Backend API Endpoints - GET/ search/raw-materials

2. GET /search/components

Provides advanced search functionality for components.

```
app.get("/search/components", async (req, res) => {
  const { query, type, status, manufacturer } = req.query;

  try {
    let sqlQuery = "SELECT * FROM components WHERE 1=1";
    const params = [];
    let paramCount = 0;

    if (query) {
      paramCount++;
      sqlQuery += ` AND (component_id ILIKE $${paramCount} OR component_type ILIK
      params.push(`%${query}%`);
    }
```

Figure 5.4.49 Backend API Endpoints - GET/ search/components

3. GET /search/watches

Provides advanced search functionality for watches with AI analysis filtering.

```
app.get("/search/watches", async (req, res) => {
  const { query, shipping_status, available, assembler, price_min, price_max, ai_

  try {
    let sqlQuery = `
      SELECT w.*,
      CASE WHEN ai.ai_analyzed IS TRUE THEN true ELSE false END as ai_analyzed,
      ai.ai_watch_model, ai.ai_market_price, ai.ai_confidence_score,
      CASE WHEN w.last_transfer_timestamp IS NOT NULL THEN true ELSE false END as
      FROM watches w
      LEFT JOIN aimodel ai ON w.watch_id = ai.watch_id
      WHERE 1=1
    `.
```

Figure 5.4.50 Backend API Endpoints - GET/ search/watches

5.5 Frontend Implementation

The frontend implementation should present differentiated user experiences that align with each stakeholder's operational requirements and responsibilities within the luxury watch supply chain ecosystem. The existing role-based architecture provides an excellent foundation for implementing specialized dashboards that cater to distinct workflow patterns.

1. Supplier Module Dashboard Implementation

The supplier dashboard should emphasize raw material registration and inventory management capabilities. Suppliers require comprehensive tools for documenting material provenance, including origin tracking, quality certifications, and blockchain registration functionality. The interface should facilitate efficient raw material registration with integrated IPFS upload capabilities for supporting documentation. Real-time inventory tracking with QR code generation enables seamless integration with downstream manufacturing processes. The dashboard should provide analytics on material utilization rates and demand patterns to support procurement planning decisions.

Step 1: Log in to Supplier Dashboard

Figure 5.5.1 Supplier Dashboard

Step 2: Select the inputs and click the “Authenticate Material” button.

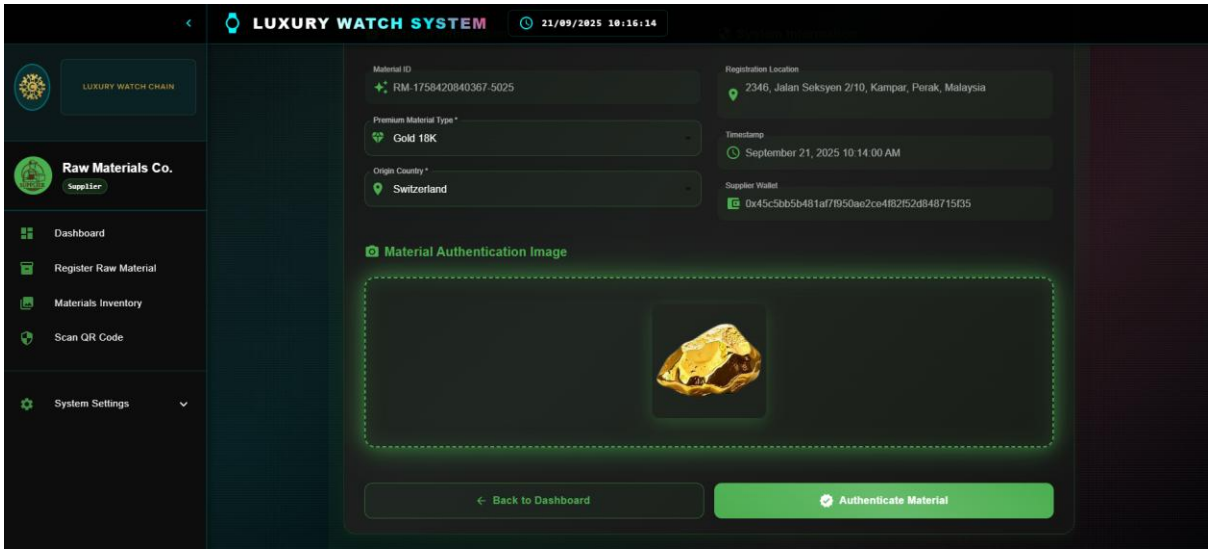

Figure 5.5.2 Register Raw Material in Supplier Dashboard

Step 3: Click the “Confirm” button in the MetaMask extension to allow the transaction.

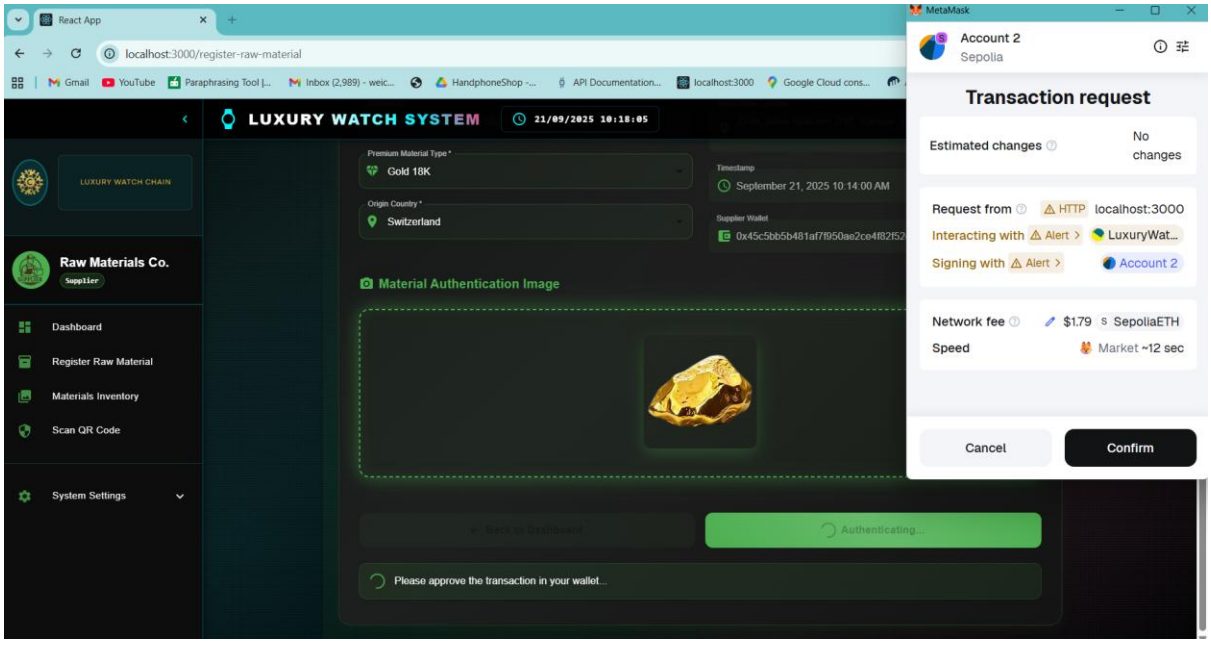

Figure 5.5.3 Confirm MetaMask Transaction in Supplier Dashboard

Step 4: A QR Code will appear once it has successfully run.

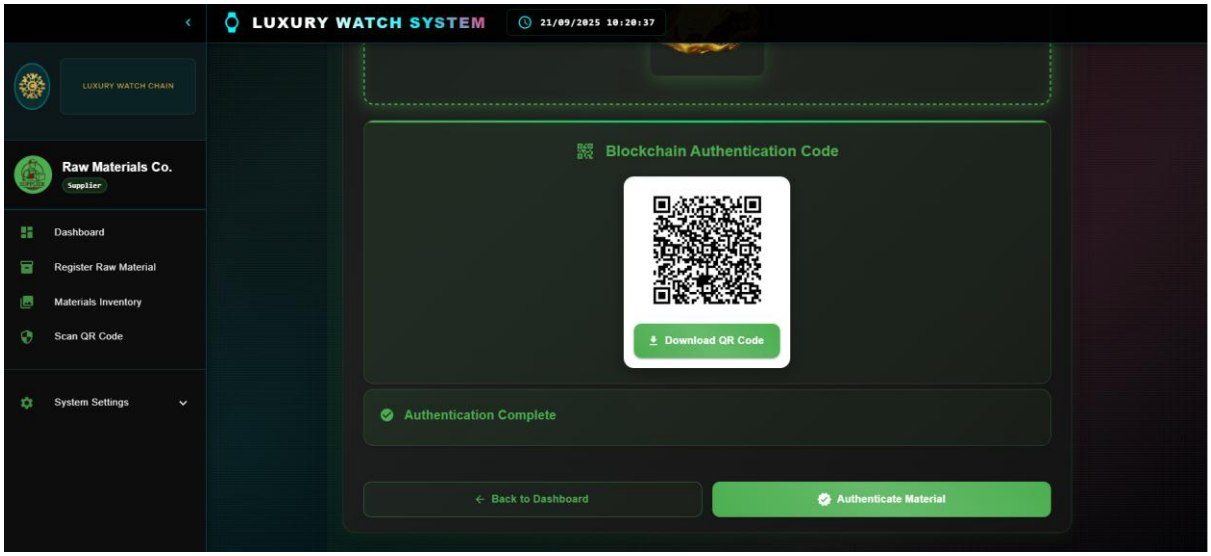

Figure 5.5.4 Raw Material QR Code

Step 5: Click "Materials Inventory" to check the registered raw material.

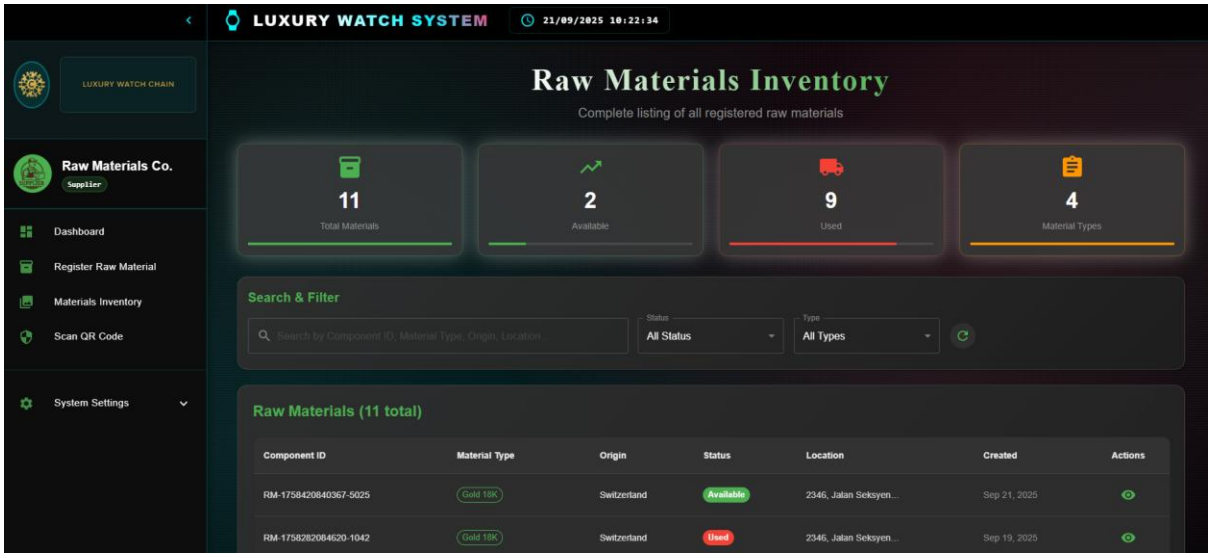

Figure 5.5.5 Raw Materials Inventory

2. Manufacturer Module Dashboard Implementation

Manufacturers require sophisticated component creation and management interfaces that integrate seamlessly with raw material sourcing. The dashboard should provide component assembly workflows that automatically verify raw material authenticity through blockchain

verification. Quality control interfaces should enable detailed component specification recording with batch tracking capabilities. Integration with certification workflows ensures proper documentation flow to certifying authorities. The interface should support bulk operations for high-volume manufacturing scenarios while maintaining detailed traceability records for each component produced.

Step 1: Log in to the Manufacturer Dashboard

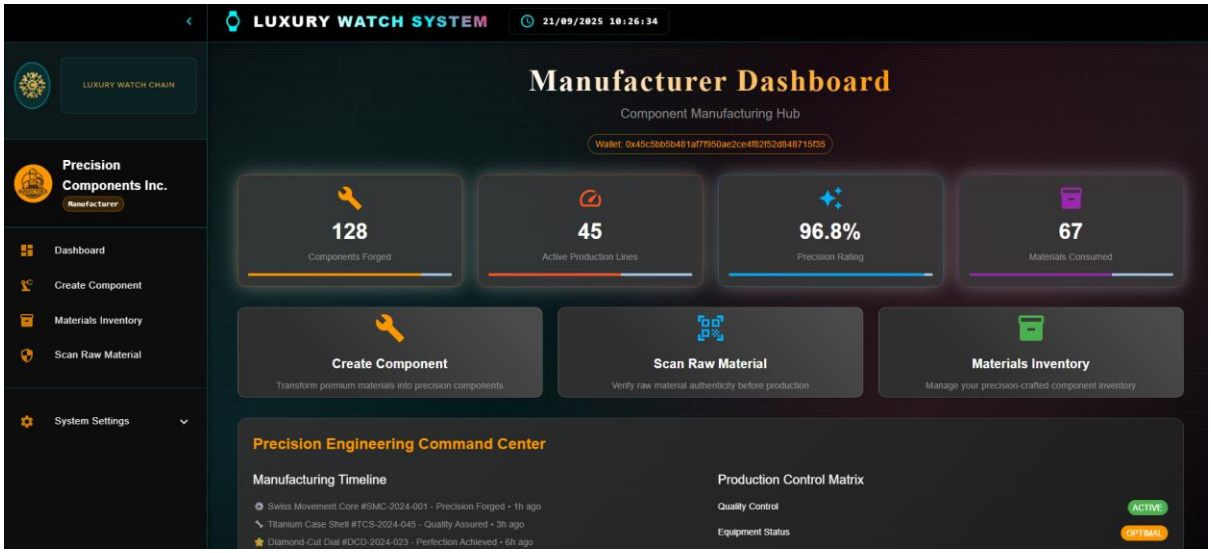

Figure 5.5.6 Manufacturer Dashboard

Step 2: Scan the registered raw material QR Code and select the inputs. Once done, click the "Forge Component" button.

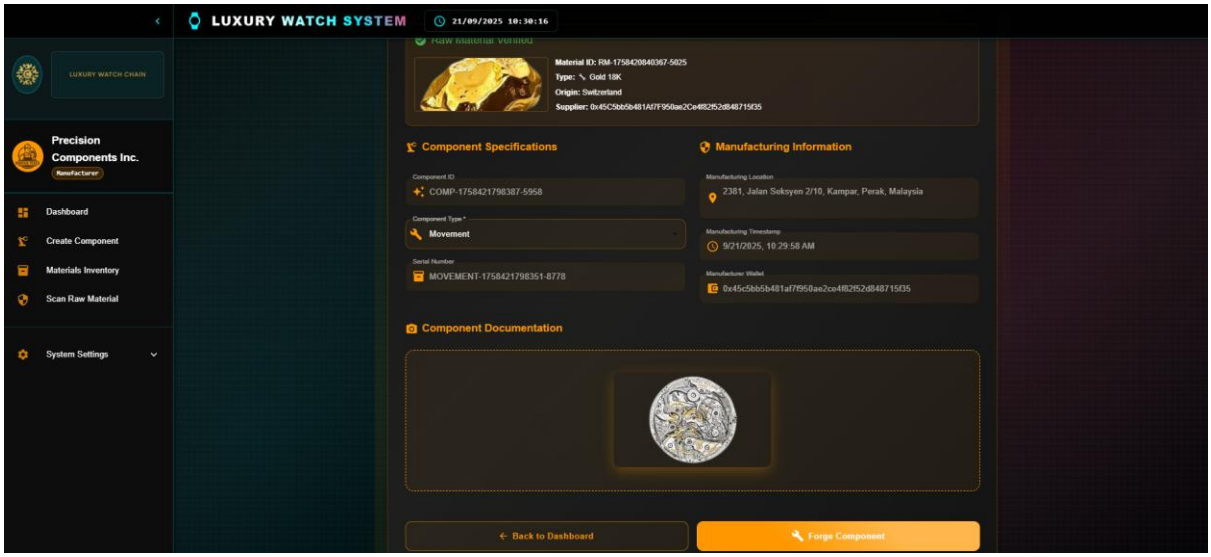

Figure 5.5.7 Create Component in Manufacturer Dashboard

Step 3: Click the “Confirm” button in the MetaMask extension to allow the transaction.

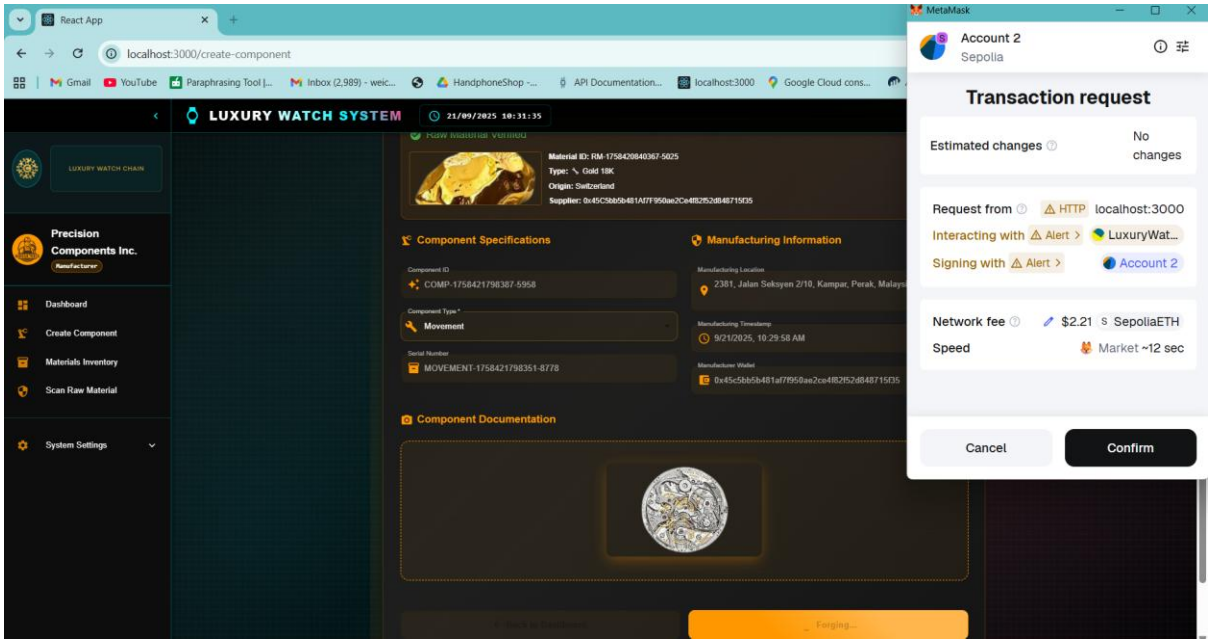

Figure 5.5.8 Confirm MetaMask Transaction in Manufacturer Dashboard

Step 4: A QR Code will appear once it has successfully run.

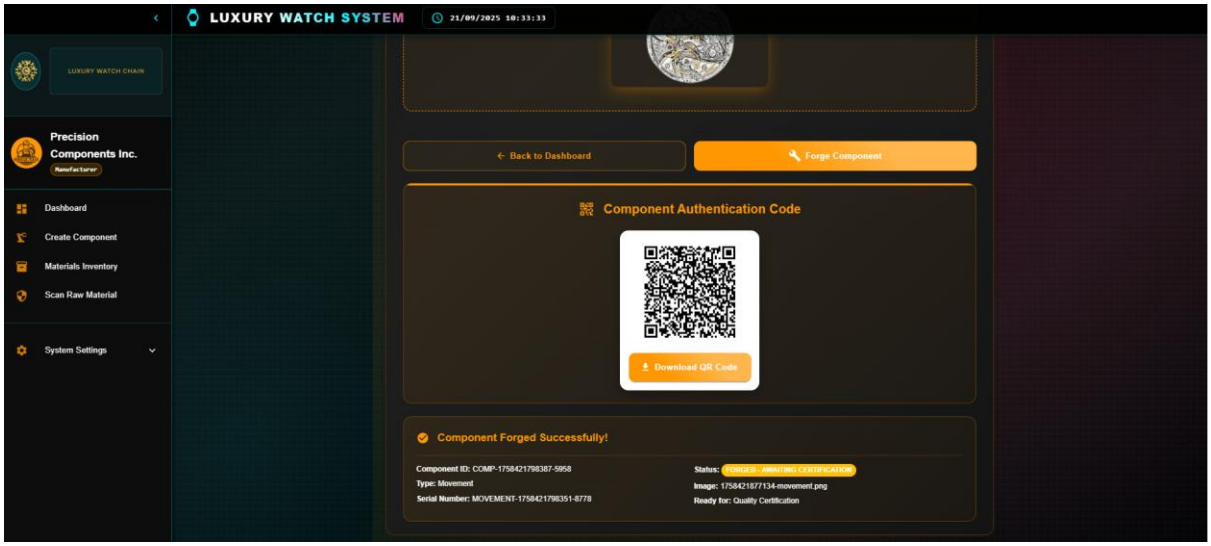

Figure 5.5.9 Component QR Code

Step 5: Click “Materials Inventory” to check the created component.

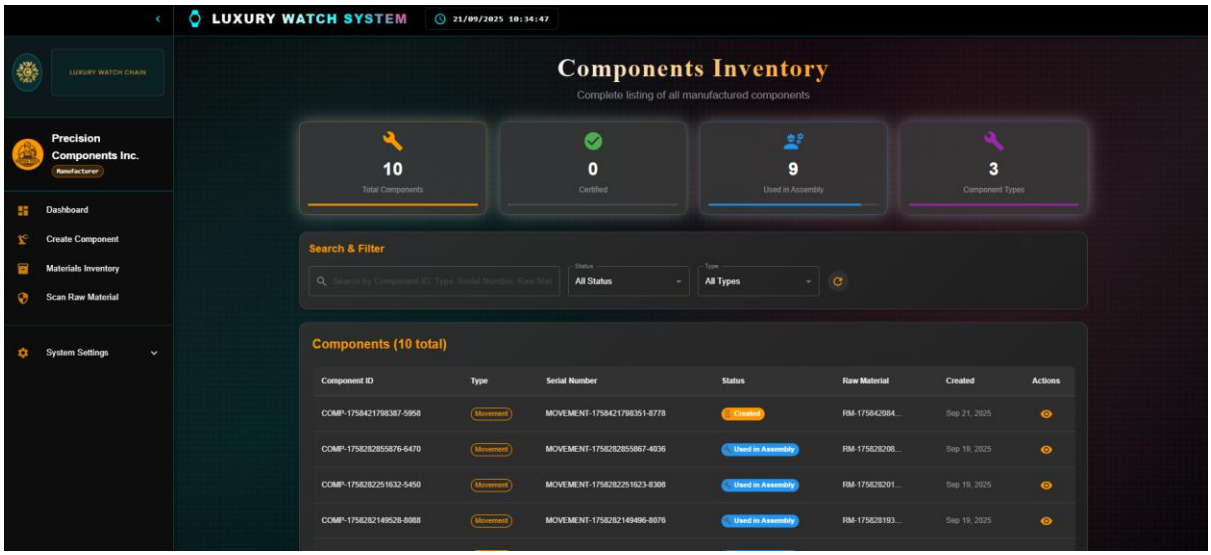

Figure 5.5.10 Components Inventory

3. Certifier Module Dashboard Implementation

The certifier dashboard should prioritize component verification workflows with comprehensive audit trail capabilities. Certification interfaces should provide detailed inspection checklists, quality assessment tools, and digital signature capabilities for blockchain-based certification records. The dashboard should enable batch certification processes while maintaining individual component verification standards. Integration with smart contract certification functions ensures immutable certification records that cannot be subsequently modified or falsified.

Step 1: Log in to the Certifier Dashboard

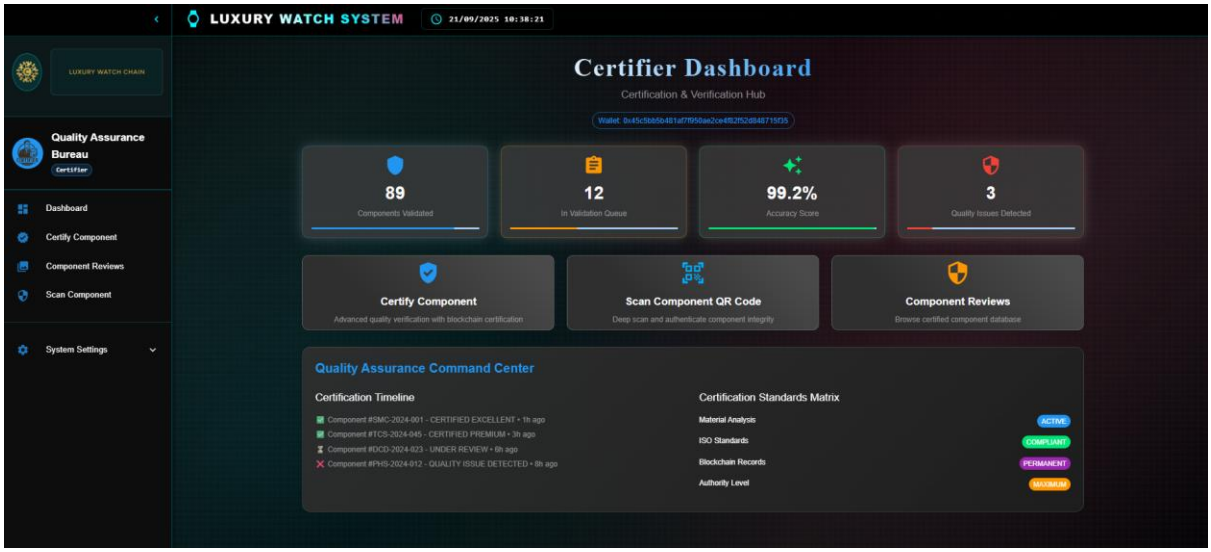

Figure 5.5.11 Certifier Dashboard

Step 2: Scan the created component QR Code and fill in the inputs. Once done, click the "Certify Component" button.

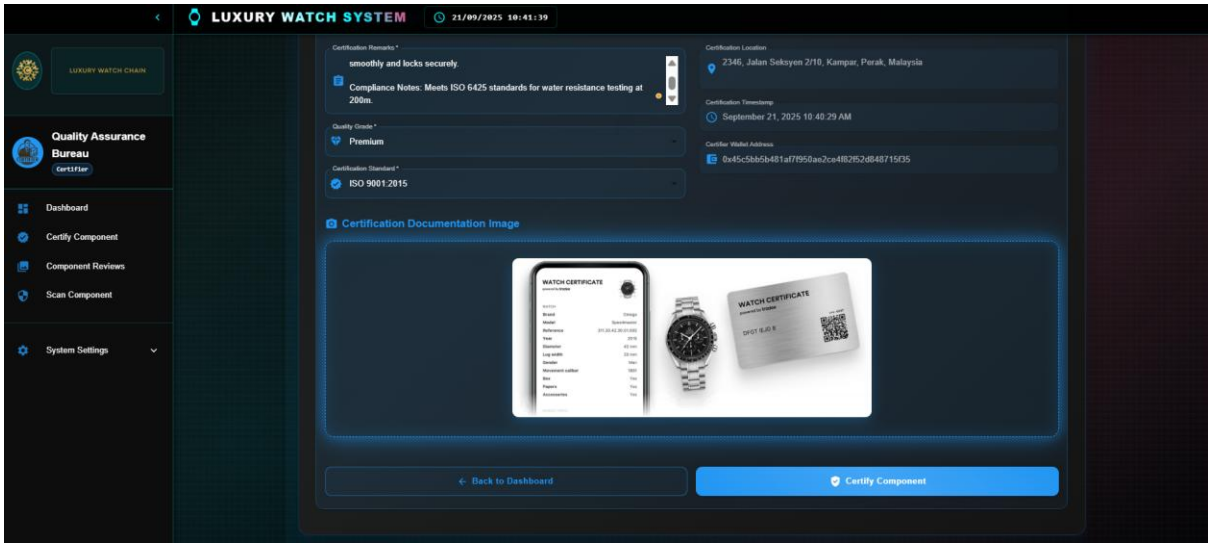

Figure 5.5.12 Certify Component in Certifier Dashboard

Step 3: Click the "Confirm" button in the MetaMask extension to allow the transaction.

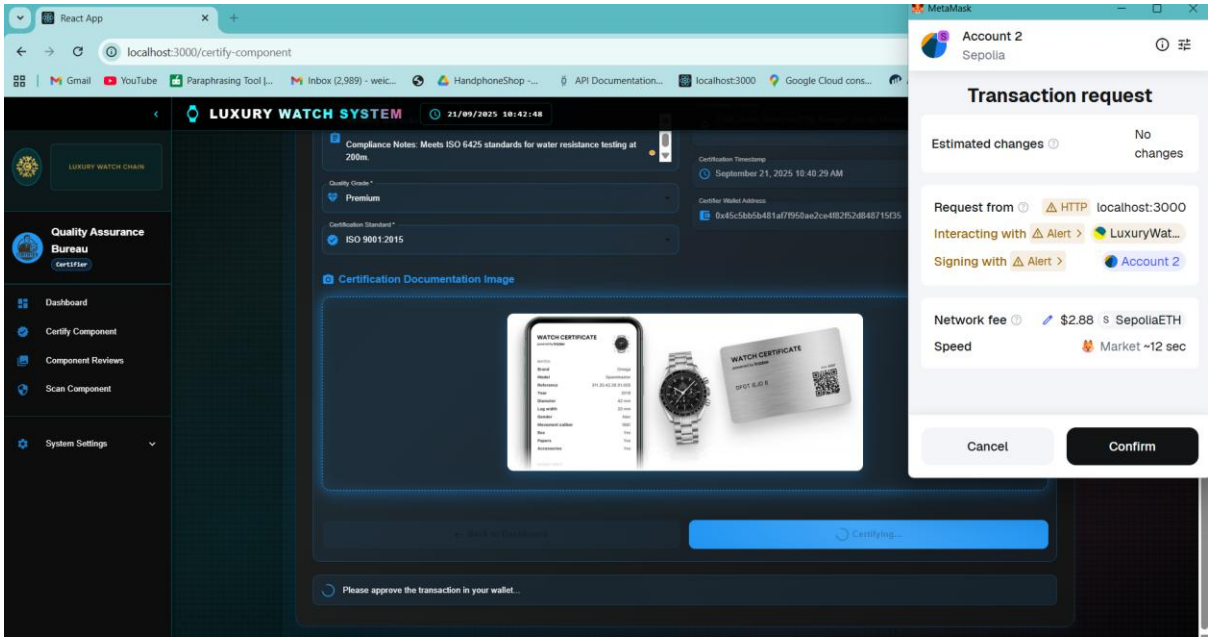

Figure 5.5.13 Confirm MetaMask Transaction in Certifier Dashboard

Step 4: A successful message will appear once it is done.

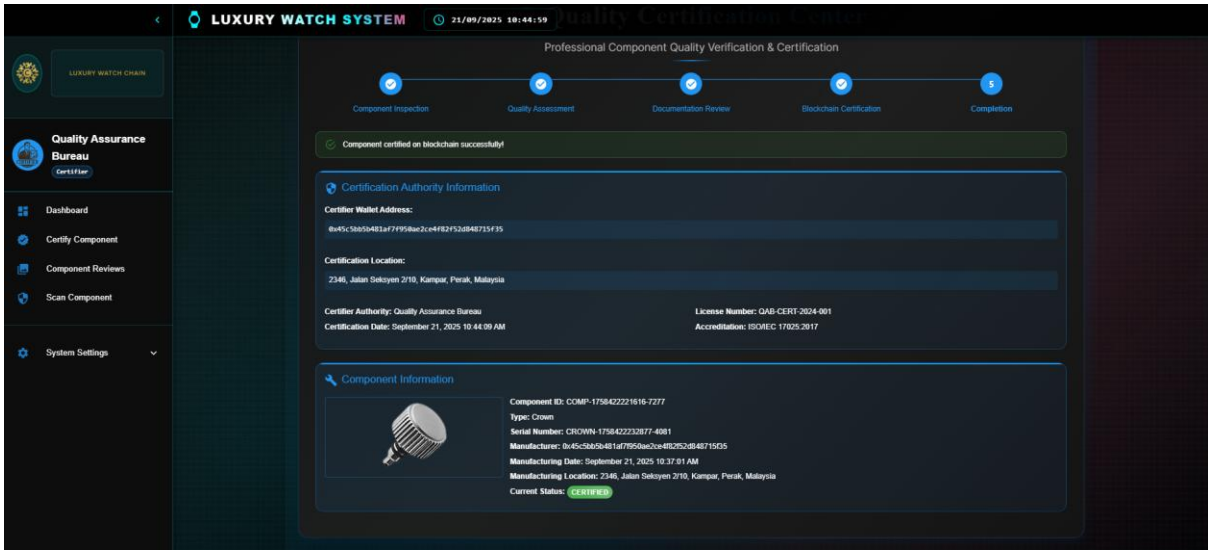

Figure 5.5.14 Successful Message in Certifier Dashboard

4. Assembler Module Dashboard Implementation

Assemblers require sophisticated watch assembly interfaces that verify component certification status before assembly authorization. The dashboard should provide component scanning

interfaces that validate certification status through blockchain verification. Assembly workflow management should guide users through required assembly steps while maintaining detailed assembly records. NFT minting integration should occur automatically upon successful watch assembly, with IPFS metadata generation including comprehensive component provenance data.

Step 1: Log in to the Assembler Dashboard

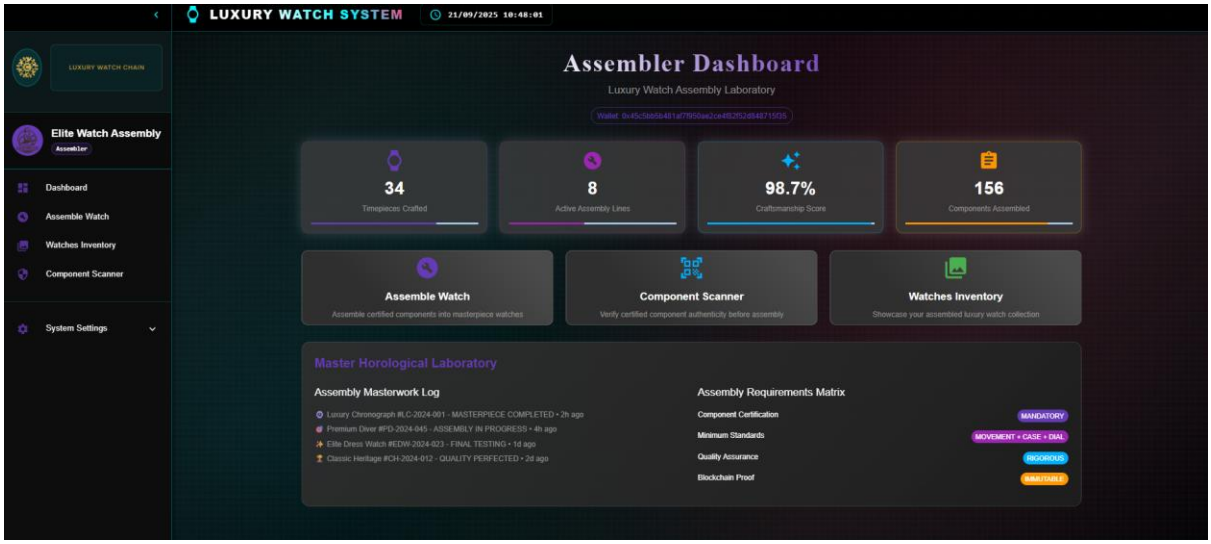

Figure 5.5.15 Assembler Dashboard

Step 2: Scan the certified components that meet the minimum of 3 certified components to assemble a watch, and then click the “Mint NFT Watch” button. At the same time, the watch will be minted as an NFT.

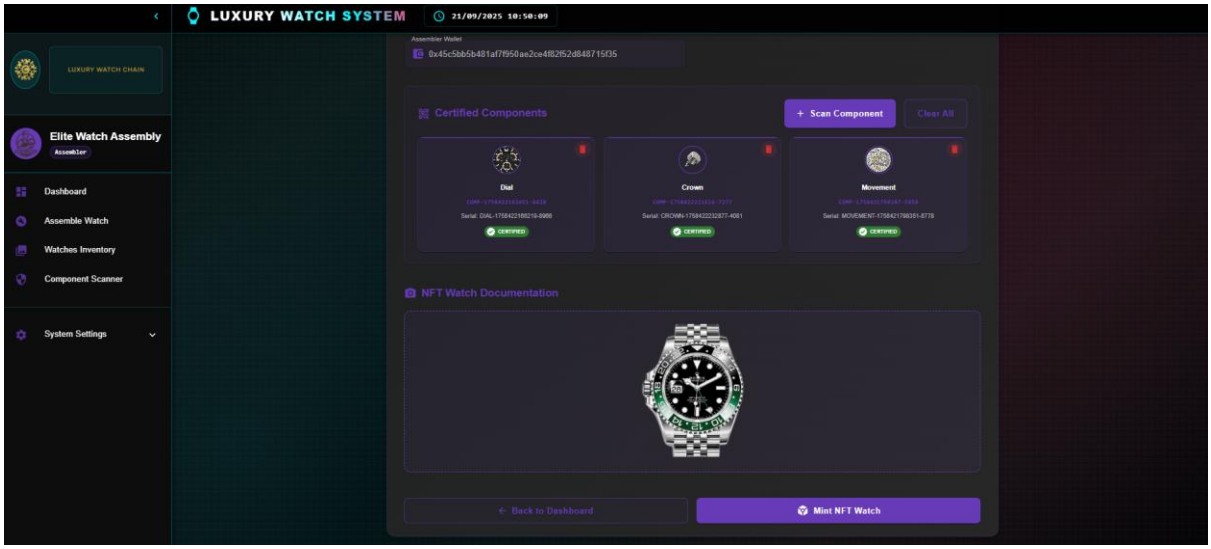

Figure 5.5.16 Assemble NFT Watch in Assembler Dashboard

Step 3: Click the “Confirm” button in the MetaMask extension to allow the transaction.

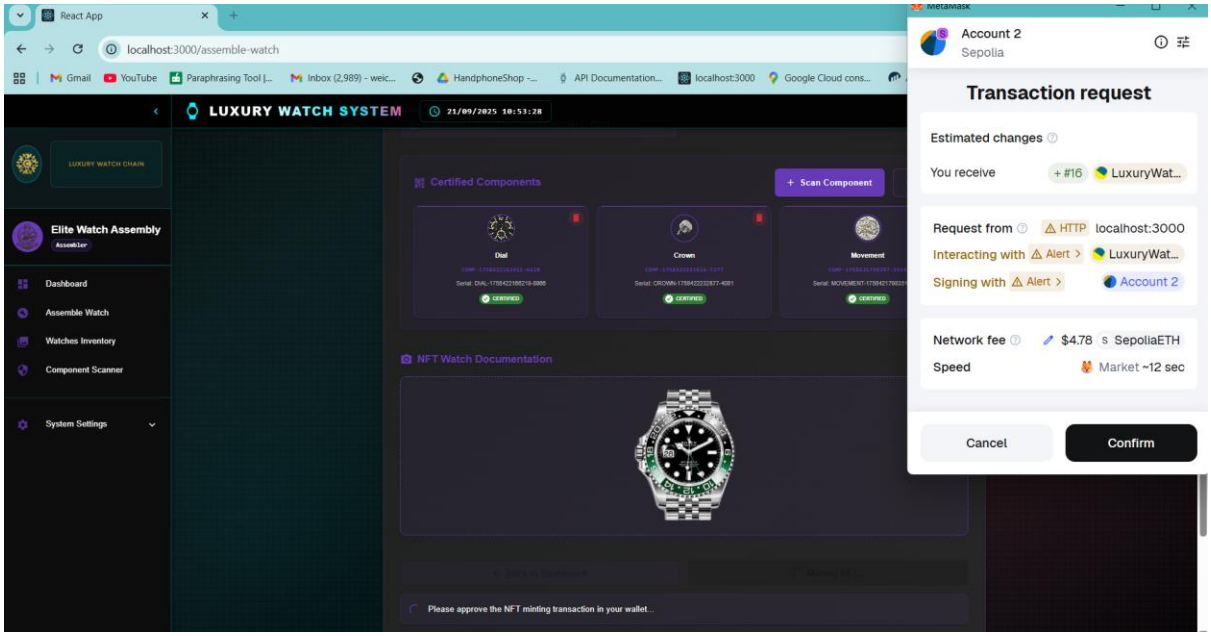

Figure 5.5.17 Confirm MetaMask Transaction in Assembler Dashboard

Step 4: A QR Code will appear once it has successfully run.

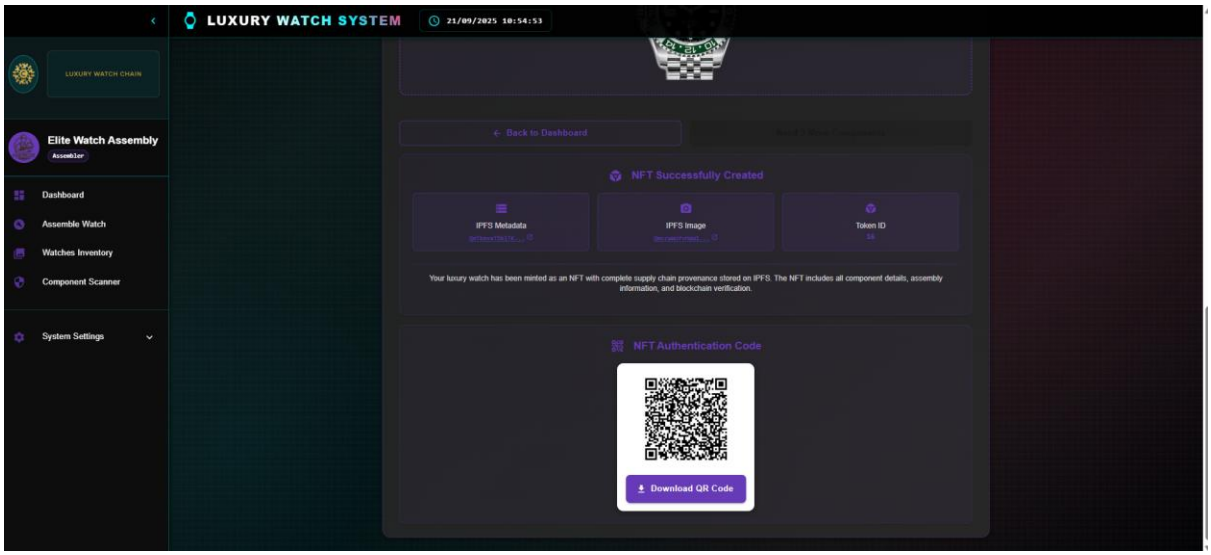

Figure 5.5.18 Watch QR Code

Step 5: Click “Watches Inventory” to check the assembled watch.

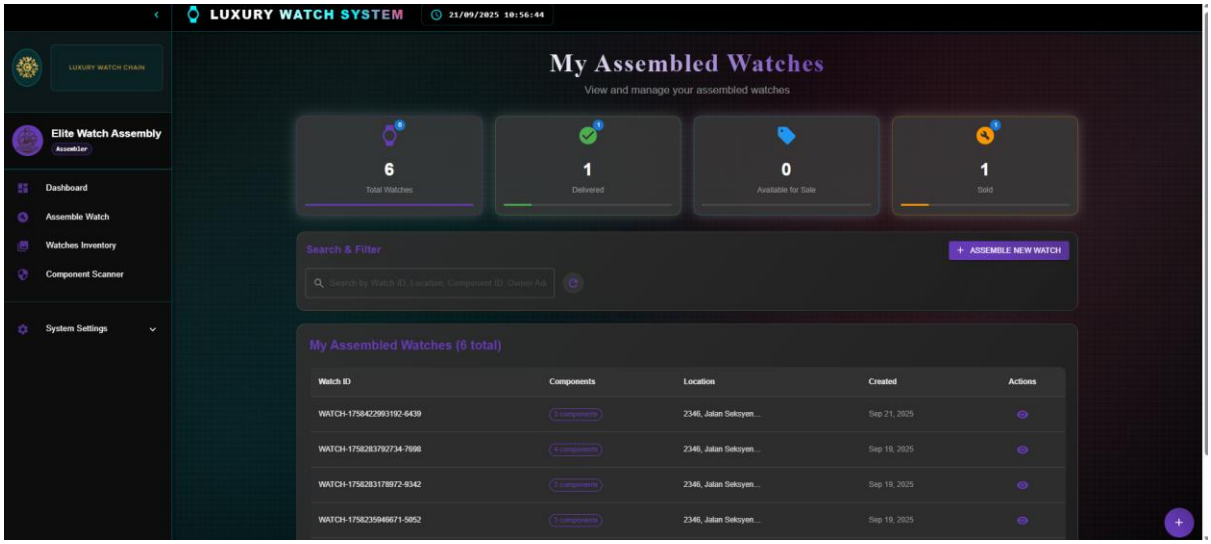

Figure 5.5.19 Watches Inventory

5. Distributor Module Dashboard Implementation

The distributor dashboard should emphasize shipping and logistics management with real-time tracking capabilities. Inventory management interfaces should provide watch status tracking, shipping documentation, and delivery confirmation workflows. Integration with NFT metadata update functions ensures accurate provenance tracking throughout distribution processes. The dashboard should support bulk shipping operations while maintaining individual watch tracking capabilities through QR code integration.

Step 1: Log in to the Distributor Dashboard

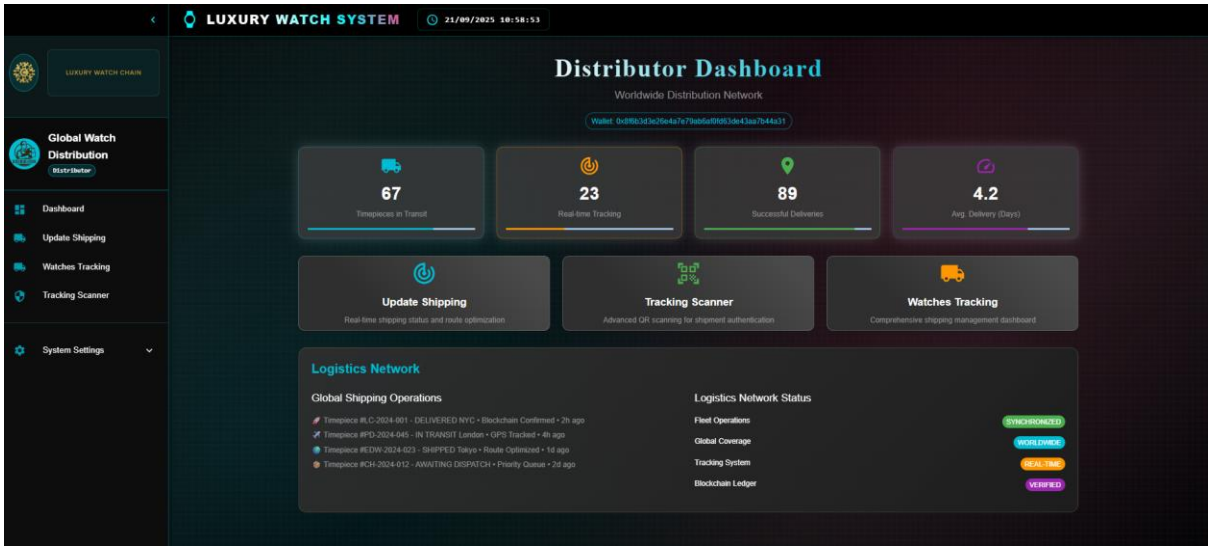

Figure 5.5.20 Distributor Dashboard

Step 2: Scan the watch QR code and click the “Update NFT Shipping Status” button to update the shipping status of the watch.

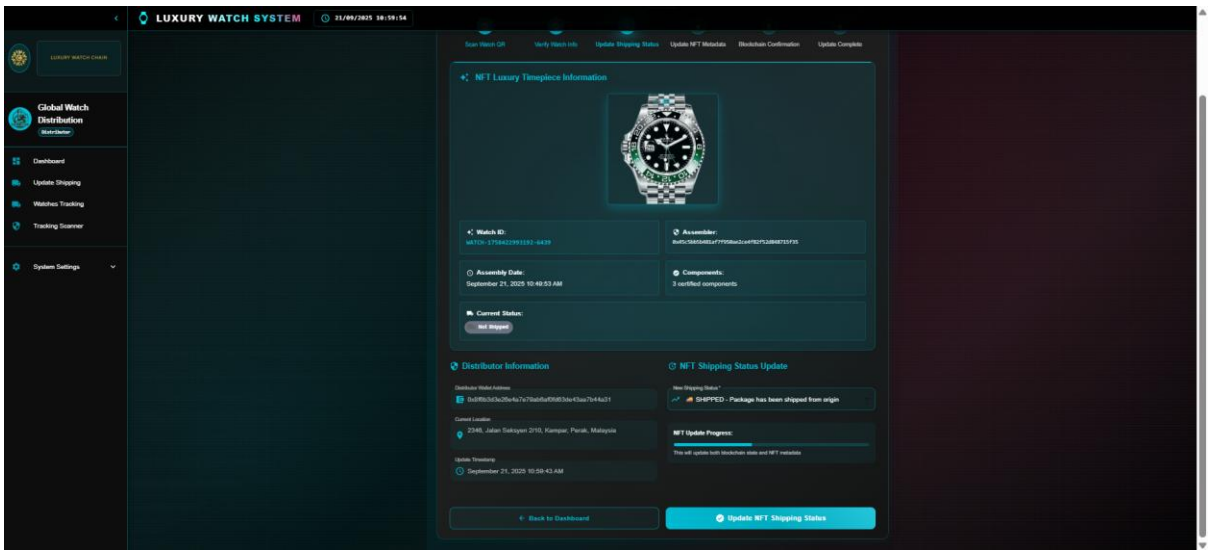

Figure 5.5.21 Update NFT Shipping Status

Step 3: Click the “Confirm” button in the MetaMask extension to allow the transaction.

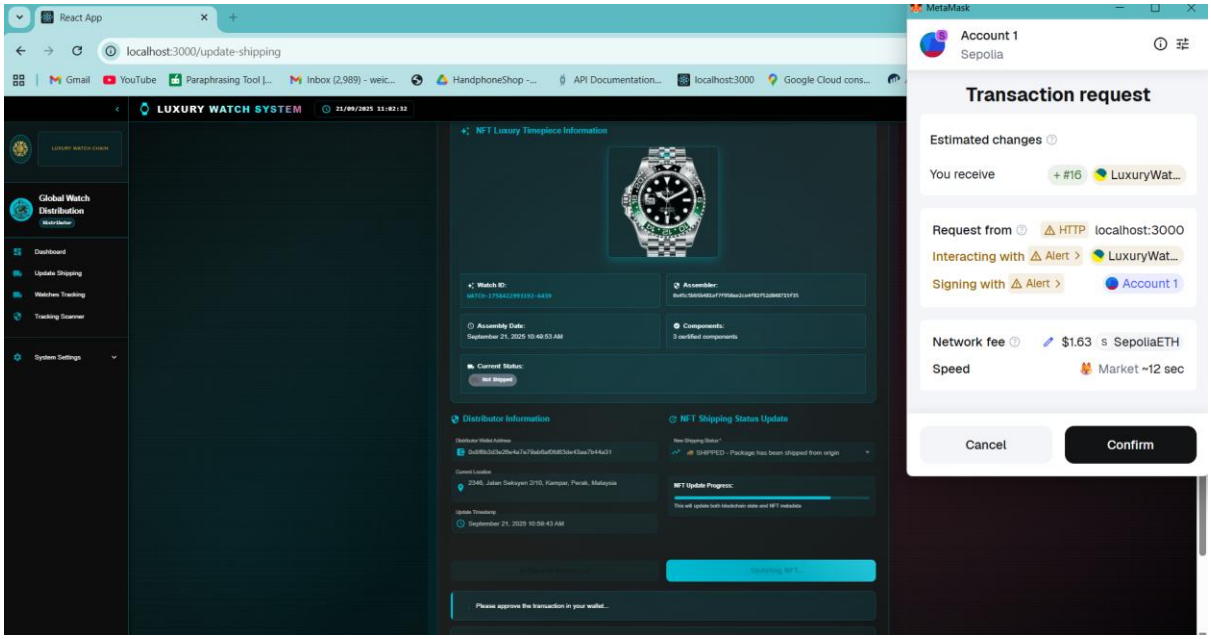

Figure 5.5.22 Confirm MetaMask Transaction in Distributor Dashboard

Step 4: Click “Show Map” to view the shipping timeline

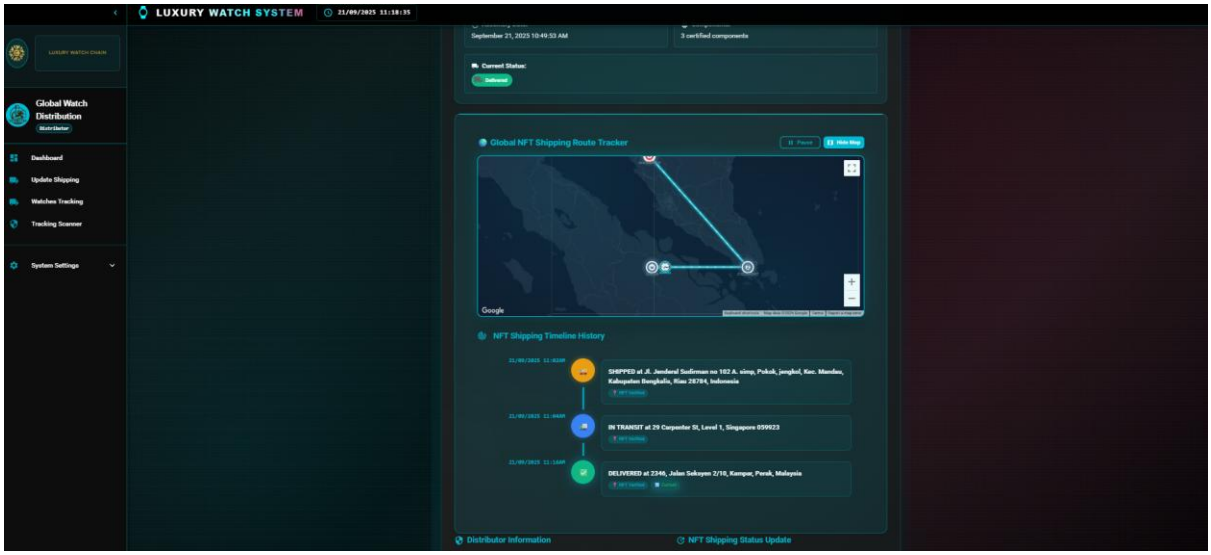

Figure 5.5.23 NFT Shipping Timeline

Step 5: Click “Watches Tracking” to view the shipping of the watch.

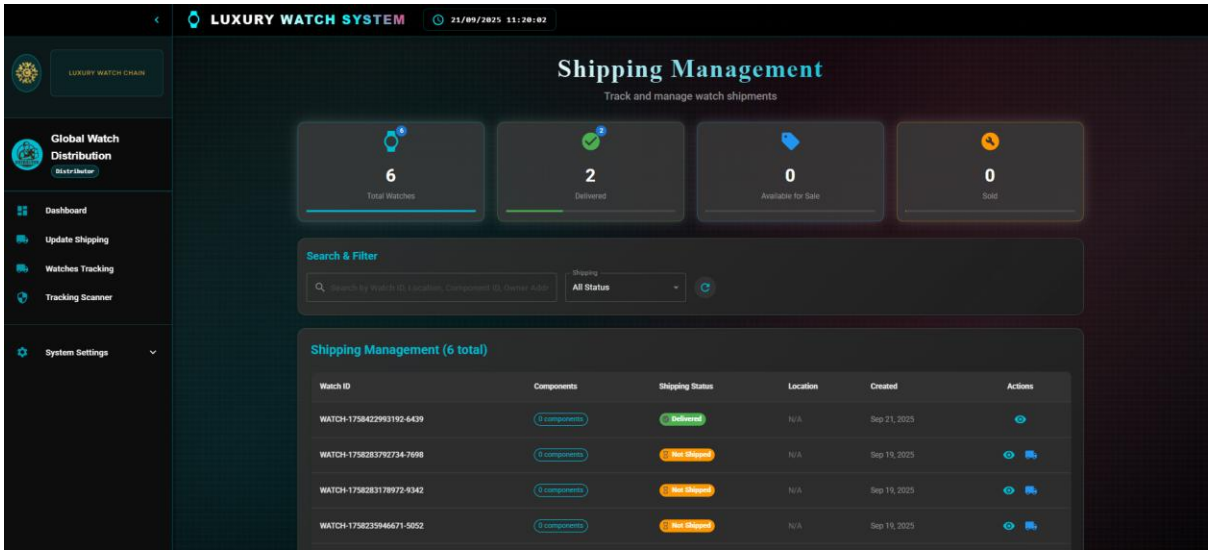

Figure 5.5.24 Shipping Management

6. Retailer Module Dashboard Implementation

Retailers require sophisticated sales management interfaces with customer-facing features for luxury goods presentation. The dashboard should provide inventory display capabilities with high-quality image presentation and detailed specification access. Sales workflow management should integrate with NFT ownership transfer functions, ensuring proper blockchain-based ownership updates. Customer verification interfaces should enable luxury goods authentication through blockchain verification. Integration with market pricing data supports dynamic pricing strategies based on provenance and market conditions.

Step 1: Log in to the Retailer Dashboard

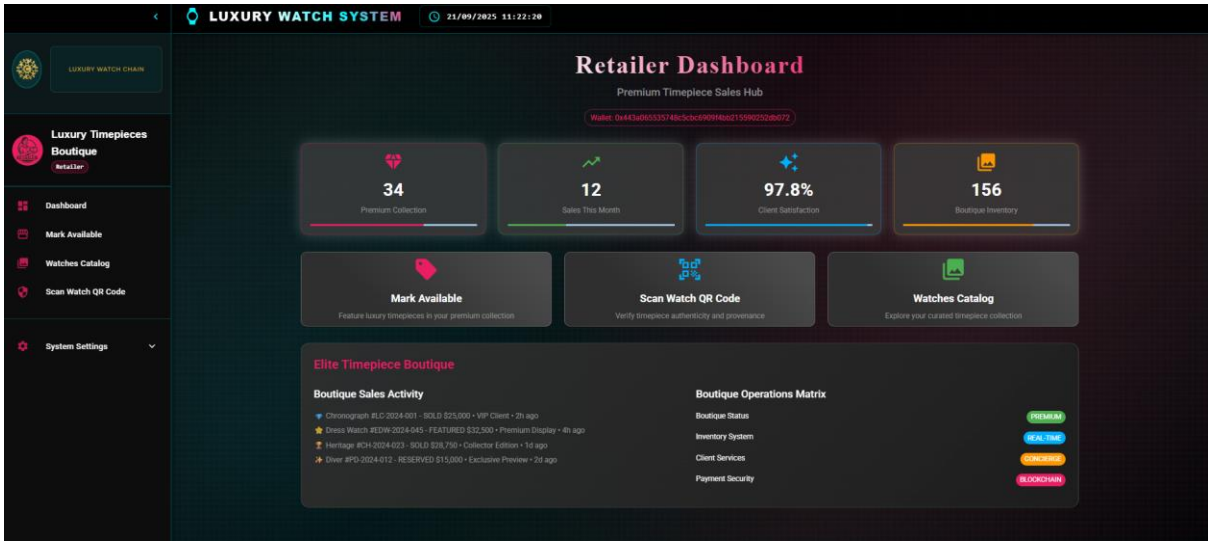

Figure 5.5.25 Retailer Dashboard

Step 2: Scan the watch QR Code and fill in the watch price. Next, click the “List NFT for Sale” button to make transactions.

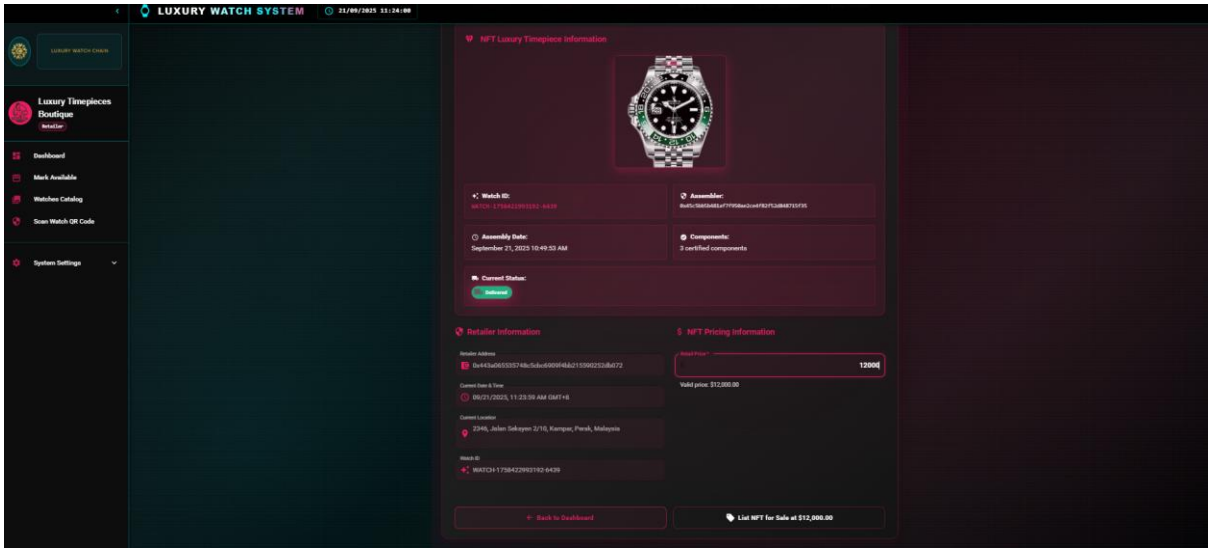

Figure 5.5.26 Set Watch Pricing

Step 3: Click the “Confirm” button in the MetaMask extension to allow the transaction.

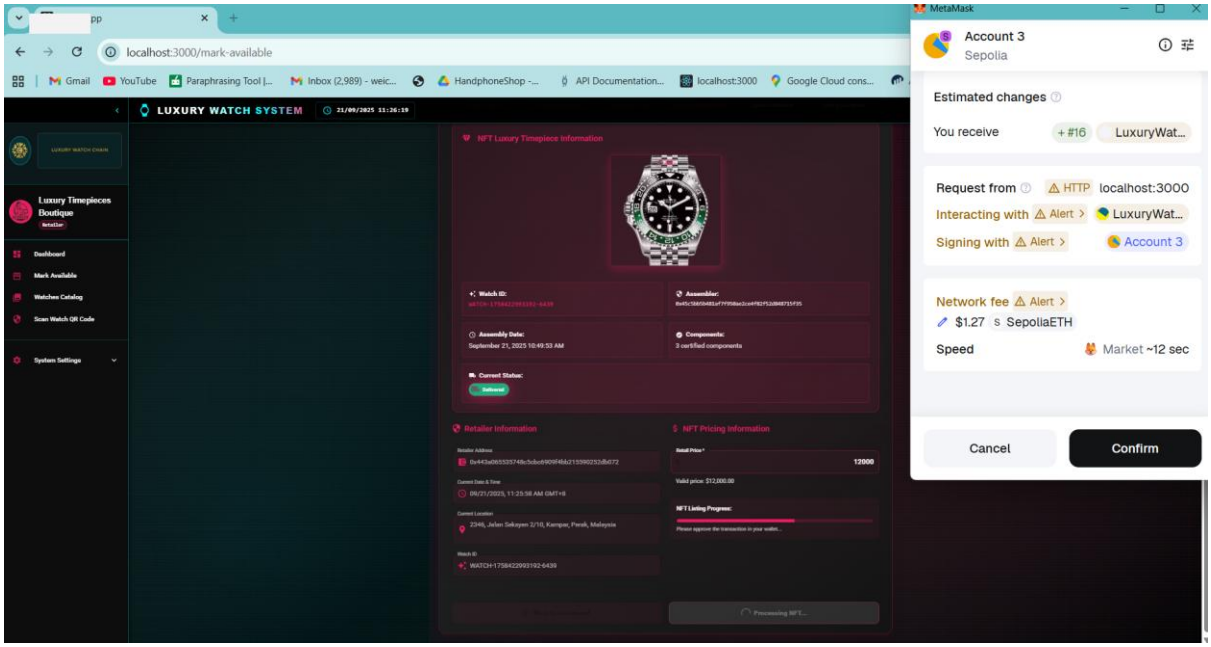

Figure 5.5.27 Confirm MetaMask Transaction in Retailer Dashboard

Step 4: A successful message will appear once it is done.

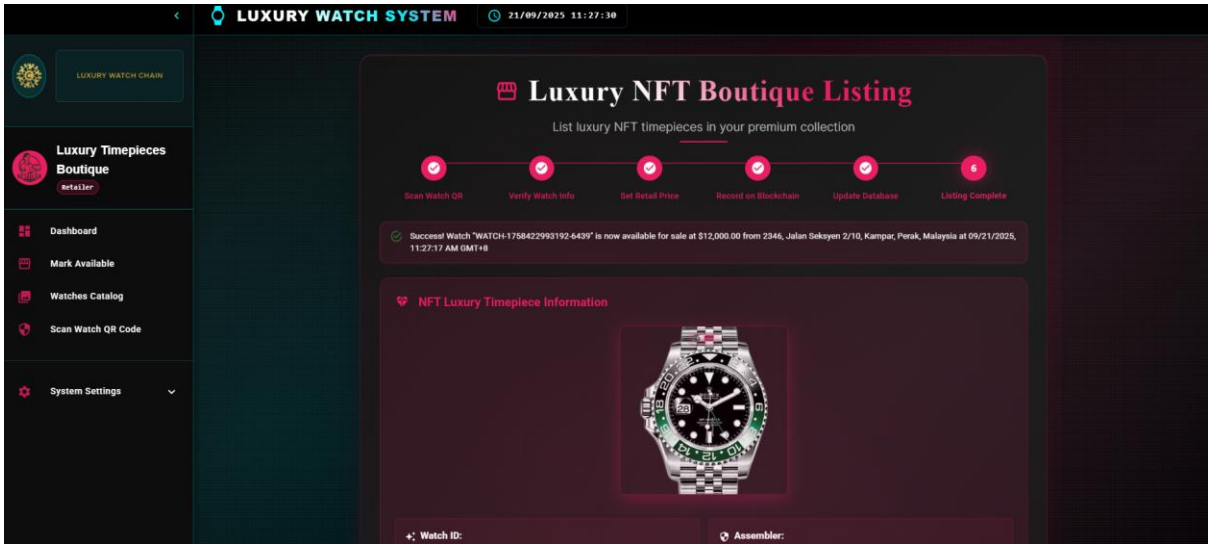

Figure 5.5.28 Successful Message in Retailer Dashboard

Step 5: Click "Watches Catalog" to view the watches.

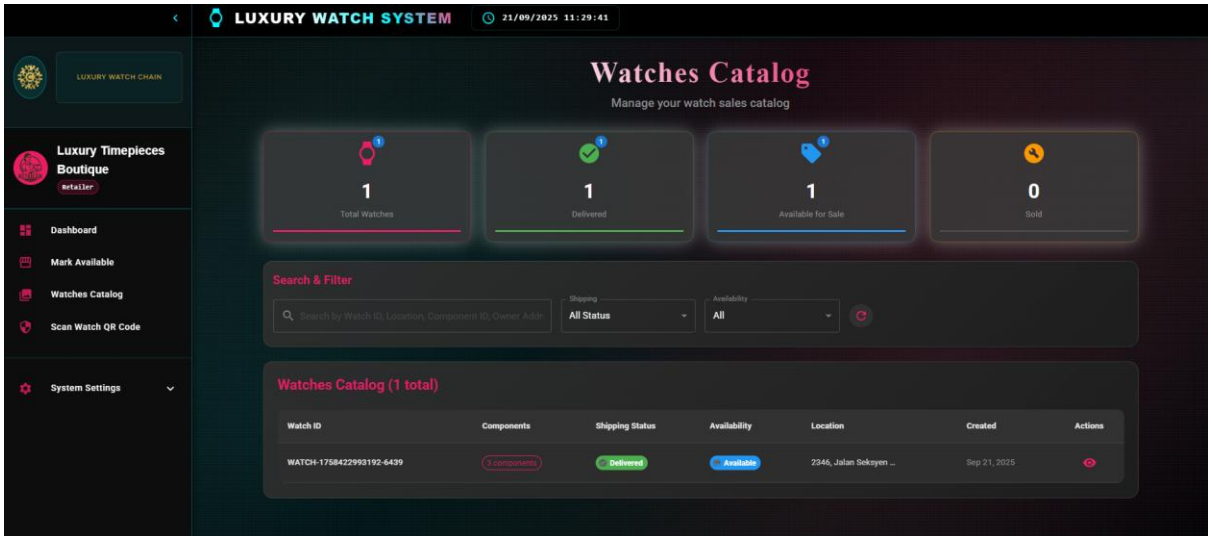

Figure 5.5.29 Watches Catalog

7. Consumer Access Interface Implementation

Consumers require intuitive interfaces for authentication, ownership verification, and provenance exploration. The interface should provide detailed traceability visualization showing complete supply chain history from raw materials through retail sale. NFT ownership verification should be easily accessible through QR code scanning or direct blockchain queries. Resale marketplace integration enables secure peer-to-peer transactions with automatic ownership transfer capabilities.

Step 1: Log in to the Consumer Dashboard

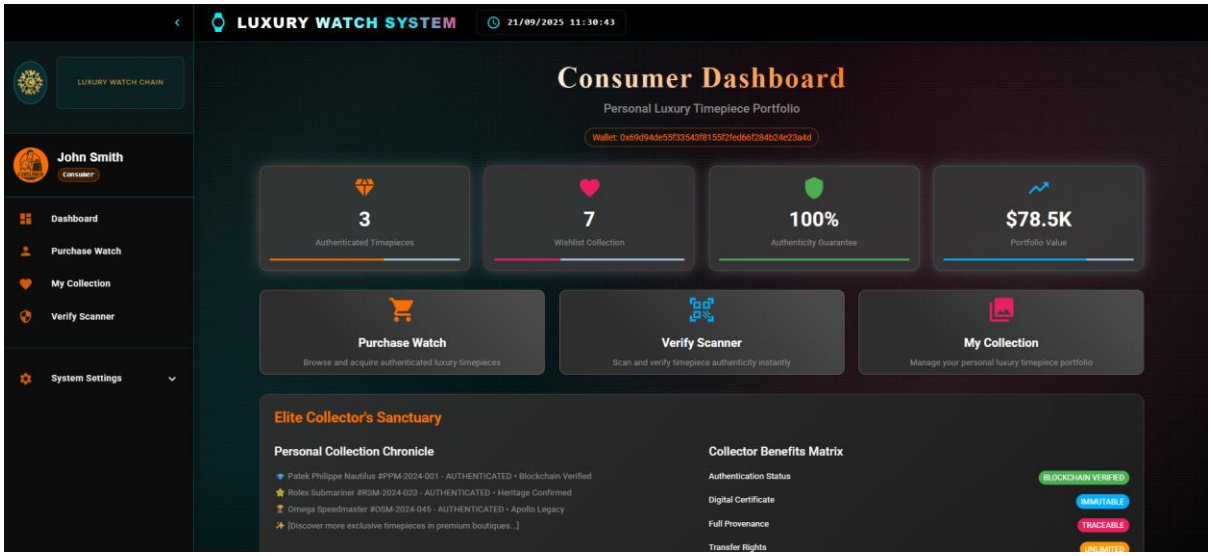

Figure 5.5.30 Consumer Dashboard

Step 2: Scan the watch QR Code and click the "Acquire NFT Ownership" button.

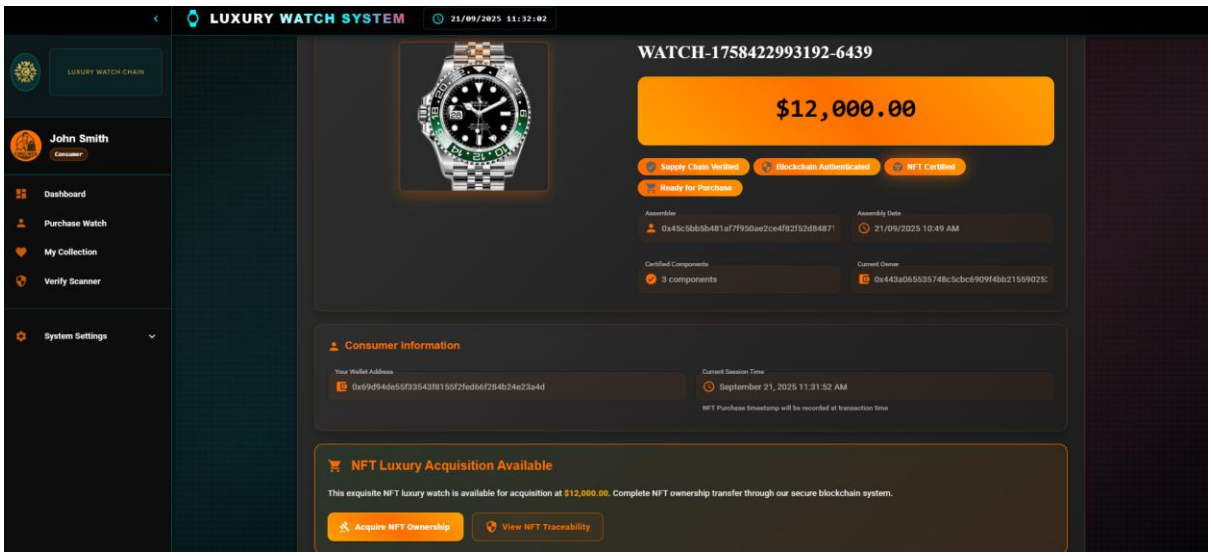

Figure 5.5.31 Purchase Watch in Consumer Dashboard

Step 3: Click the “Confirm” button in the MetaMask extension to allow the transaction.

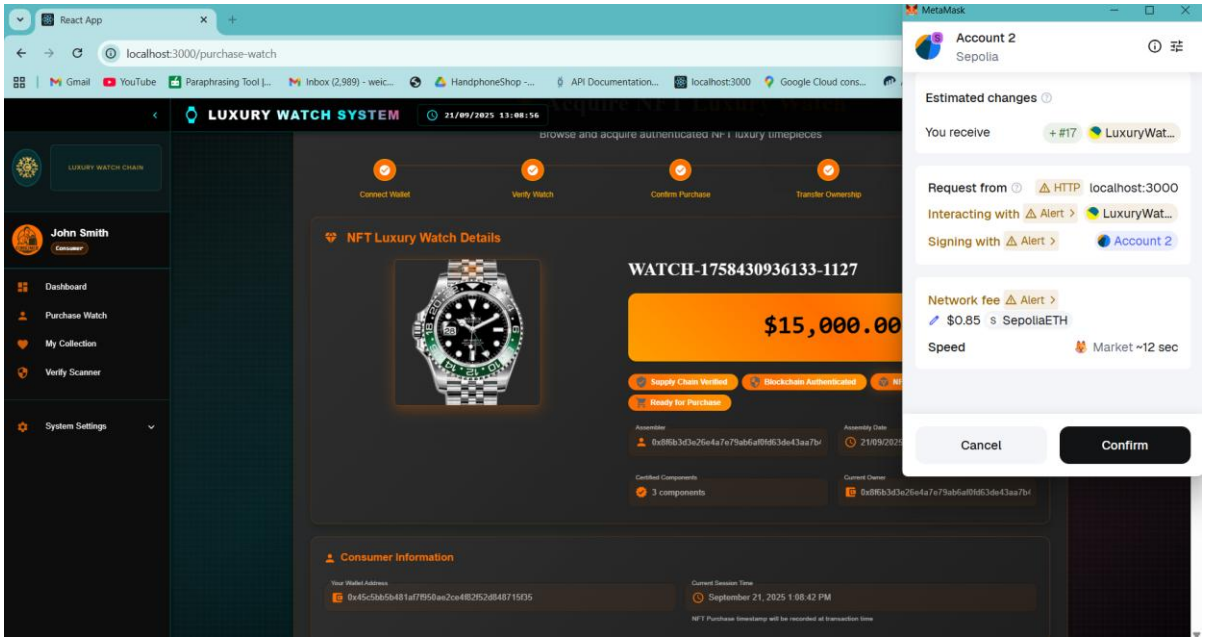

Figure 5.5.32 Confirm MetaMask Transaction in Consumer Dashboard

Step 4: An alert message will appear once the NFT ownership transfer is successful.

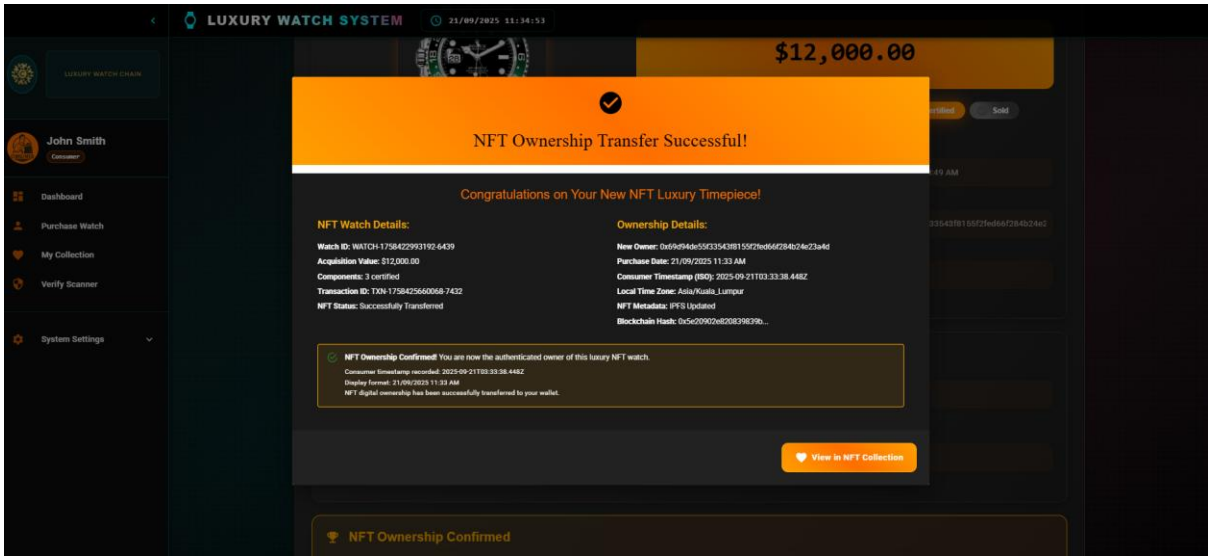

Figure 5.5.33 Successful Ownership Transfer Alert Message

Step 5: Click the “My Collection” button to view owned watches.

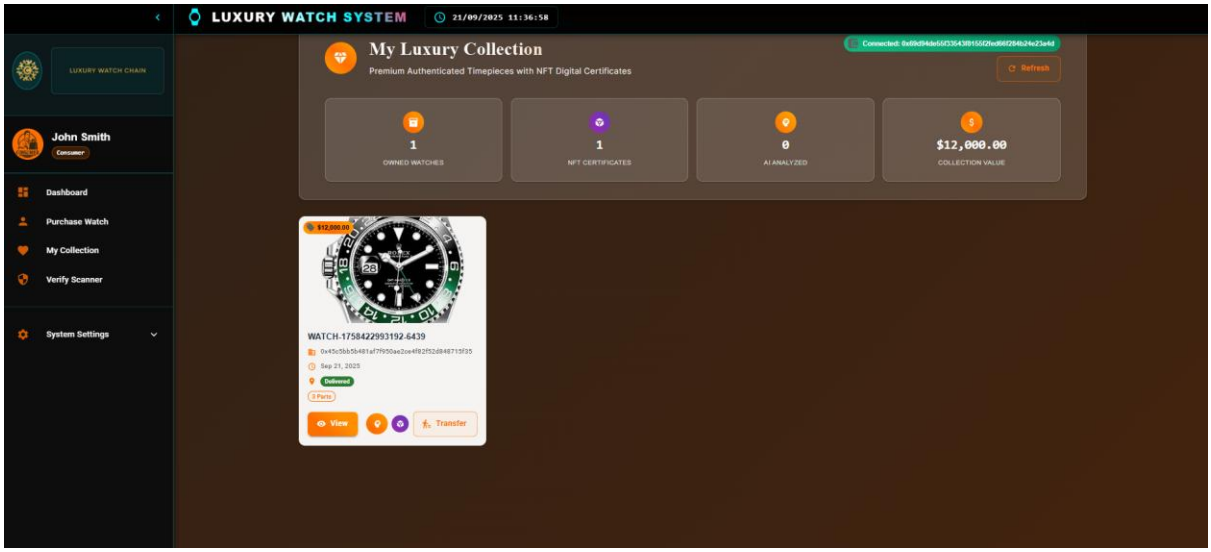

Figure 5.5.34 My Collection Watches in Consumer Dashboard

Step 5: Click the "View" button to view the watch history

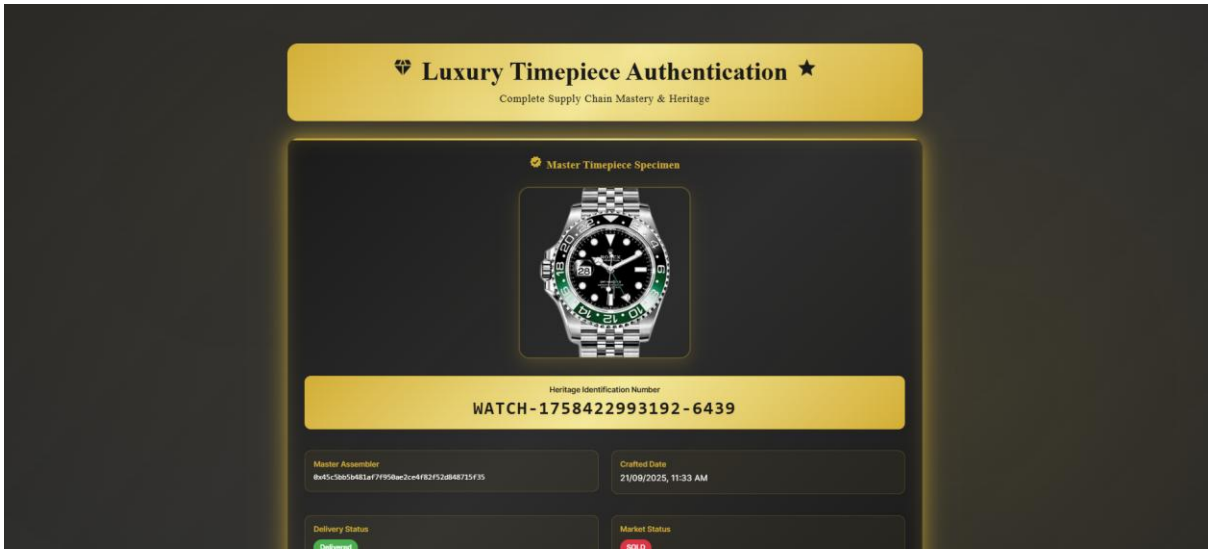

Figure 5.5.35 View Watch 1

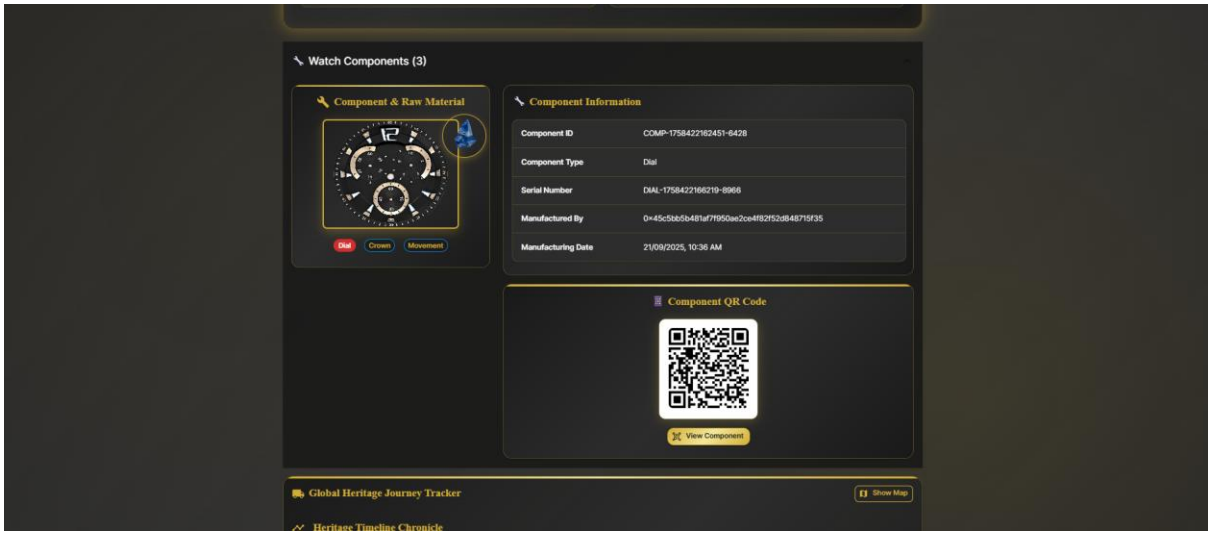

Figure 5.5.36 View Watch 2

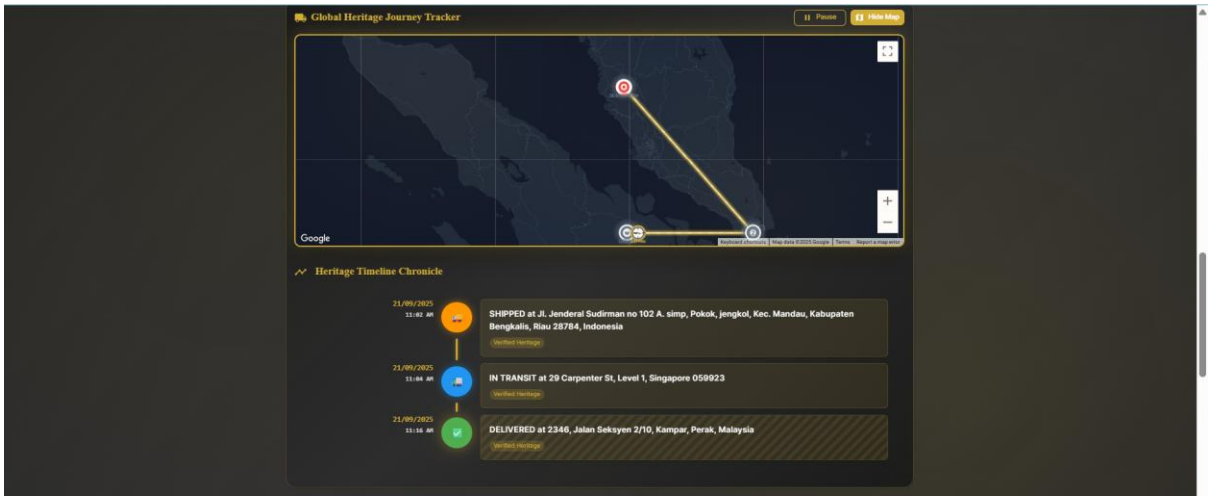

Figure 5.5.37 View Watch 3

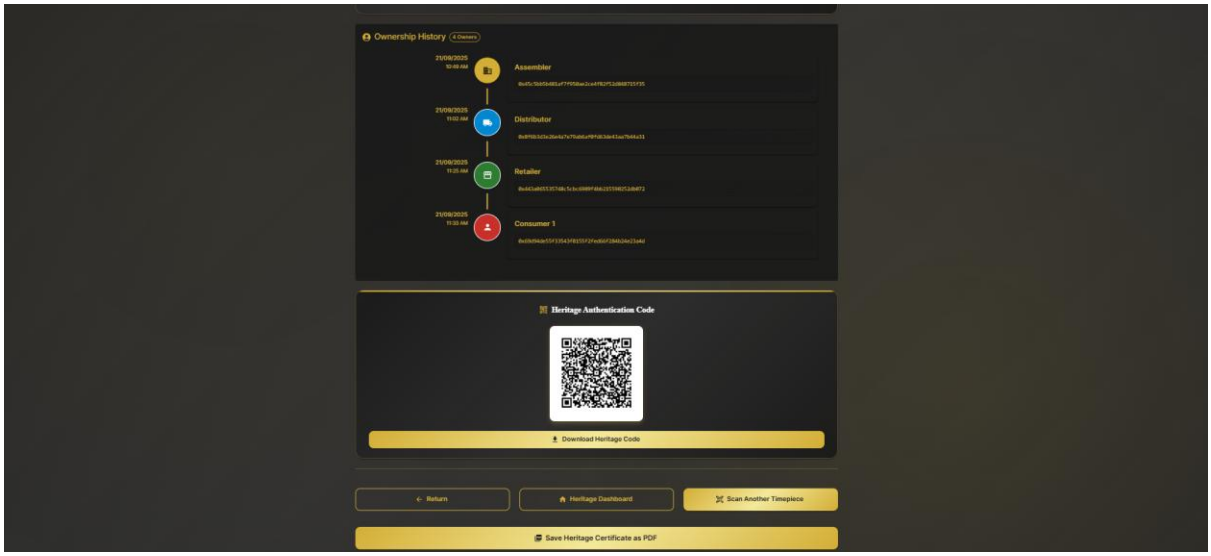

Figure 5.5.38 View Watch 4

Step 6: Click the purple colour button (View NFT) to view the NFT digital certificate

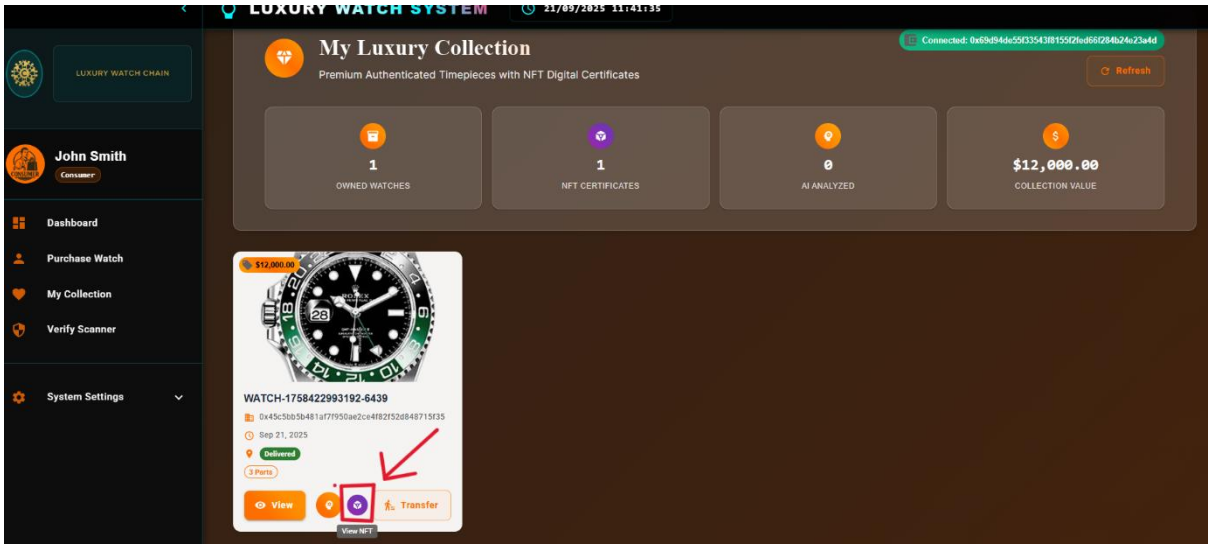

Figure 5.5.39 View NFT Digital Certificate Button

Step 7: This is an NFT Digital Certificate

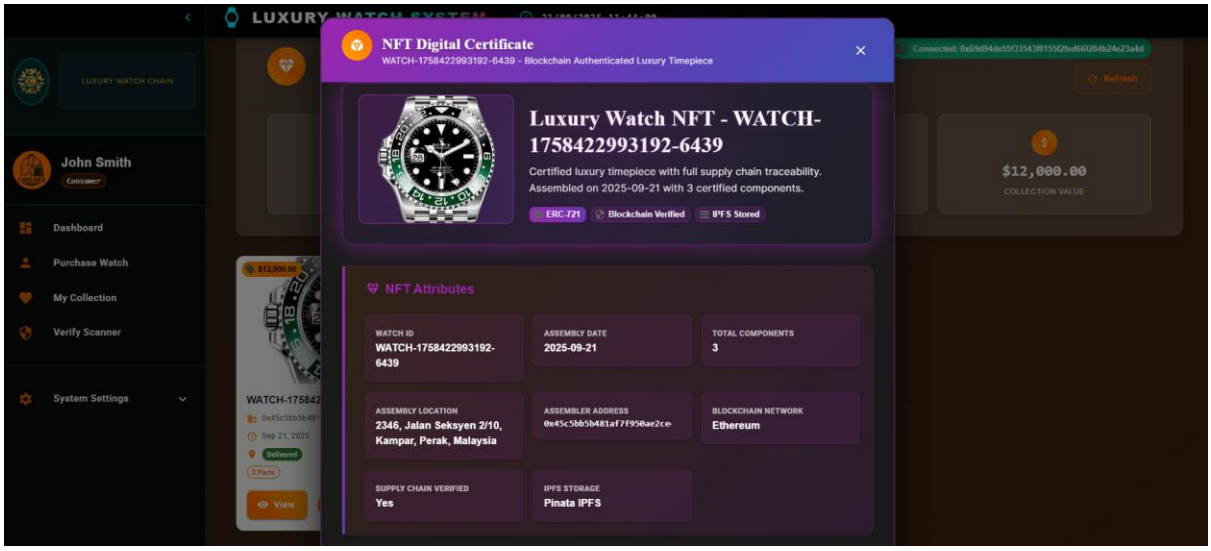

Figure 5.5.40 NFT Digital Certificate

Step 8: Click the analyze button and then click the “Start Enhanced AI Analysis”

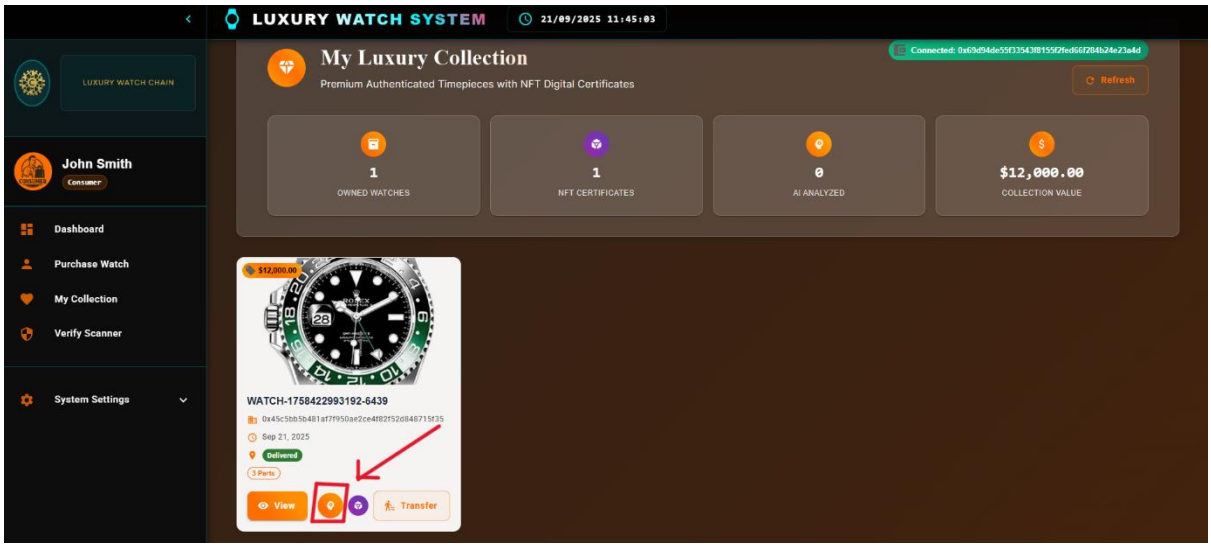

Figure 5.5.41 AI Model Analyze Button

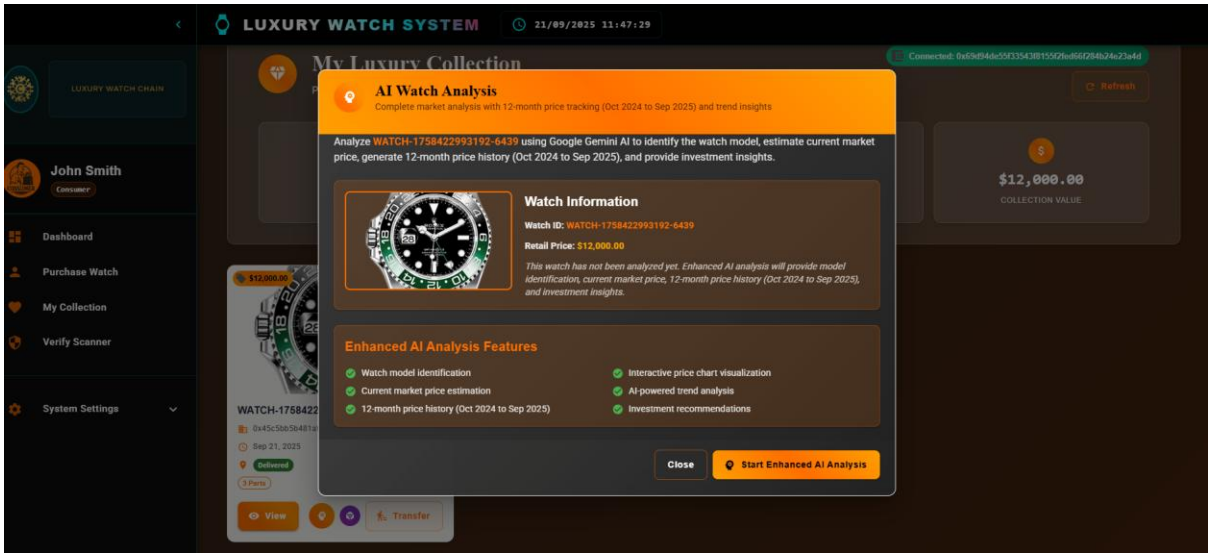

Figure 5.5.42 AI Watch Analysis

Step 9: The system uses the AI model (Gemini 2.5) to identify the watch model, make an analysis, and investment recommendations.

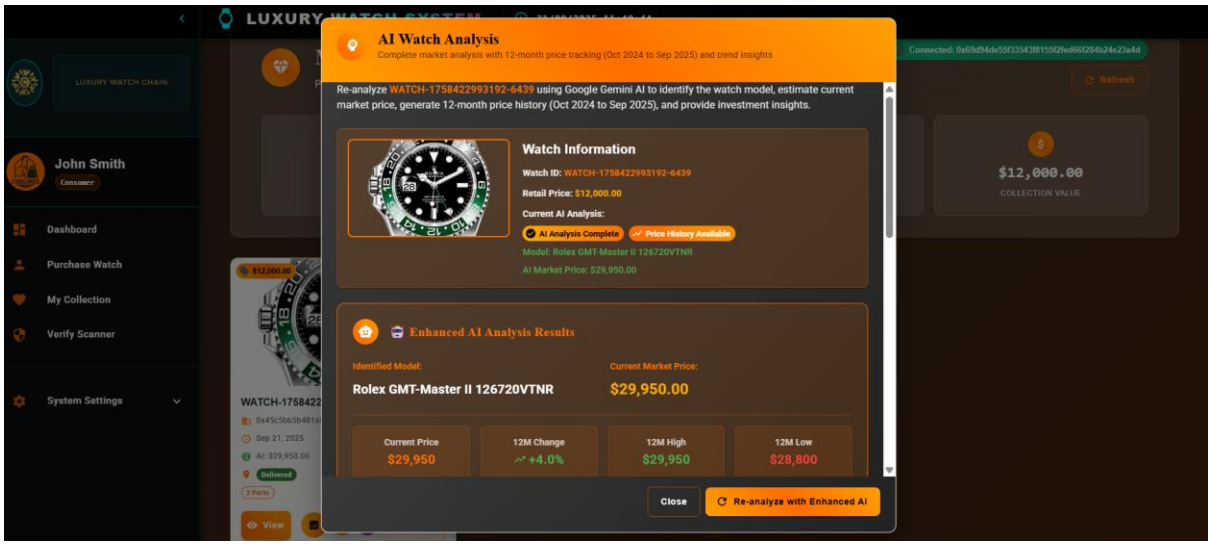

Figure 5.5.43 AI Watch Analysis Result 1

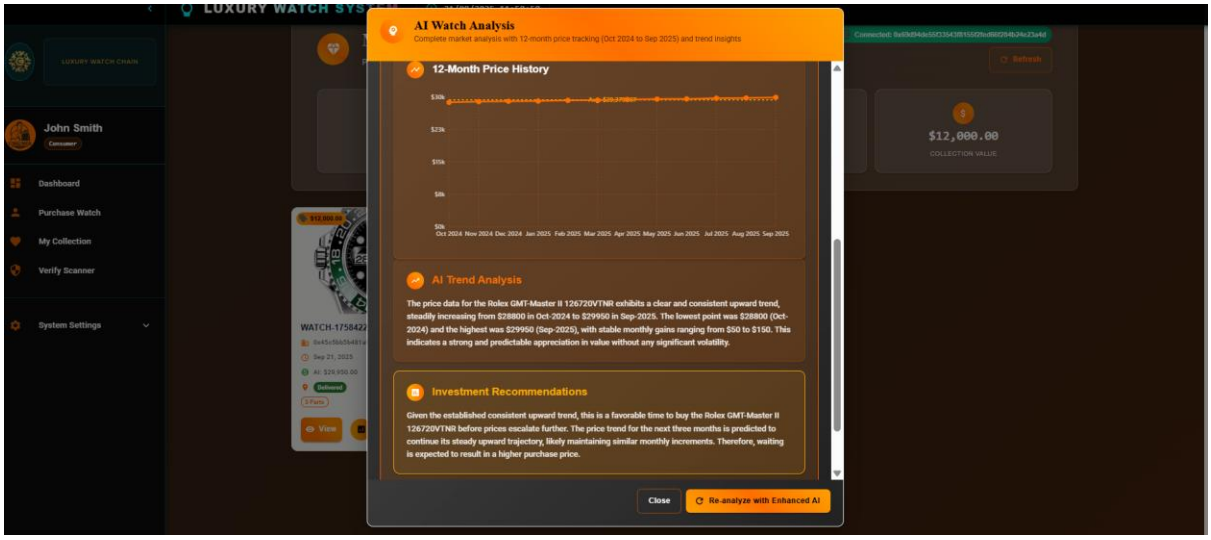

Figure 5.5.44 AI Watch Analysis Result 2

Step 11: Click the "Transfer" button to make the ownership transfer of the watch

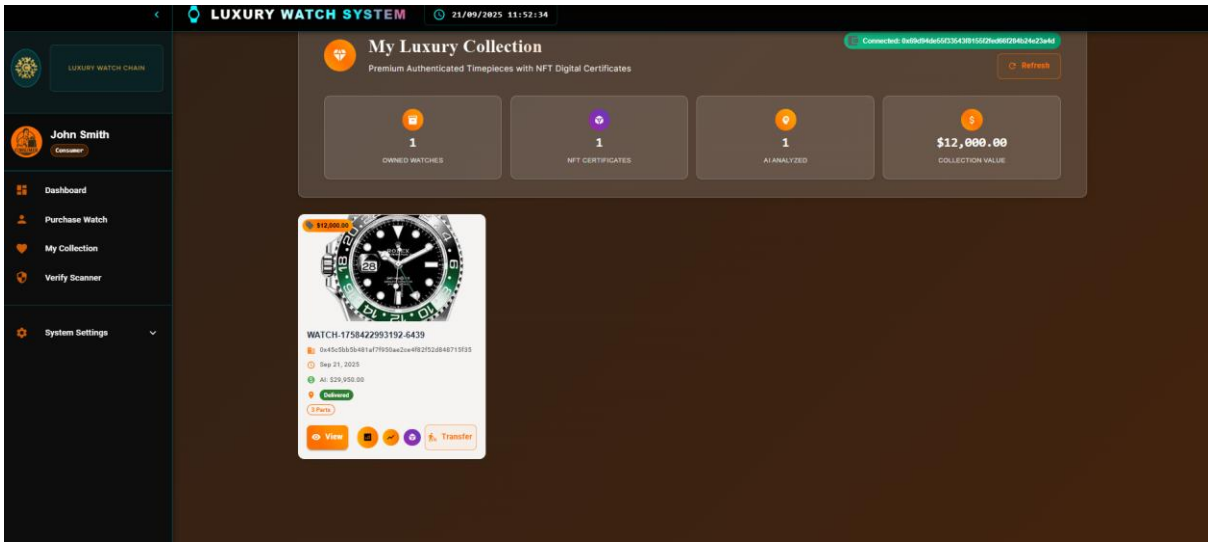

Figure 5.5.45 Ownership Transfer

Step 12: Enter the new owner with the valid wallet address. The new owner of this watch is "0x8f6b3d3e26e4a7e79ab6af0fd63de43aa7b44a31" and then click the "Transfer NFT" button to make the transactions.

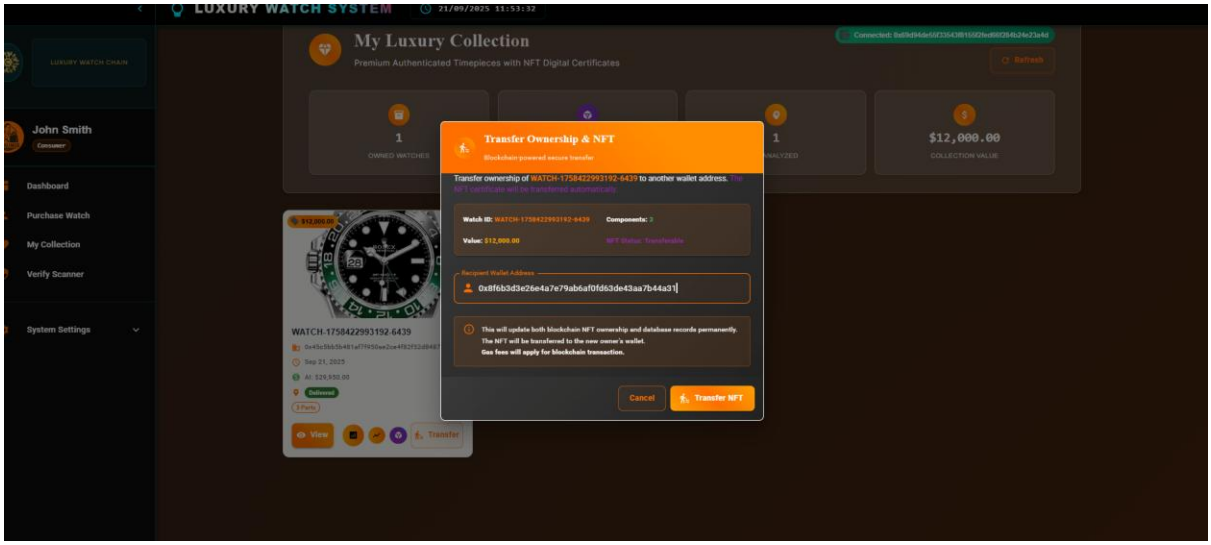

Figure 5.5.46 Enter Valid Wallet Address

Step 13: Click the "Confirm" button in the MetaMask extension to allow the transaction.

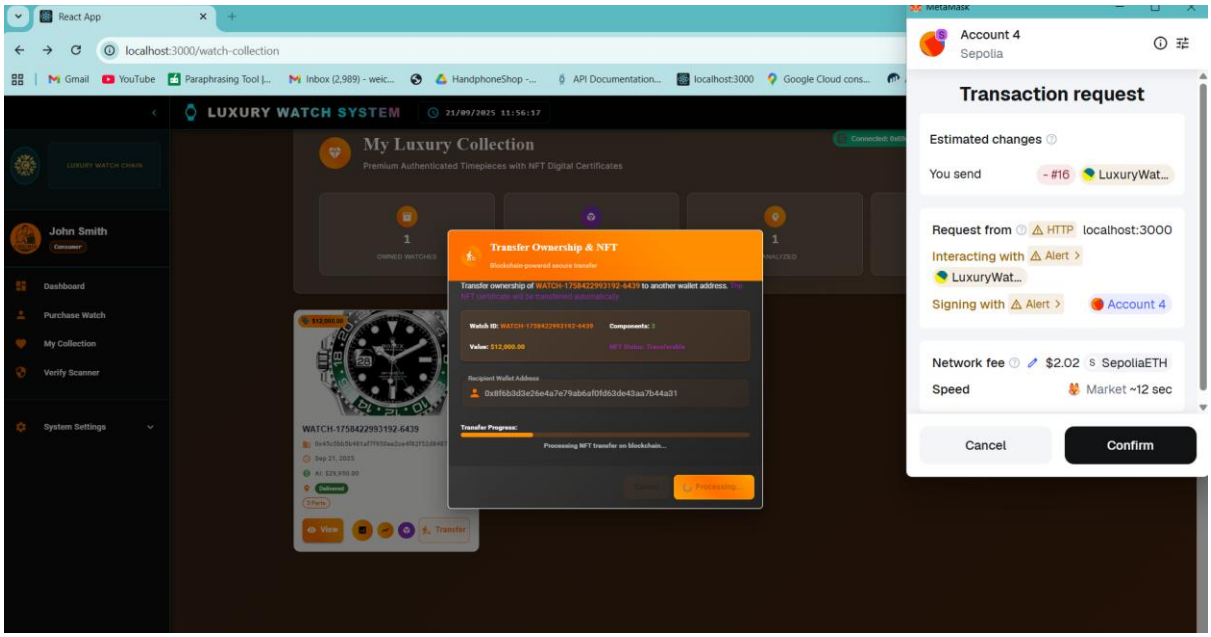

Figure 5.5.47 Ownership Transfer Transaction through MetaMask

Step 14: Change the wallet address to "0x8f6b3d3e26e4a7e79ab6af0fd63de43aa7b44a31" and then the transferred watch will be shown.

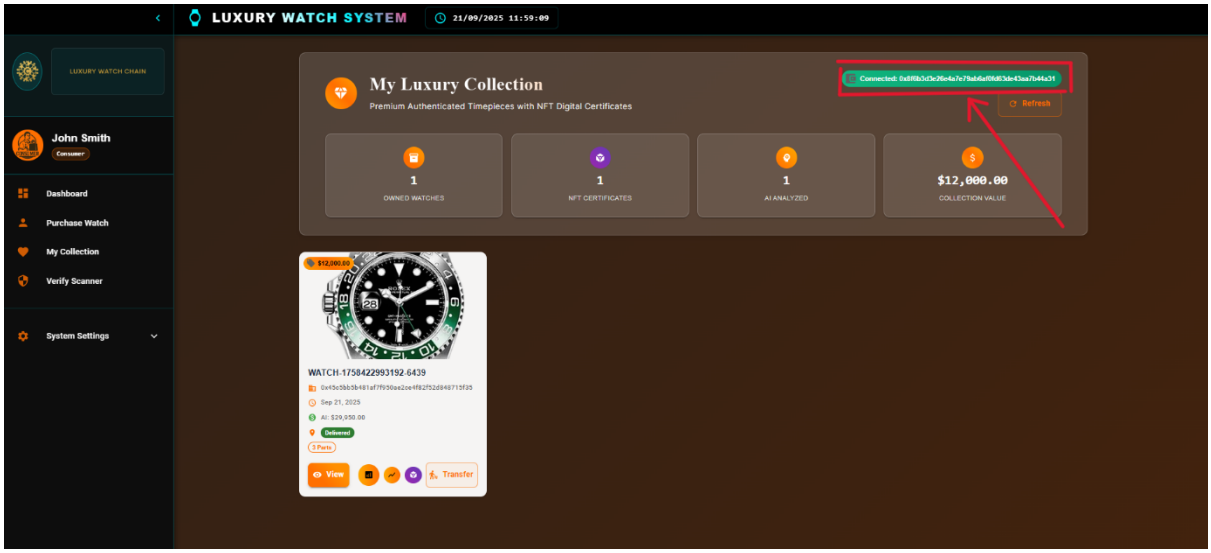

Figure 5.5.48 New Owner of NFT Watch

Step 15: Verify blockchain transactions in Etherscan through

https://sepolia.etherscan.io/address/0x8f6b3d3e26e4a7e79ab6af0fd63de43aa7b44a31

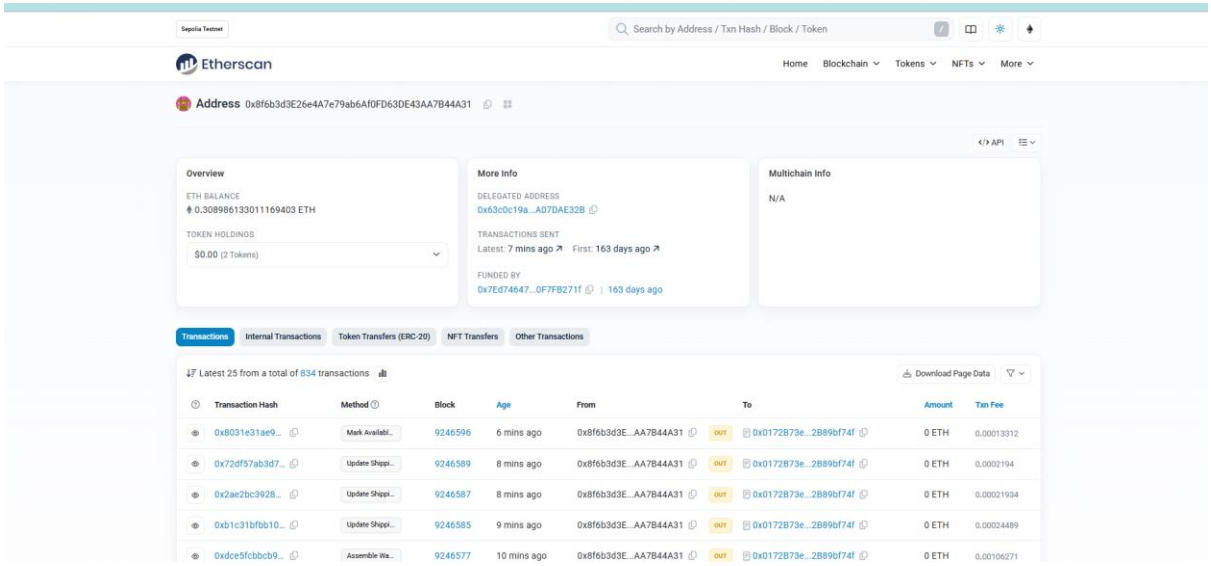

Figure 5.5.49 View Transactions on Etherscan

Step 16: Verify NFT Watch Ownership in Etherscan through

https://sepolia.etherscan.io/nft/0x0172b73e1c608cf50a8adc16c9182782b89bf74f/16

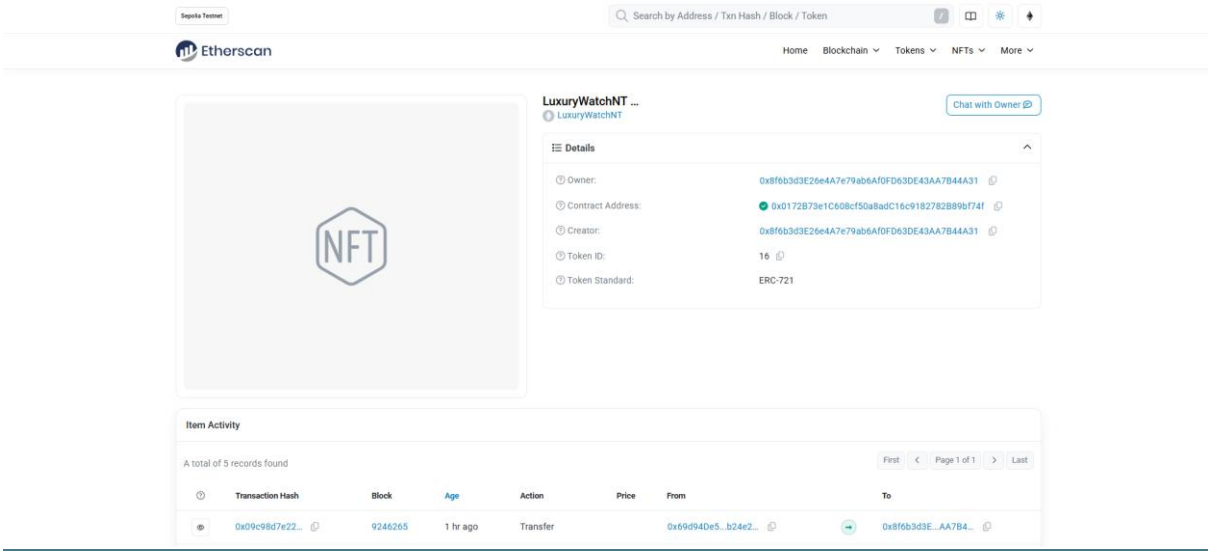

Figure 5.5.50 NFT Watch Ownership in Etherscan

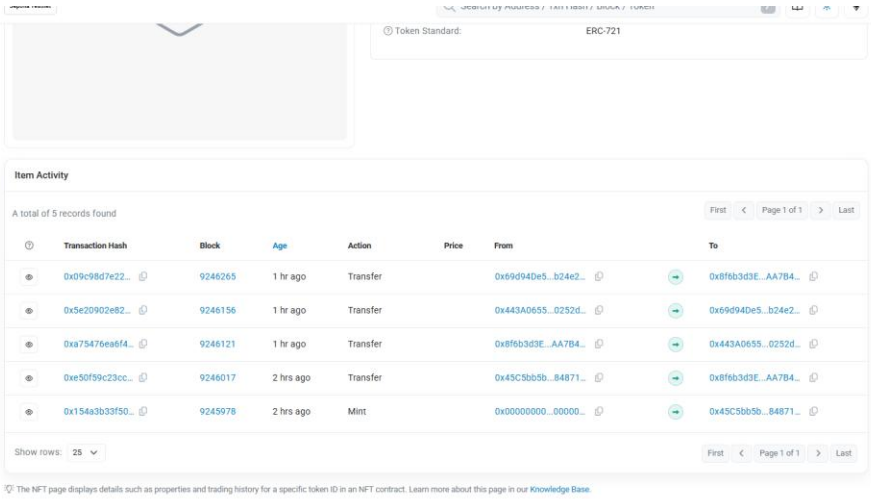

Figure 5.5.51 NFT Watch Ownership in Etherscan 2

5.6 Implementation Issues and Challenges

1. Blockchain Integration Complexities

The most significant challenge involved coordinating smart contract operations with off-chain database updates. Maintaining consistency between NFT ownership on the blockchain and current_owner fields in the database created complex synchronization issues. When blockchain transactions failed due to gas issues or network problems while database updates succeeded, the system required sophisticated rollback mechanisms. Additionally, MetaMask wallet integration introduced user experience challenges with transaction confirmations, network switching, and handling rejected transactions that demanded extensive error handling and state management.

2. Multi-Stakeholder Workflow Coordination

Managing seven different stakeholder roles with sequential process validation presented the most complex business logic challenges. Ensuring that components could only be assembled after certification, preventing double-spending of raw materials, and coordinating concurrent operations by different stakeholders required extensive validation logic. The challenge of maintaining proper supply chain sequence progression while supporting simultaneous operations by multiple users created race conditions that demanded sophisticated concurrency control and conflict resolution mechanisms.

3. IPFS Storage Reliability Issues

Dependency on external IPFS services through Pinata introduced critical reliability challenges that affected core NFT functionality. Network connectivity issues and service outages required implementing comprehensive fallback mechanisms with local metadata storage. The immutable nature of IPFS content conflicted with requirements for metadata updates during supply chain progression, necessitating complex versioning strategies and hash management systems while maintaining NFT standard compliance and marketplace compatibility.

5.7 Concluding Remarks

This comprehensive luxury watch supply chain management system represents a sophisticated convergence of traditional craftsmanship authentication with cutting-edge blockchain technology. The proposed architecture demonstrates exceptional technical depth through its hybrid smart contract approach, which maintains operational efficiency while ensuring immutable provenance tracking from raw material sourcing through consumer ownership.

Besides, the integration of NFT tokenization with IPFS metadata storage creates a robust framework for luxury goods authentication that addresses critical industry challenges including counterfeiting, supply chain opacity, and ownership verification. The role-based dashboard implementation strategy provides stakeholder-specific interfaces that optimize workflow efficiency while maintaining comprehensive traceability requirements throughout the entire product lifecycle.

Furthermore, the technical implementation showcases advanced understanding of distributed systems architecture, combining PostgreSQL database management, Express.js backend services, React frontend frameworks, and Ethereum smart contract development into a cohesive enterprise solution. The progressive development methodology and comprehensive testing strategy demonstrate professional software development practices suitable for production deployment.

This system establishes a foundational framework that could transform luxury goods authentication practices across multiple industries beyond horology. The combination of blockchain immutability, decentralized storage, and role-based access controls creates a scalable solution that addresses both current market needs and future regulatory requirements for supply chain transparency.

Last but not least, the project represents significant academic contribution to the intersection of blockchain technology and supply chain management, demonstrating practical application of theoretical concepts in a real-world context. The comprehensive documentation

and systematic approach provide valuable reference material for future research in decentralized luxury goods authentication systems.

Chapter 6 :System Evaluation And Discussion

6.1 System Testing and Performance Metrics

This section presents a comprehensive evaluation of the luxury watch supply chain management system through systematic testing methodologies and quantitative performance analysis. The testing framework encompasses smart contract functionality validation, backend API performance assessment, and frontend component integration verification across all stakeholder roles within the supply chain ecosystem.

The system validation employs a hierarchical testing approach consisting of four distinct layers: smart contract testing at the blockchain layer, backend API testing for business logic validation, frontend component testing for user interface functionality, and end-to-end integration testing for complete workflow verification. This methodology ensures comprehensive coverage of all system components and their interdependencies.

6.2 Testing Set Up and Result

Smart Contract Layer

```
C:\Users\USER\Desktop\Testing\final\FYP\contract>npm install --save-dev @nomicfoundation/hardhat-toolbox

changed 1 package, and audited 705 packages in 51s

134 packages are looking for funding
  run `npm fund` for details

44 vulnerabilities (16 low, 8 moderate, 8 high, 12 critical)

To address issues that do not require attention, run:
  npm audit fix

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.

C:\Users\USER\Desktop\Testing\final\FYP\contract>npm install --save-dev chai mocha hardhat-gas-reporter
npm warn deprecated glob@8.1.0: Glob versions prior to v9 are no longer supported

added 42 packages, removed 5 packages, changed 20 packages, and audited 742 packages in 12s

145 packages are looking for funding
  run `npm fund` for details

41 vulnerabilities (16 low, 6 moderate, 7 high, 12 critical)
```

Figure 6.2.1 Smart Contract Test Tools

1. **Hardhat Toolbox** serves as the foundational testing environment, providing local blockchain simulation, contract compilation, and deployment automation. The framework creates isolated testing conditions that replicate Ethereum network behavior while enabling rapid test execution and debugging capabilities.

2. **Chai assertion library** implements validation mechanisms for smart contract behavior verification. The library enables precise testing of gas consumption patterns, state transitions, error conditions, and transaction outcomes through comprehensive assertion methods tailored to blockchain applications.
3. **Mocha testing framework** provides organizational structure and execution management through hierarchical test organization and asynchronous operation support. The framework accommodates blockchain-specific requirements including transaction mining delays and timeout configurations for network communications.
4. **Gas Reporter** delivers performance monitoring and cost analysis functionality, tracking transaction costs at the function level and generating optimization recommendations. The tool provides essential data for production deployment planning and operational cost assessment.

The integrated framework establishes a systematic testing approach where Hardhat Toolbox creates the blockchain environment, Mocha structures test execution, Chai validates contract behavior, and Gas Reporter monitors performance metrics. This methodology ensures comprehensive coverage of functional requirements, security compliance, and performance optimization.

Testing Implementation Results

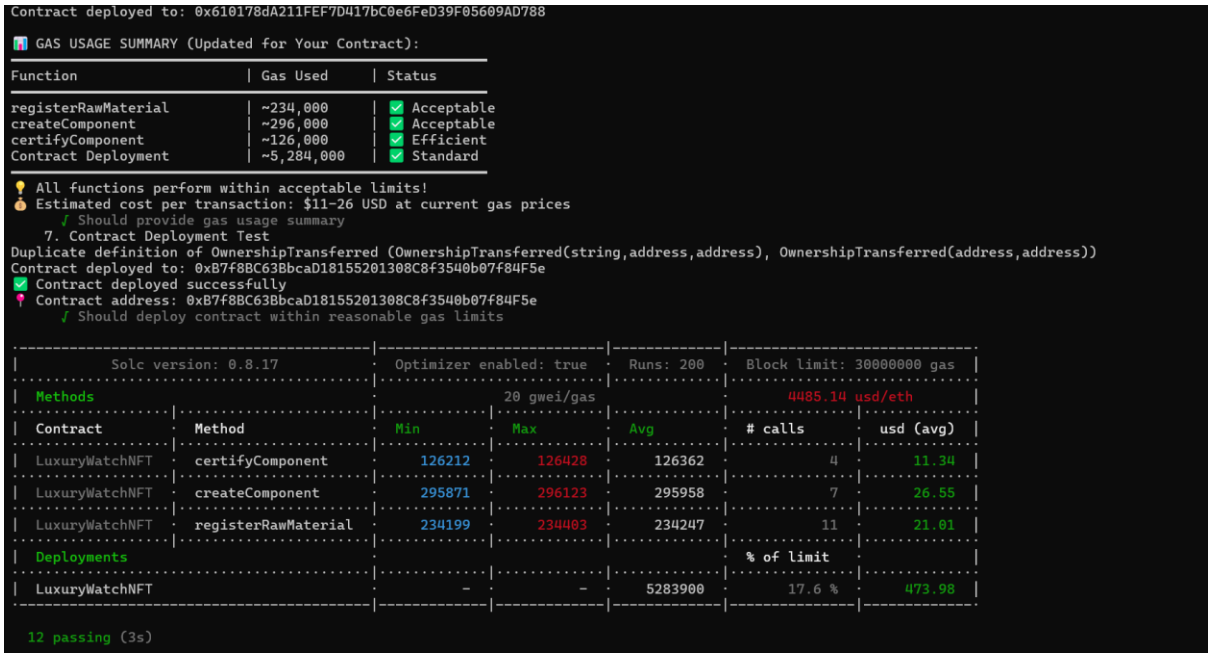

Figure 6.2.2 Blockchain Testing Performance Metrics

Backend API Endpoints - Postman (Manual/Interactive Testing)

1. Basic Health Check

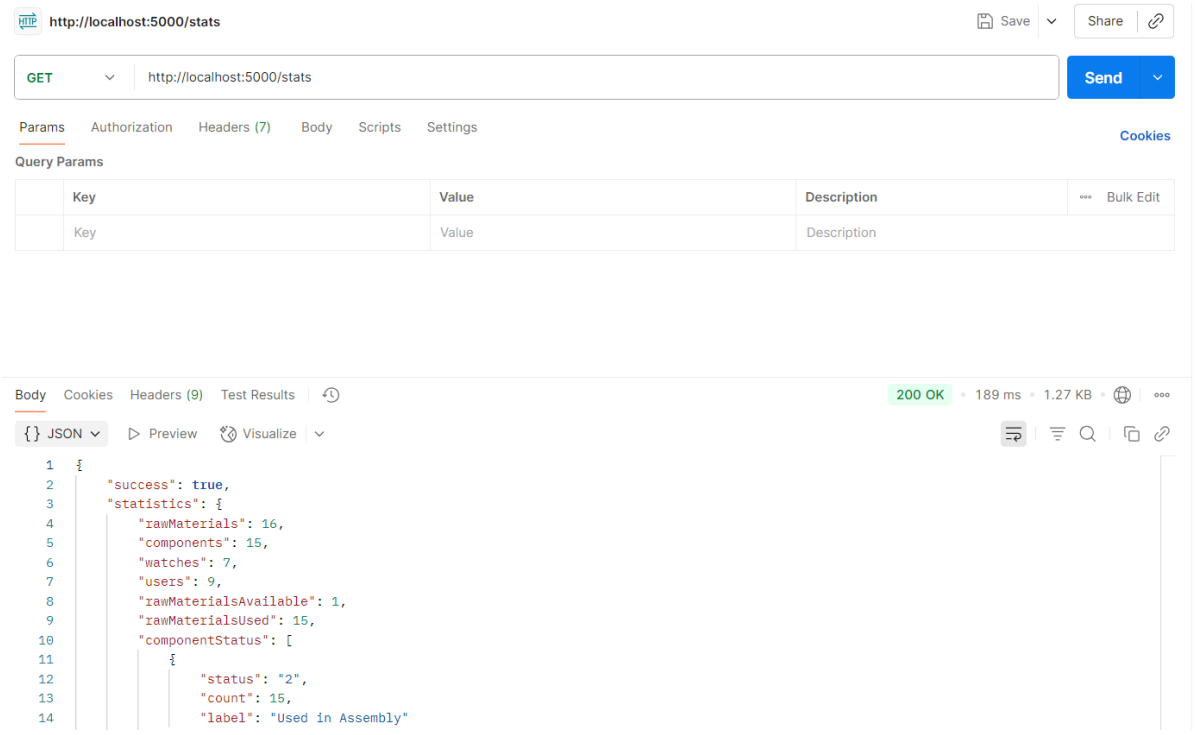

Figure 6.2.3 Postman Testing (Basic Health Check)

2. Authentication and User Management

Test User Registration

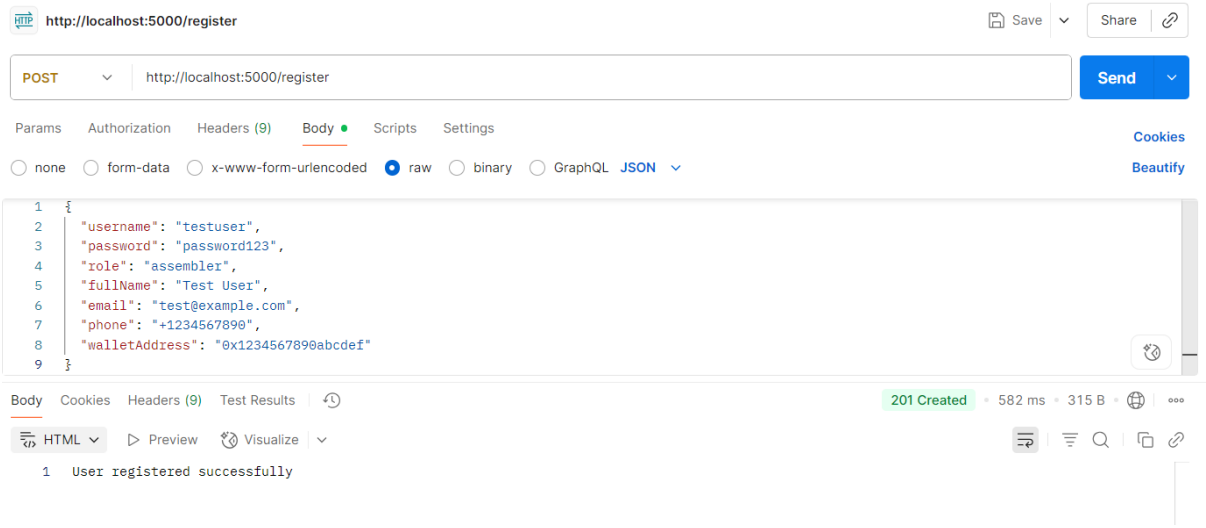

Figure 6.2.4 Postman Testing (Test User Registration)

Test User Login

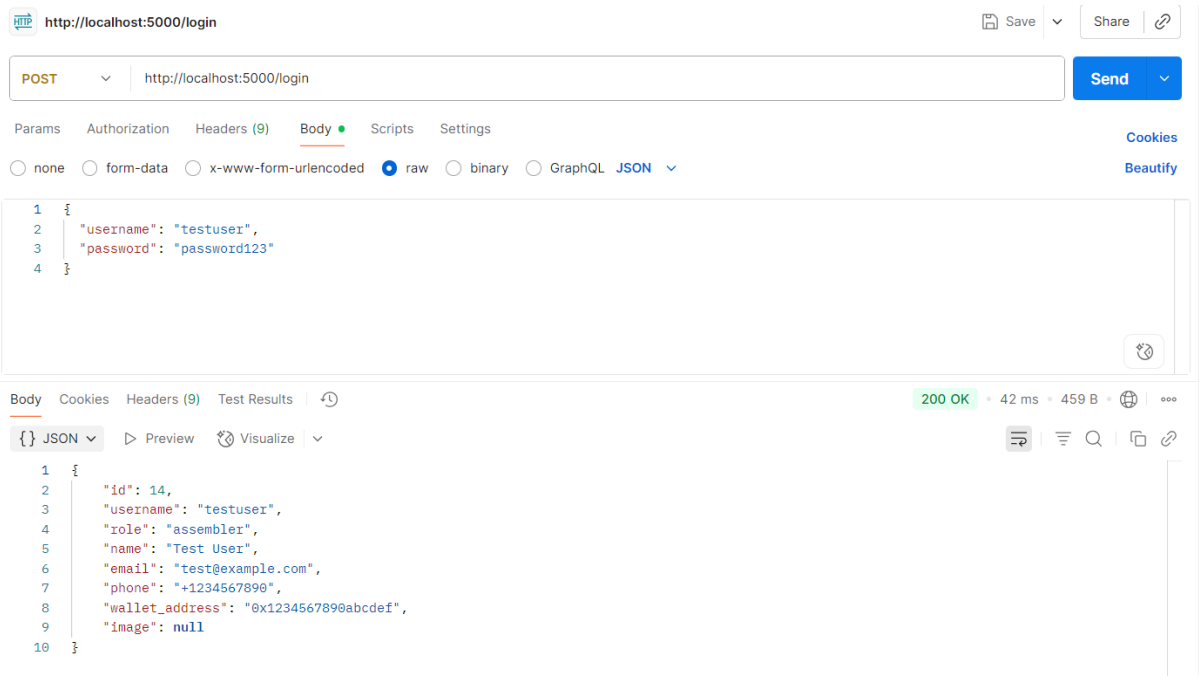

Figure 6.2.5 Postman Testing (Test User Login)

Add Account (Admin)

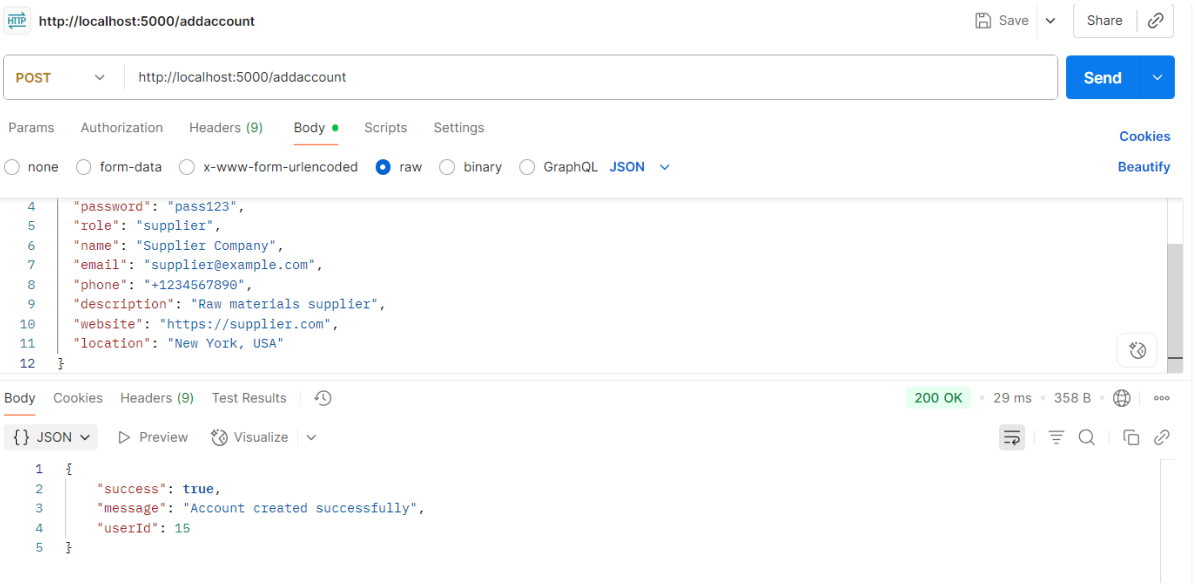

Figure 6.2.6 Postman Testing (Add Account- Admin)

Get All Profiles

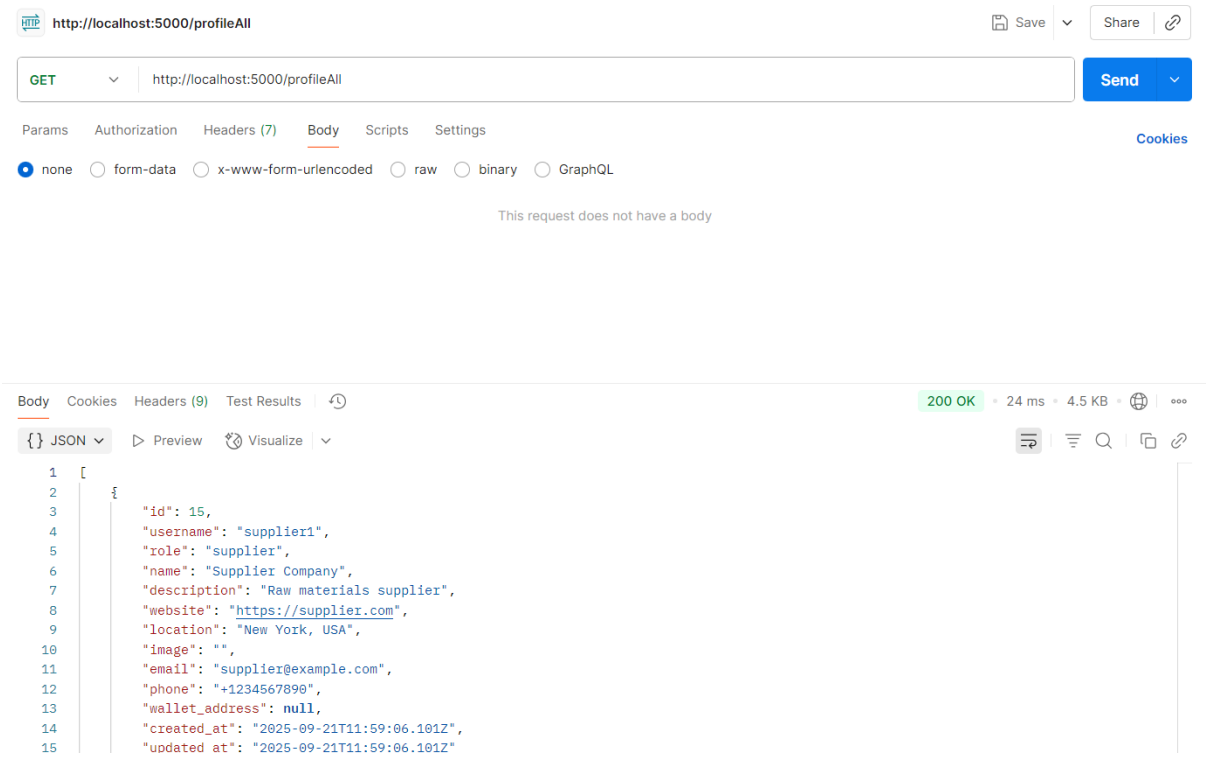

Figure 6.2.7 Postman Testing (Get All Profiles)

Get Specific Profile

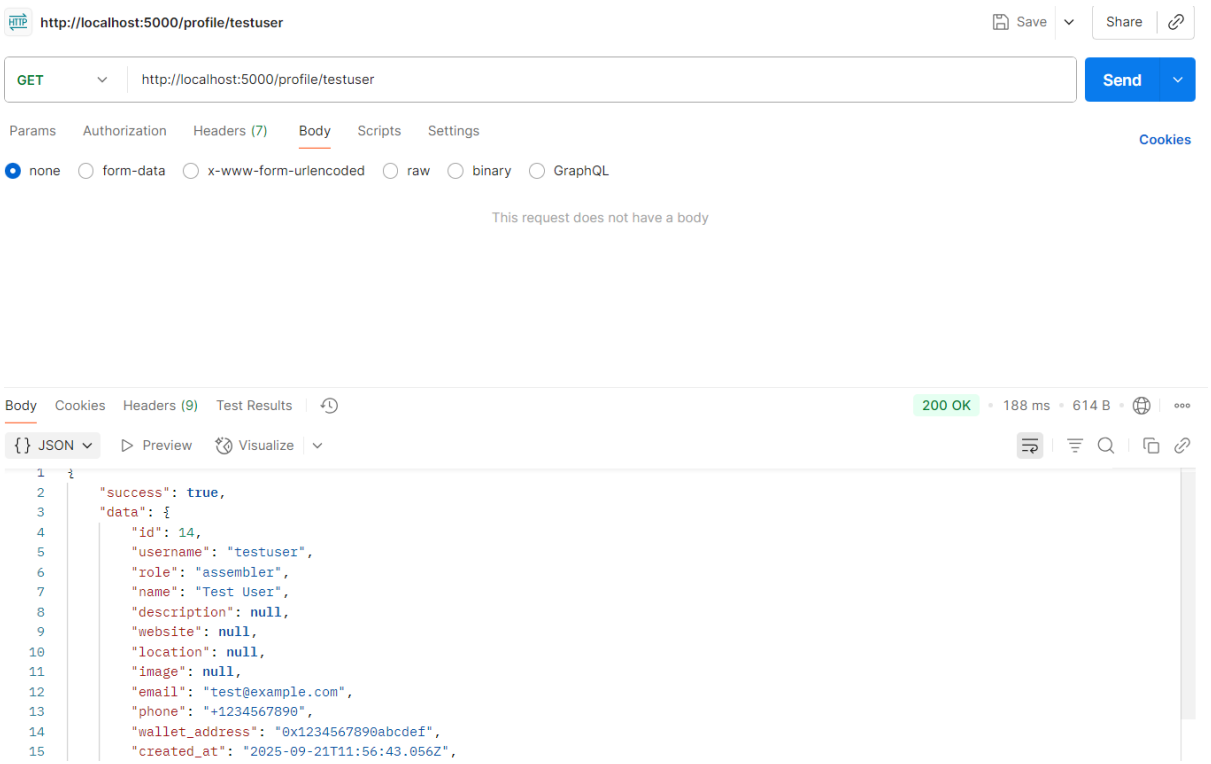

Figure 6.2.8 Postman Testing (Get Specific Profile)

3. File Upload Testing

Upload Profile Image

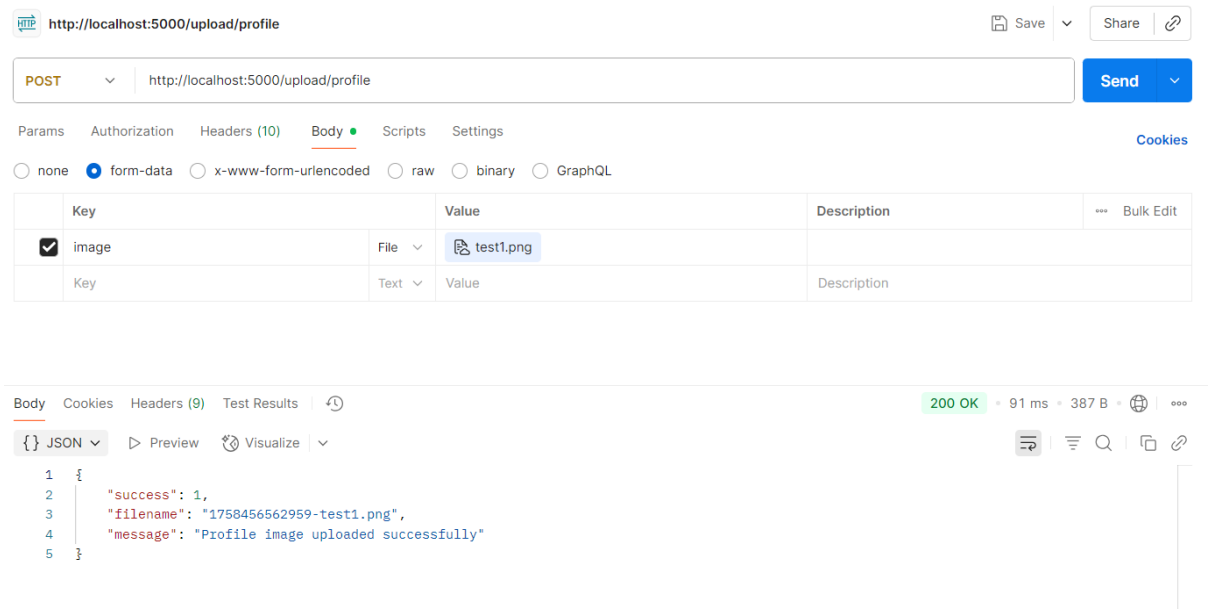

Figure 6.2.9 Postman Testing (Upload Profile Image)

Upload Raw Material Image

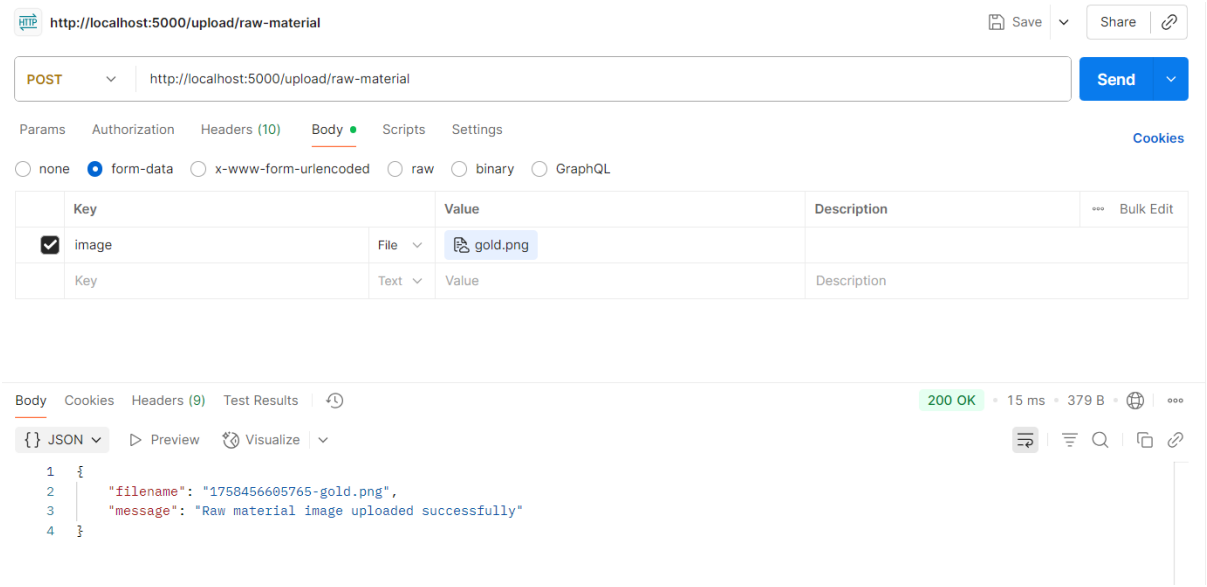

Figure 6.2.10 Postman Testing (Upload Raw Material Image)

Upload Component Image

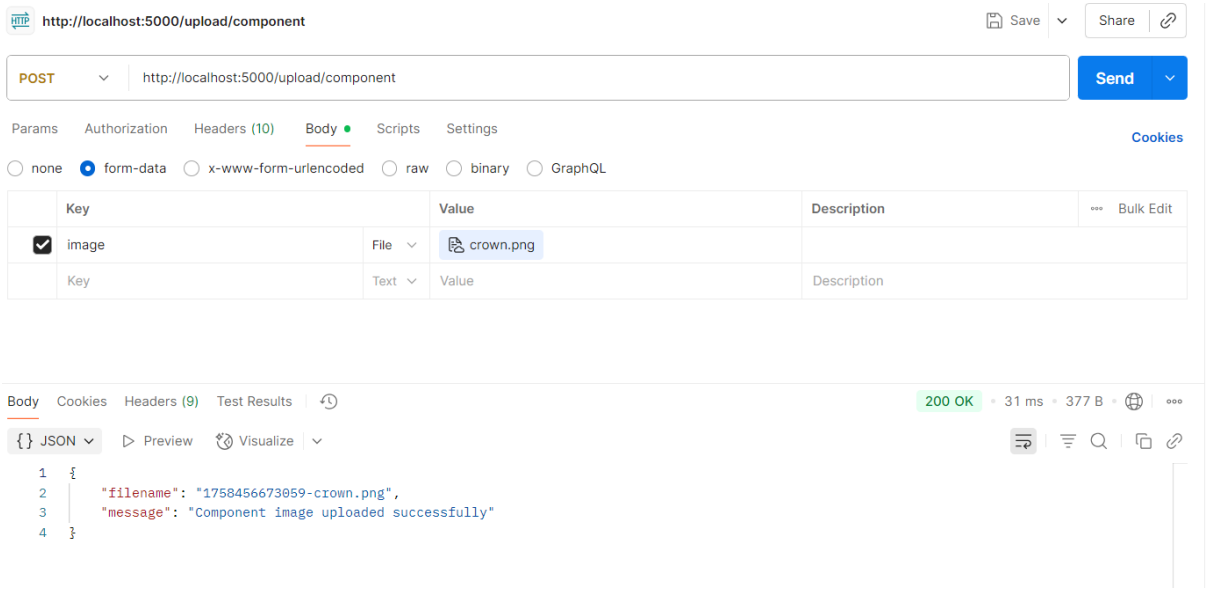

Figure 6.2.11 Postman Testing (Upload Component Image)

Upload Watch Image

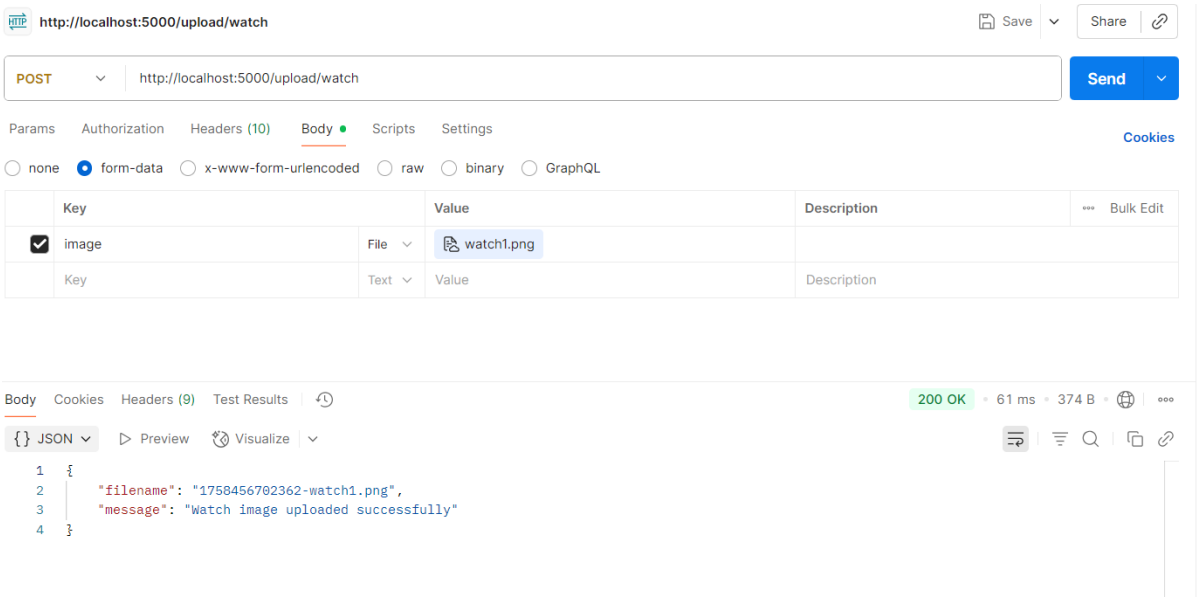

Figure 6.2.12 Postman Testing (Upload Watch Image)

4. Supplier Module

Add Raw Material

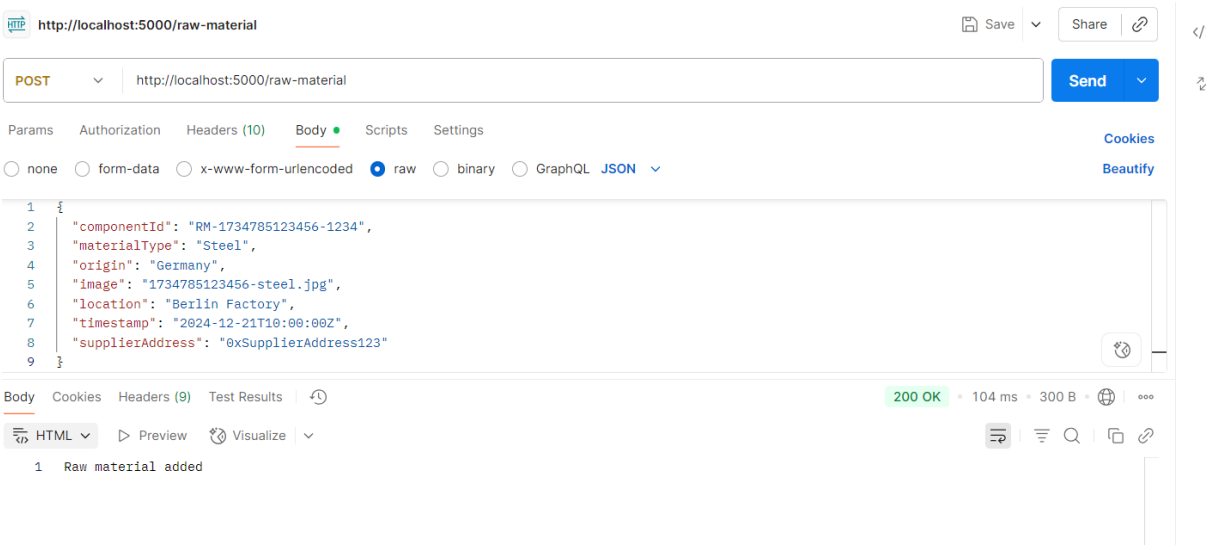

Figure 6.2.13 Postman Testing (Add Raw Material)

Get All Raw Materials

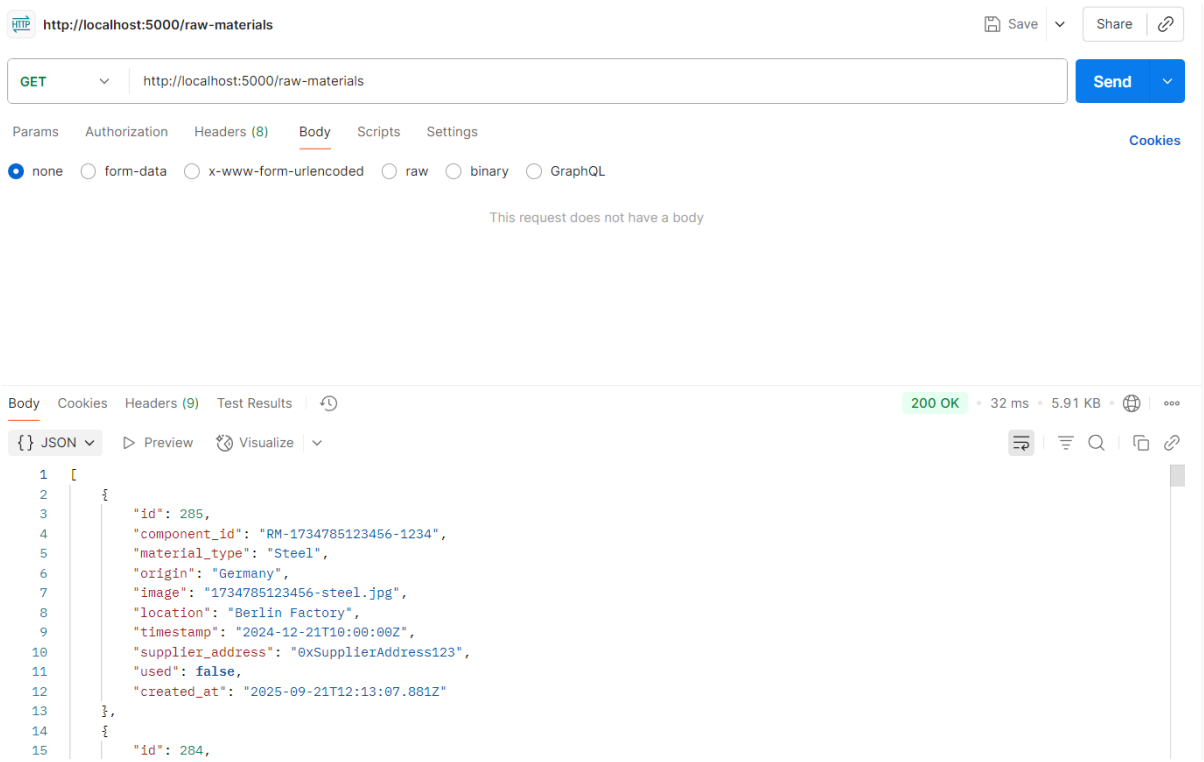

Figure 6.2.14 Postman Testing (Get All Raw Materials)

Get Specific Raw Material

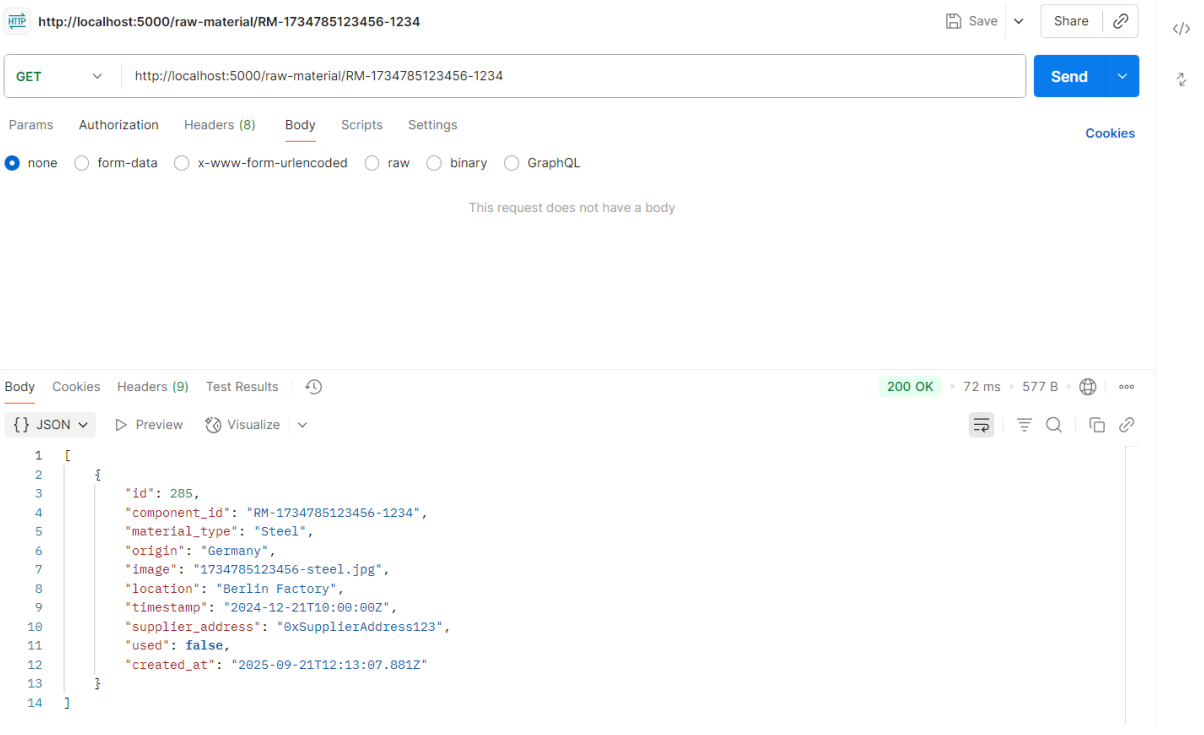

Figure 6.2.15 Postman Testing (Get Specific Raw Material)

Get Raw Materials by Supplier

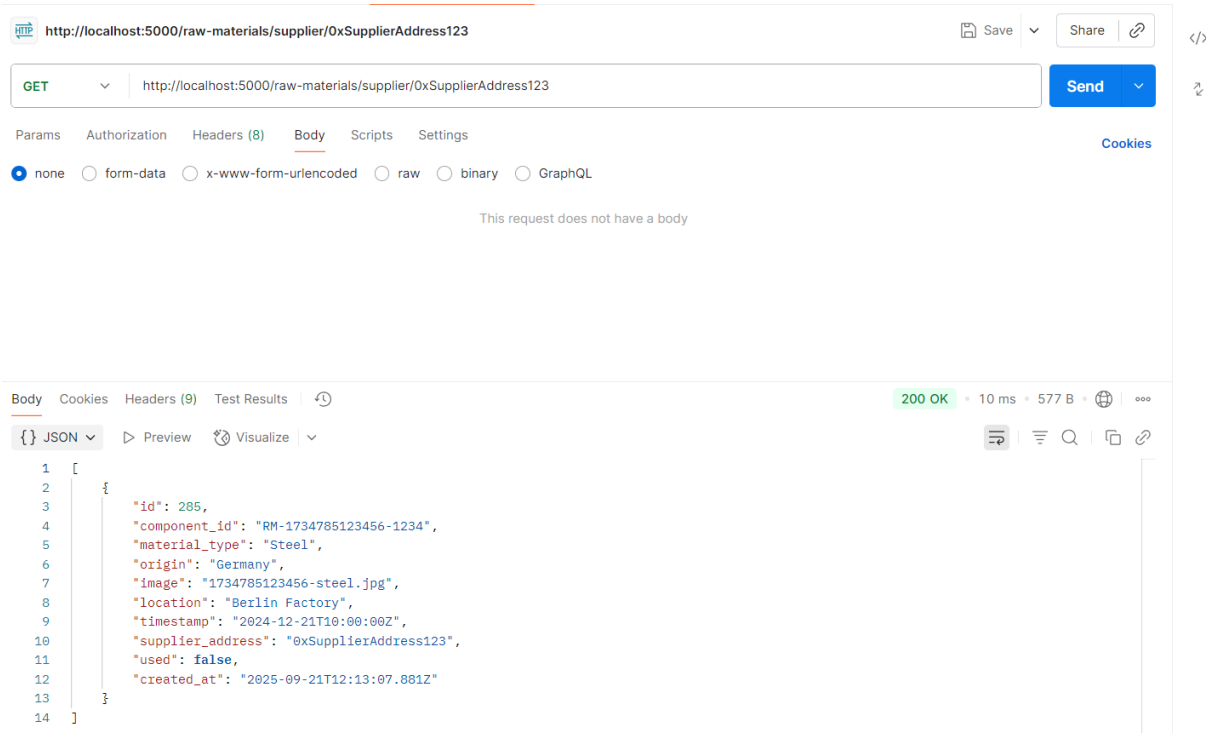

Figure 6.2.16 Postman Testing (Get Raw Materials by Supplier)

5. Manufacturer Module

Add Component

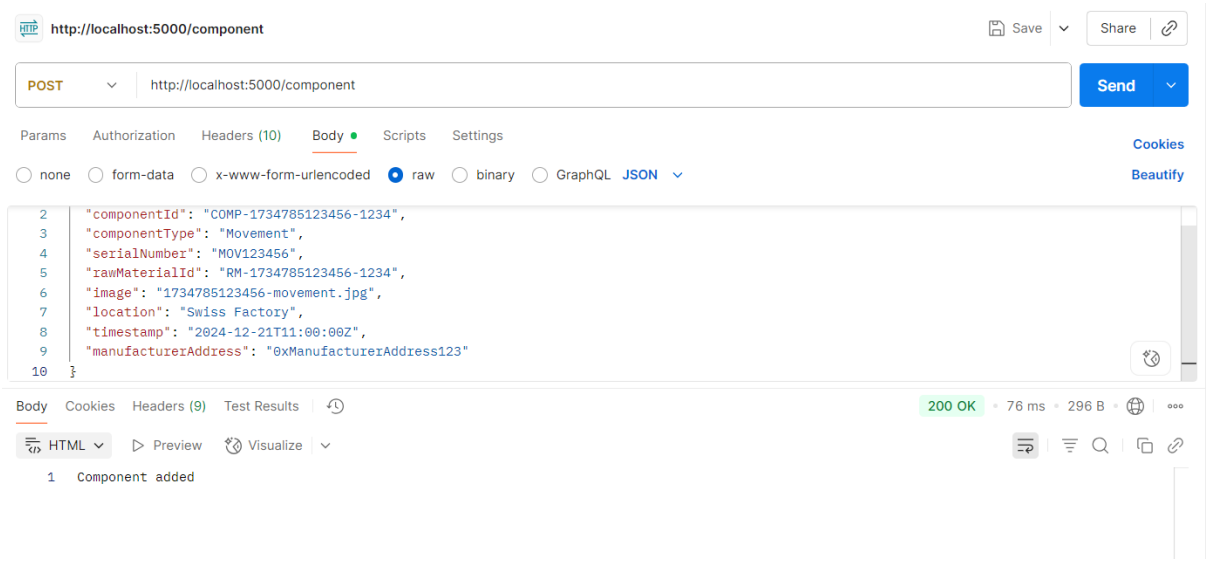

Figure 6.2.17 Postman Testing (Add Component)

Get All Components

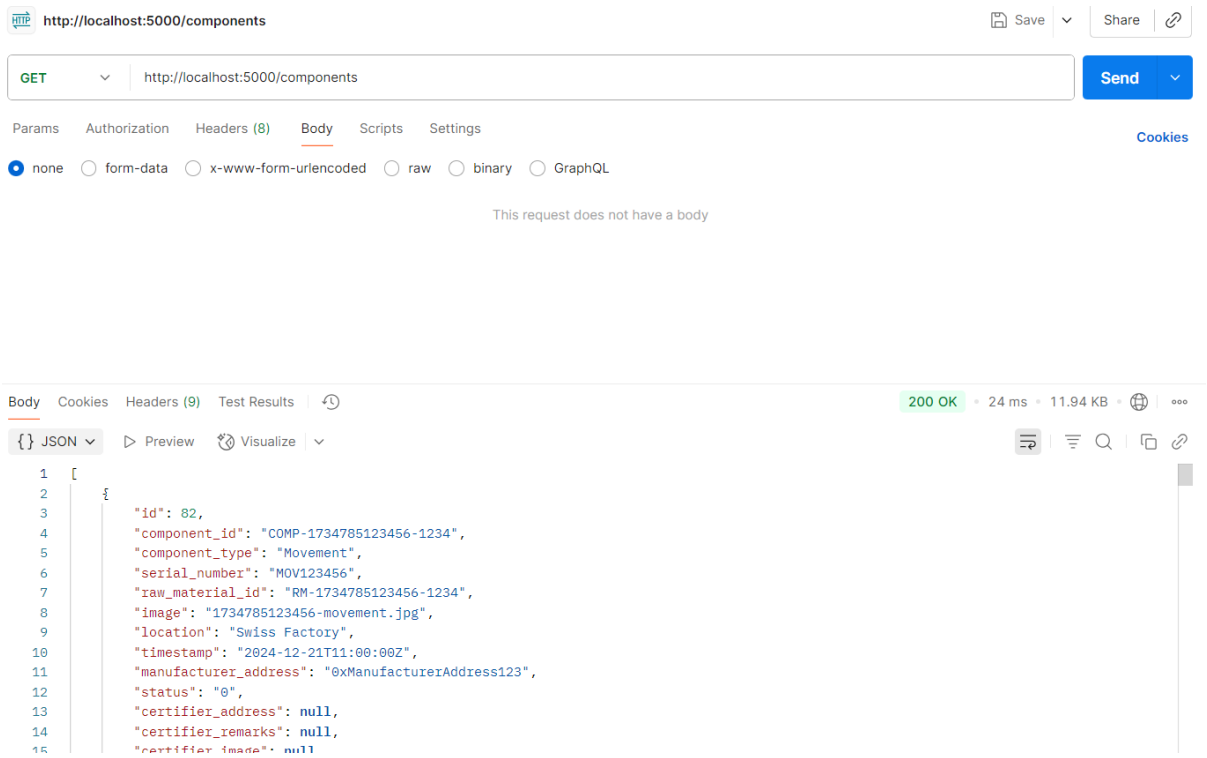

Figure 6.2.18 Postman Testing (Get All Components)

Get Specific Component

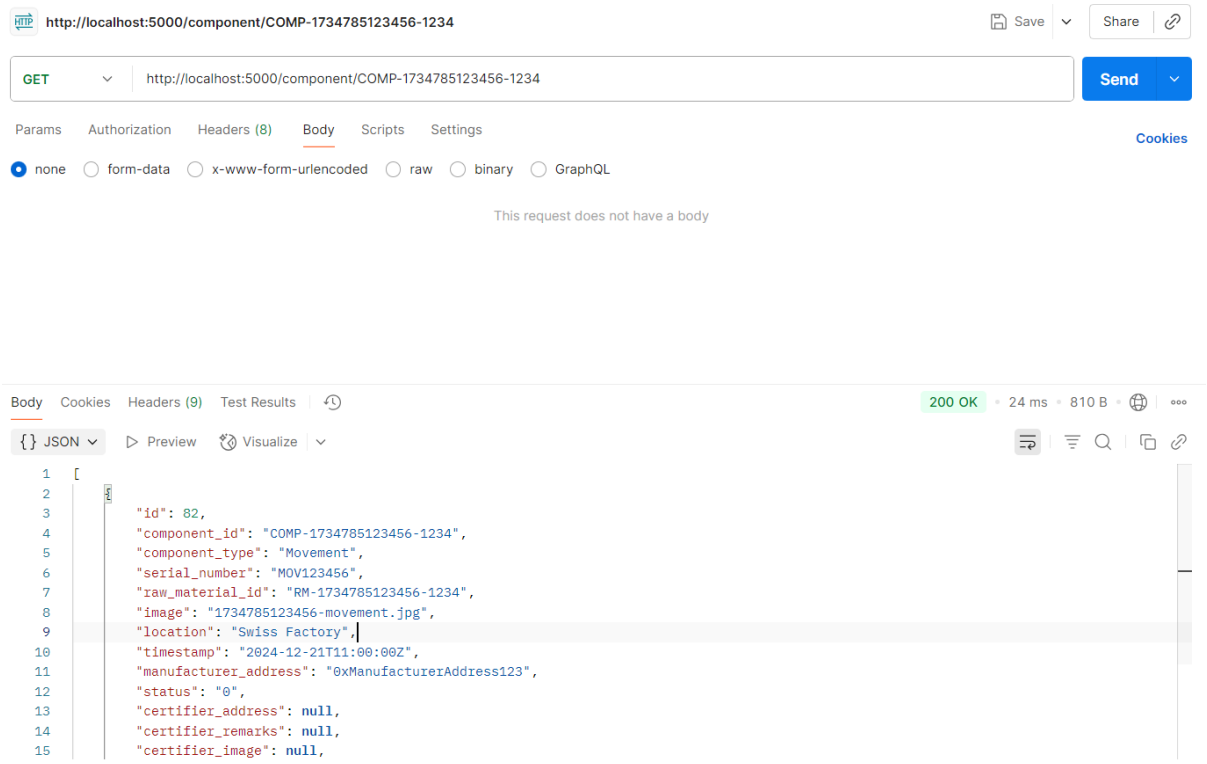

Figure 6.2.19 Postman Testing (Get Specific Component)

Get Components by Manufacturer

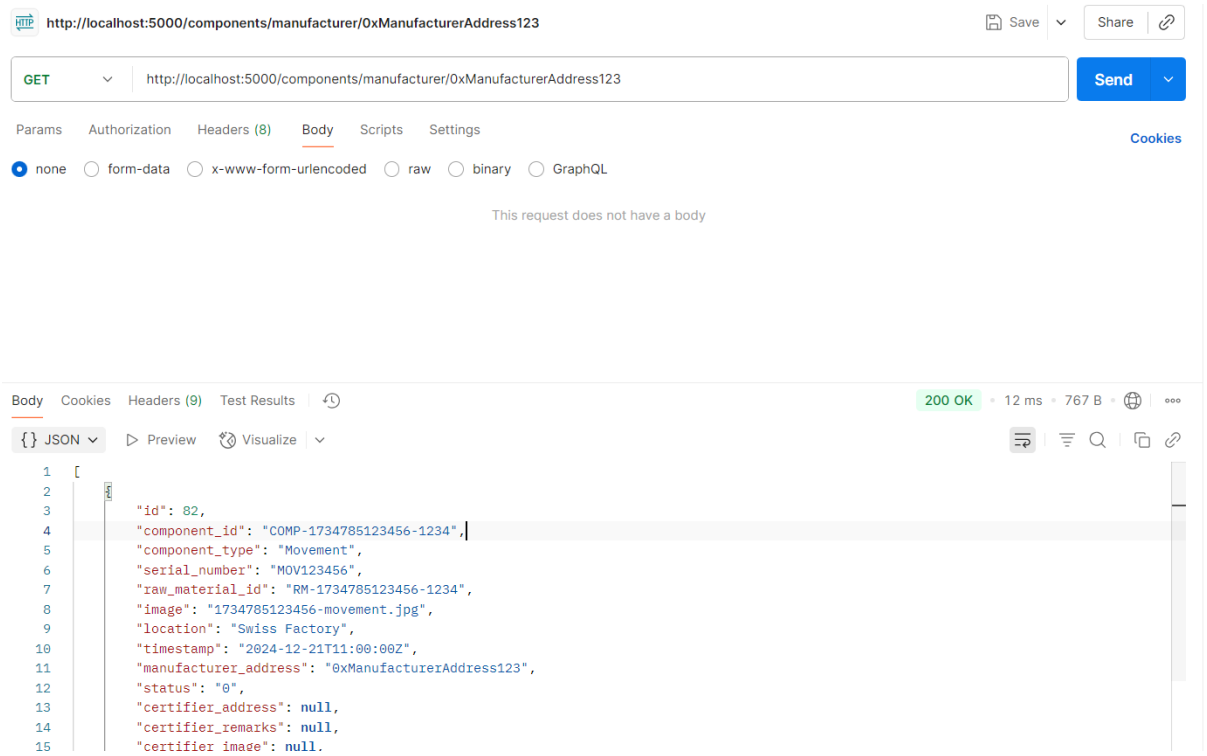

Figure 6.2.20 Postman Testing (Get Components by Manufacturer)

6. Certifier Module

Certify Component

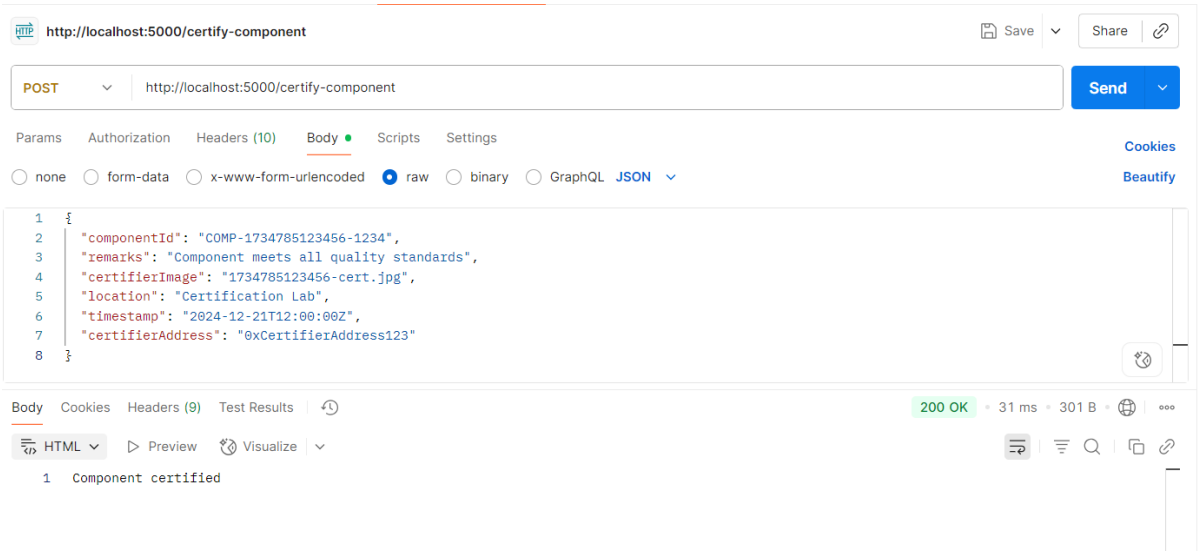

Figure 6.2.21 Postman Testing (Certify Component)

Get Certified Components

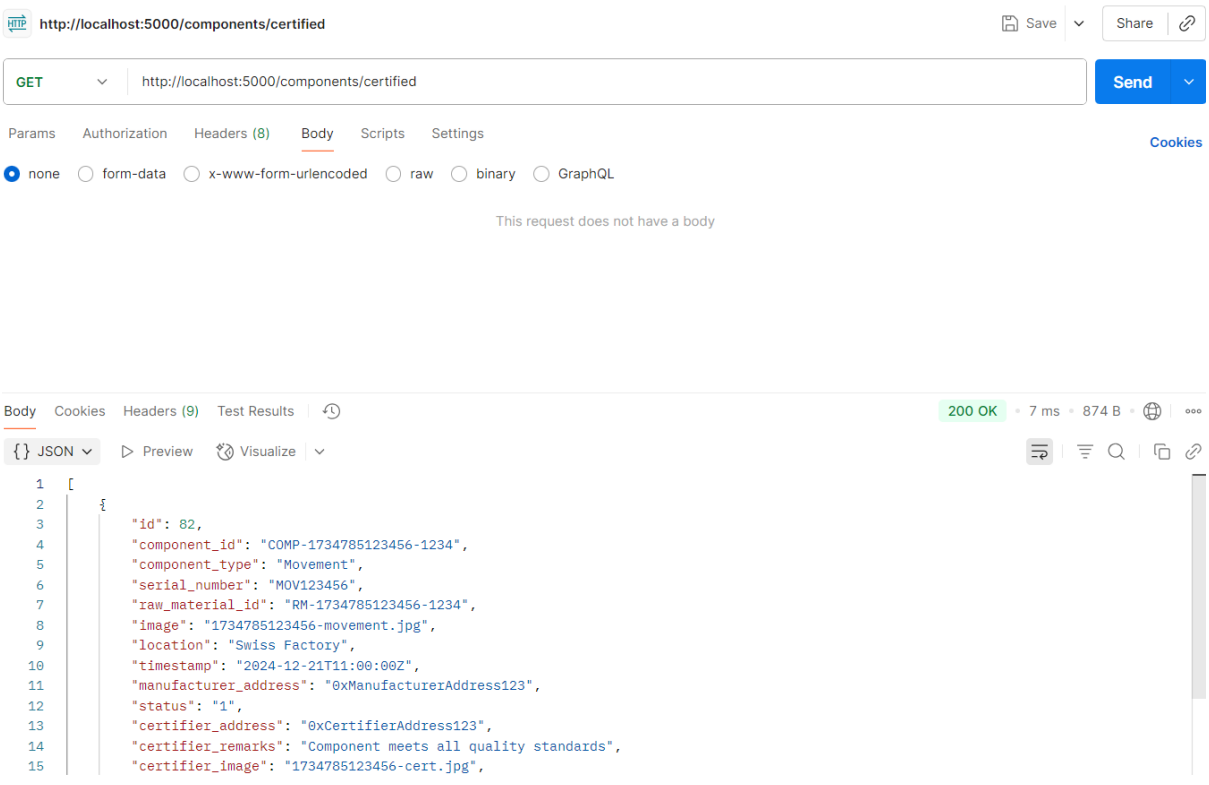

Figure 6.2.22 Postman Testing (Get Certified Components)

7. Assembler Module

Create Watch (with NFT generation)

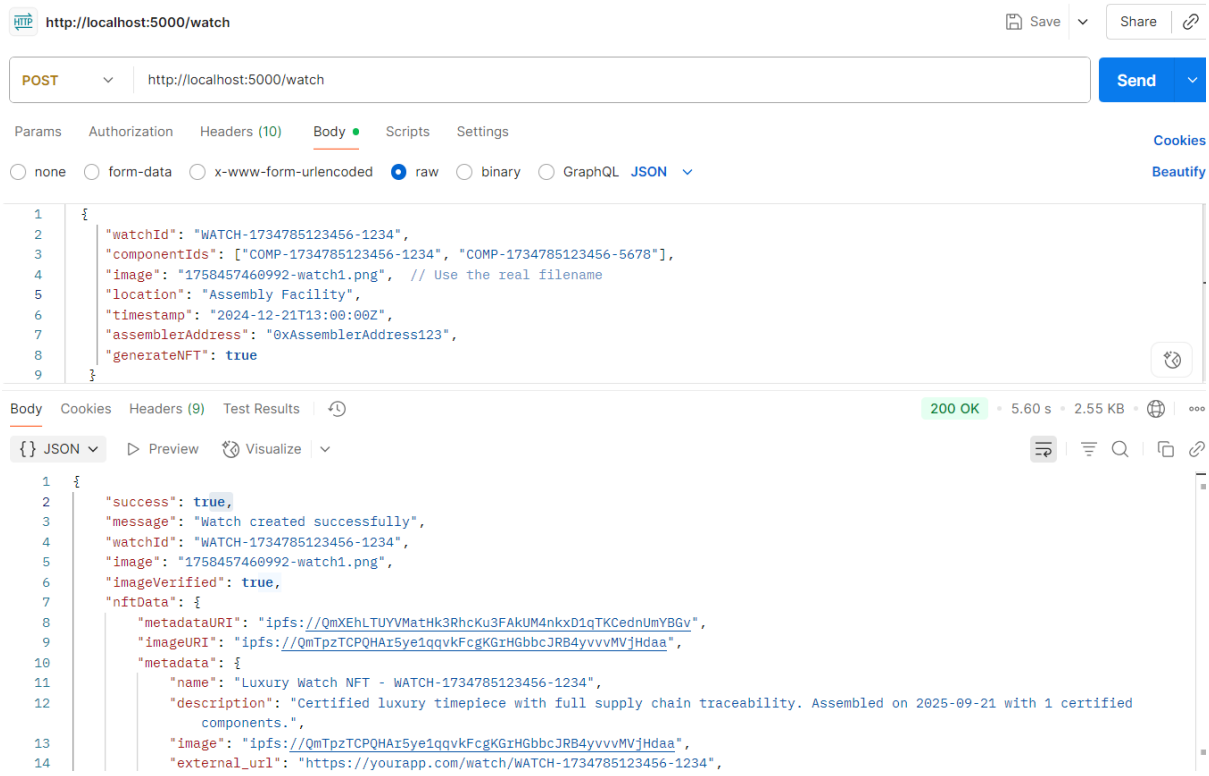

Figure 6.2.23 Postman Testing (Create Watch - with NFT generation)

Get All Watches

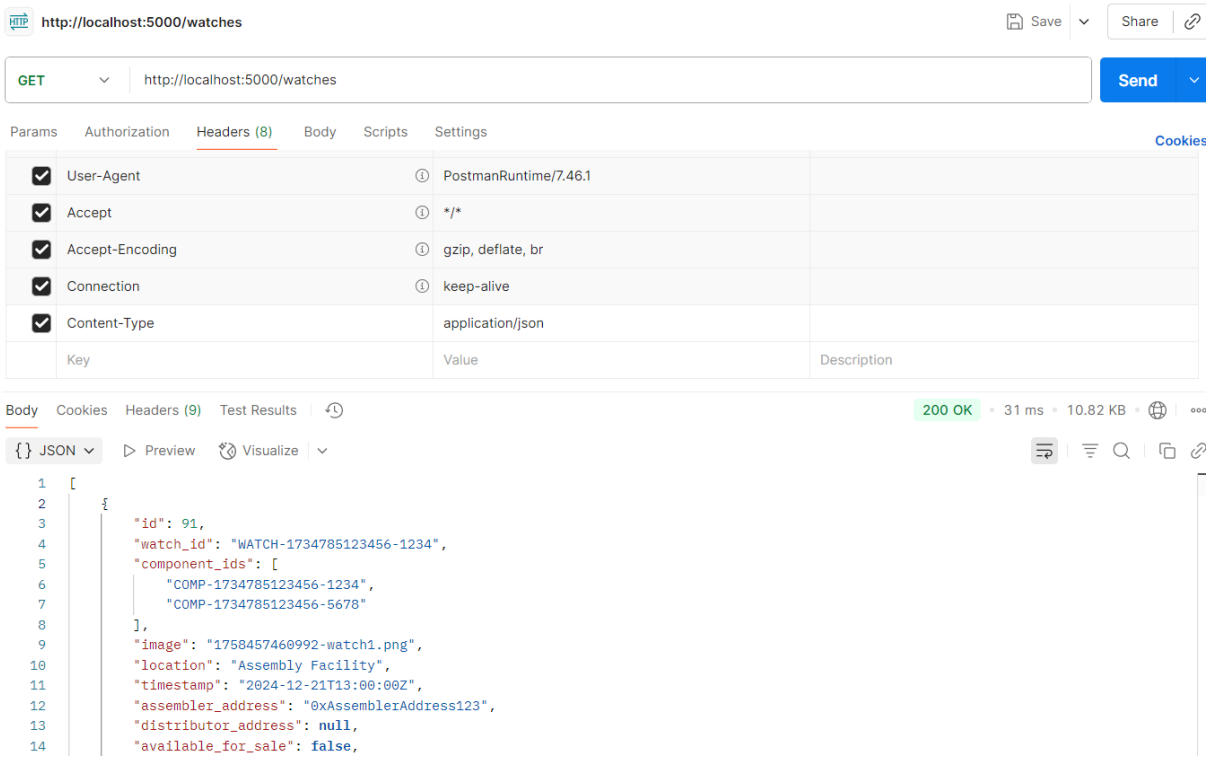

Figure 6.2.24 Postman Testing (Get All Watches)

Get Specific Watch

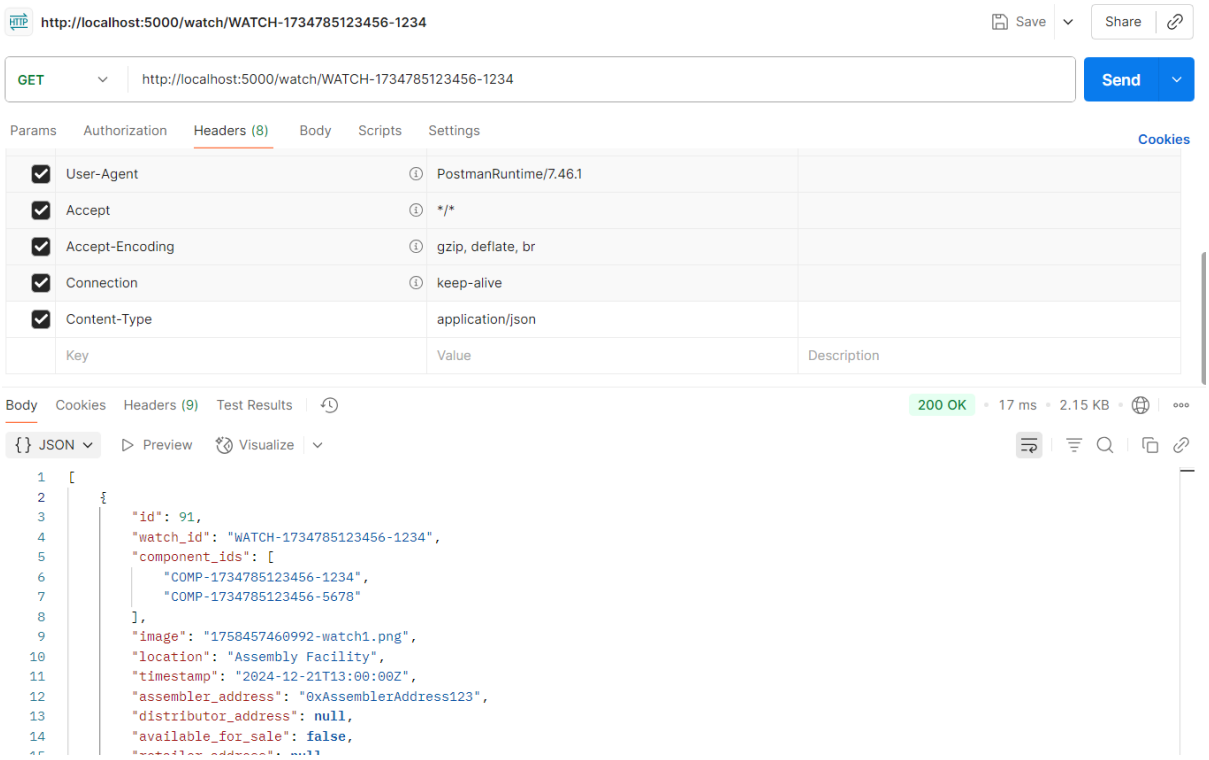

Figure 6.2.25 Postman Testing (Get Specific Watch)

Get Watches by Assembler

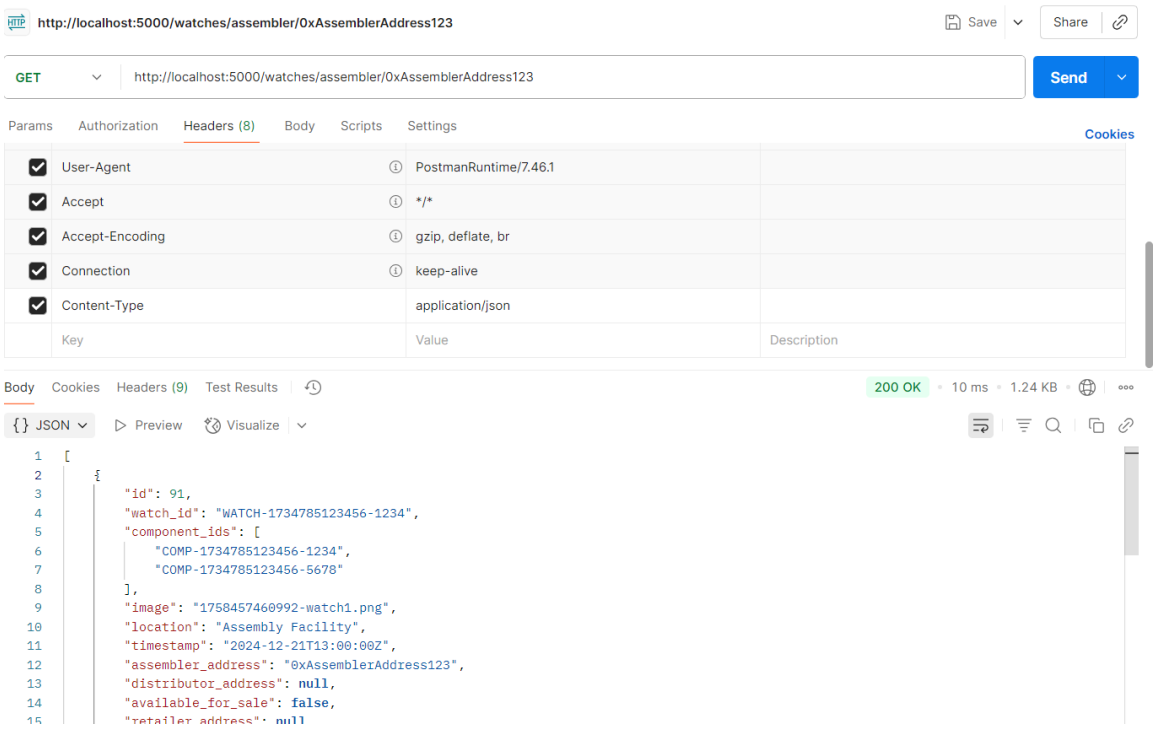

Figure 6.2.26 Postman Testing (Get Watches by Assembler)

8. AI Analysis Module

Test Pinata Connection

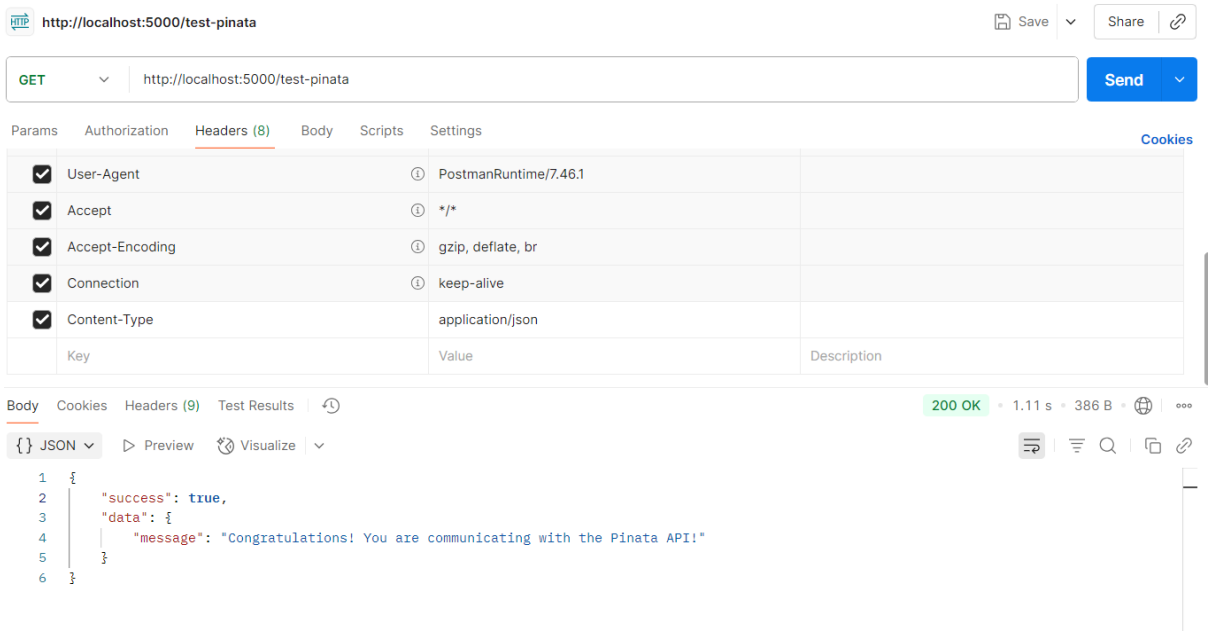

Figure 6.2.27 Postman Testing (Test Pinata Connection)

Test IPFS Upload

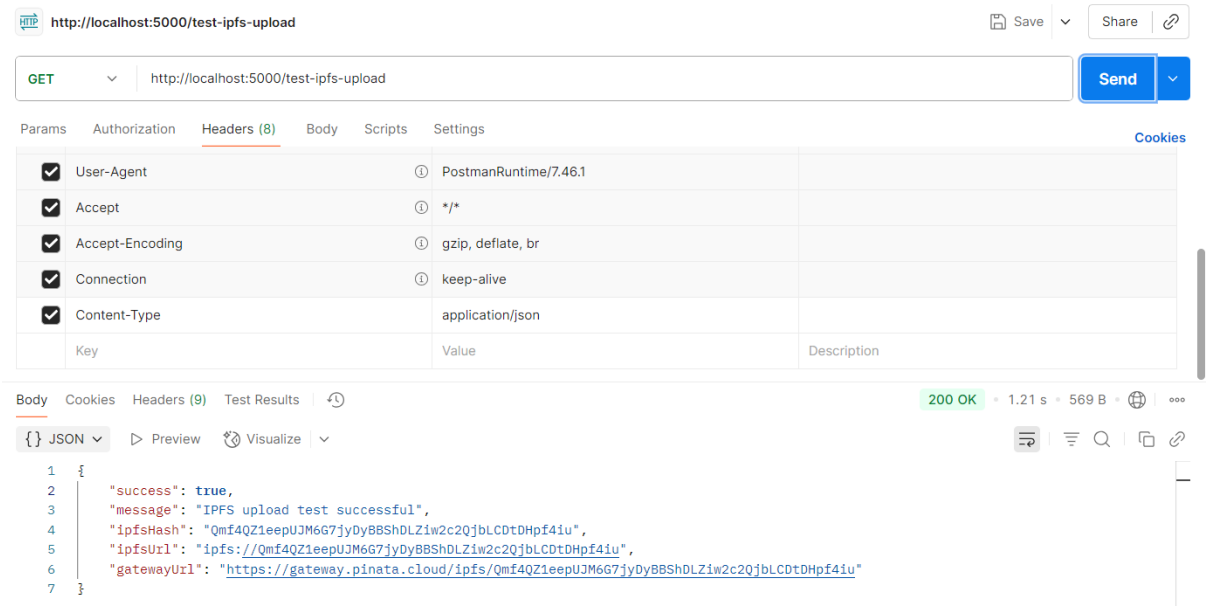

Figure 6.2.28 Postman Testing (Test IPFS Upload)

Analyze Watch with AI

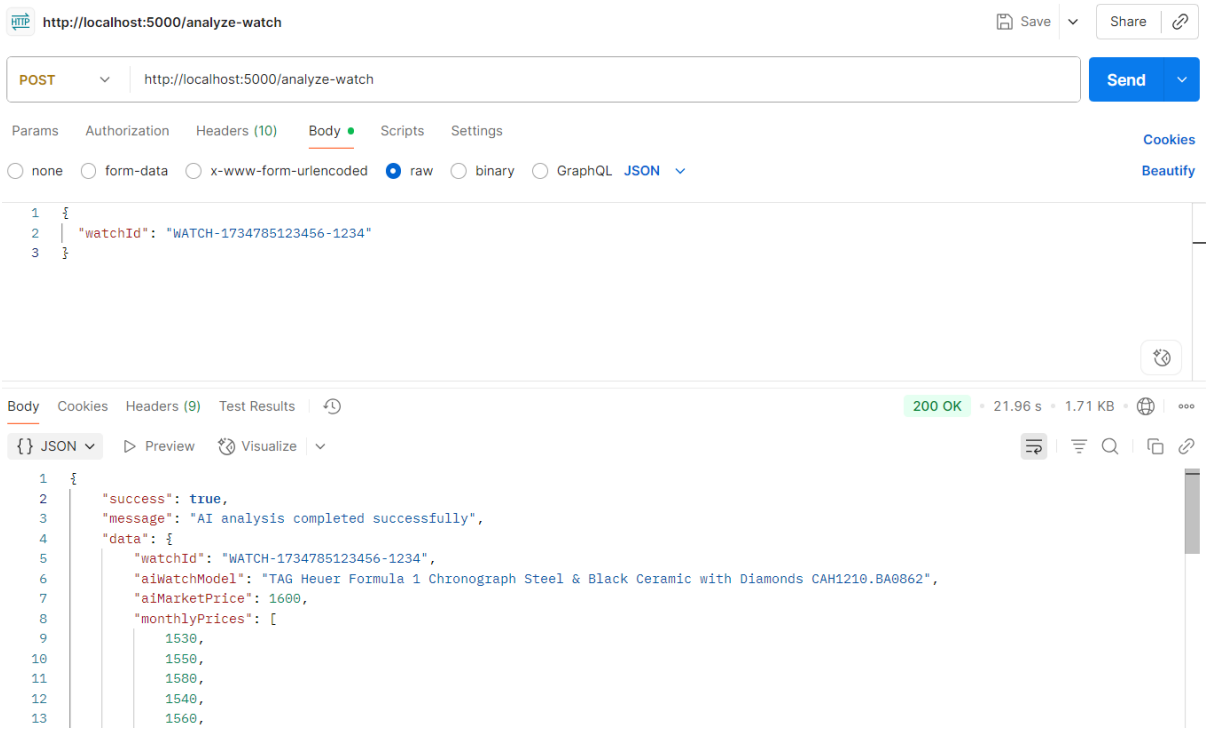

Figure 6.2.29 Postman Testing (Analyze Watch with AI)

Get AI Analysis Results

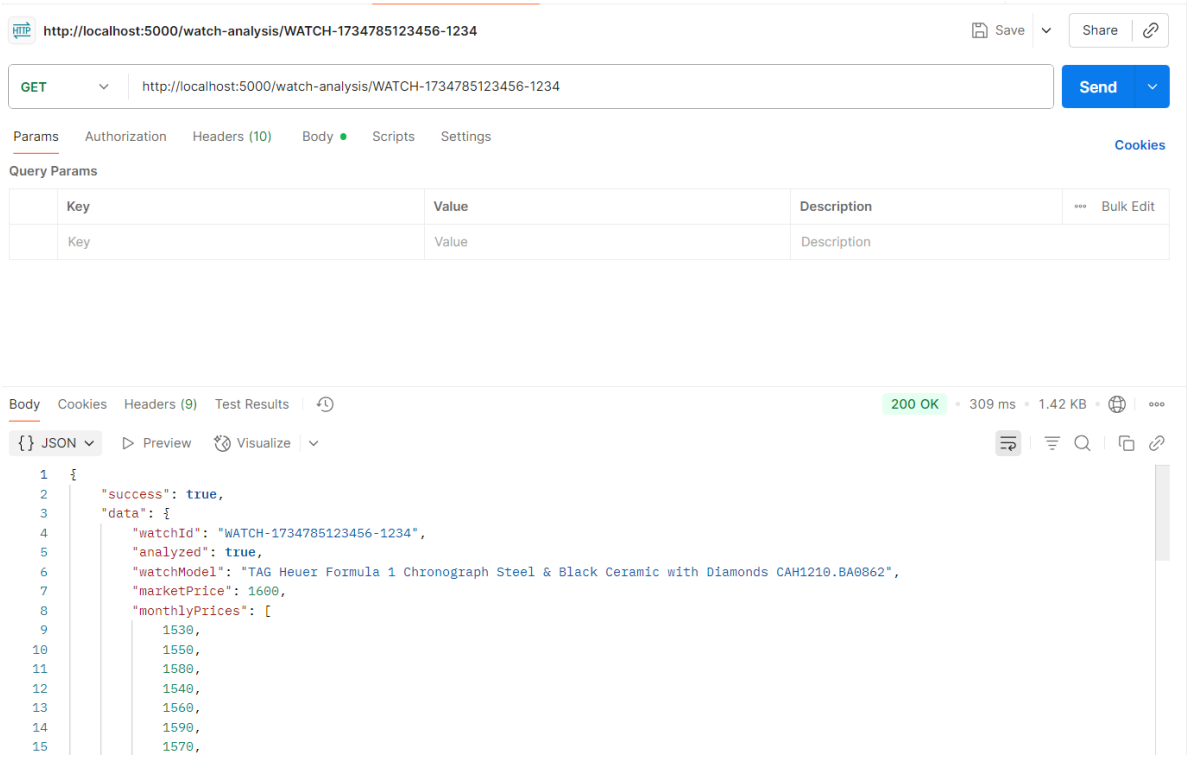

Figure 6.2.30 Postman Testing (Get AI Analysis Results)

9. NFT Metadata Module

Generate NFT Metadata

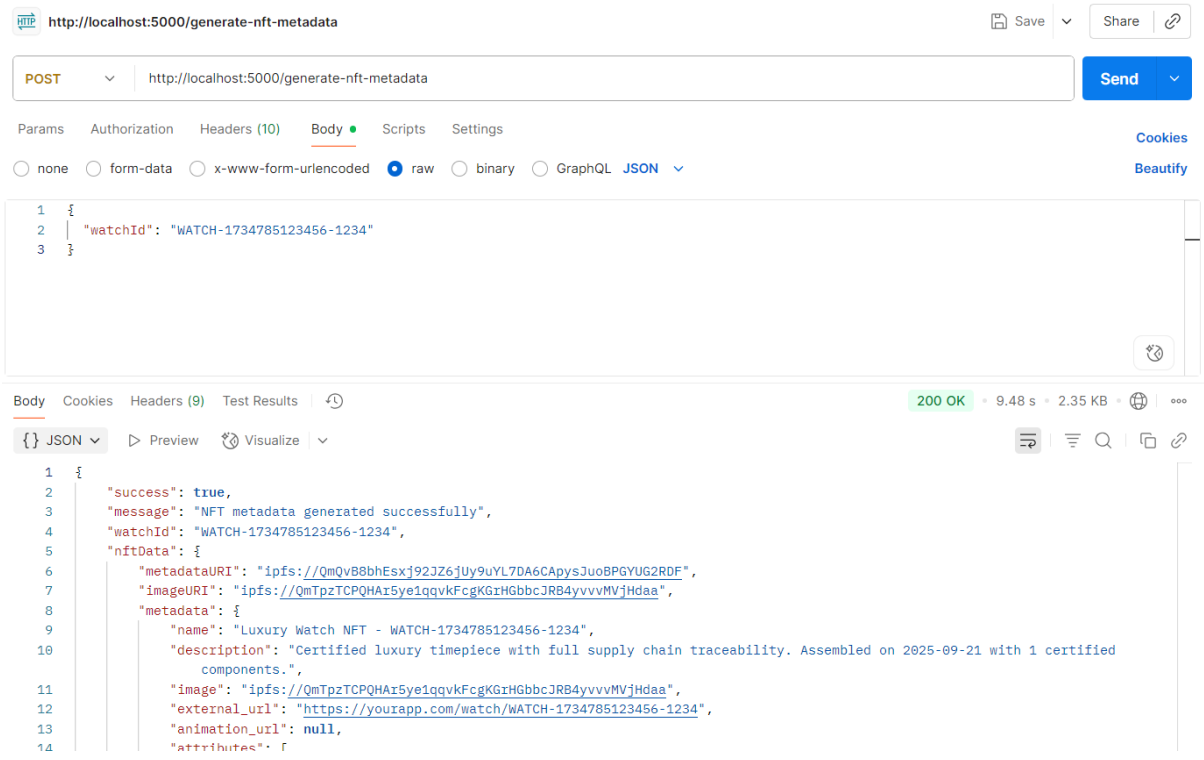

Figure 6.2.31 Postman Testing (Generate NFT Metadata)

Update NFT Metadata

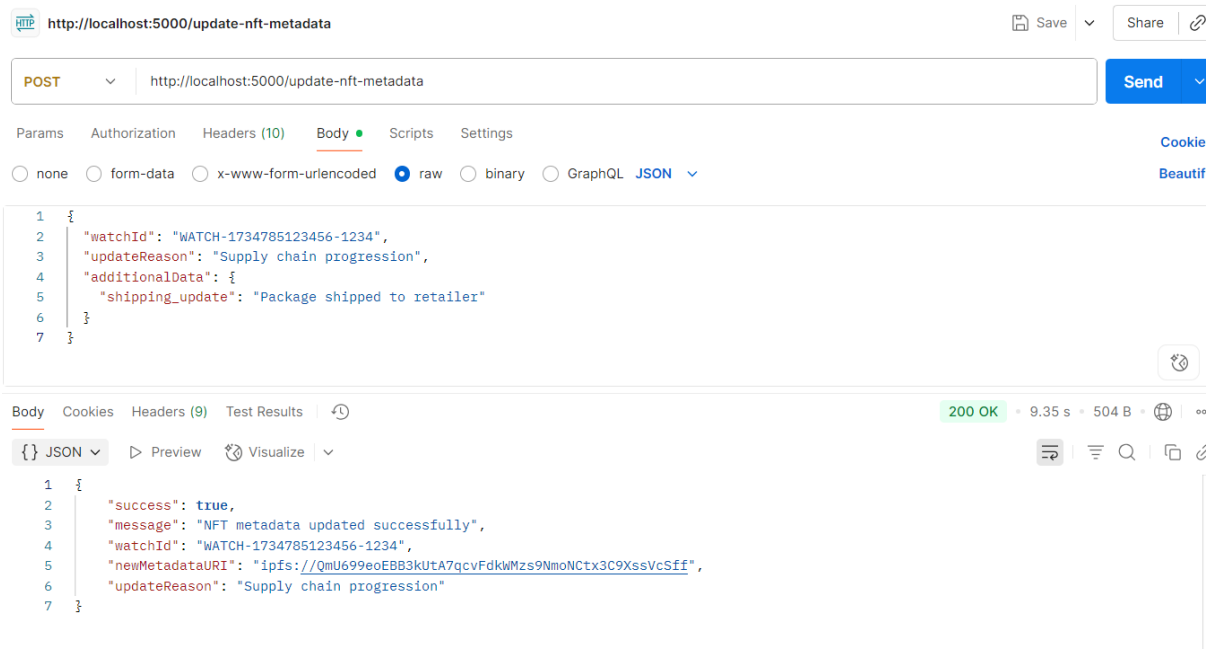

Figure 6.2.32 Postman Testing (Update NFT Metadata)

Get NFT Metadata

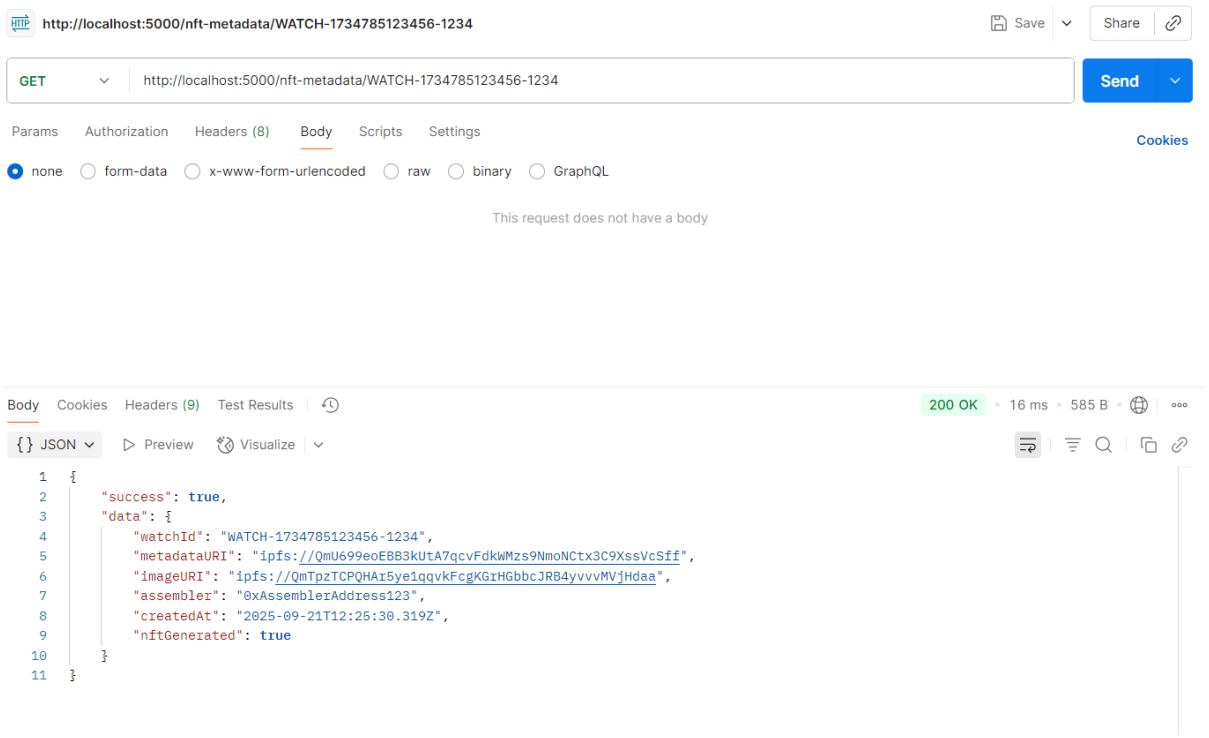

Figure 6.2.33 Postman Testing (Get NFT Metadata)

10. Distributor Module

Update Shipping Status

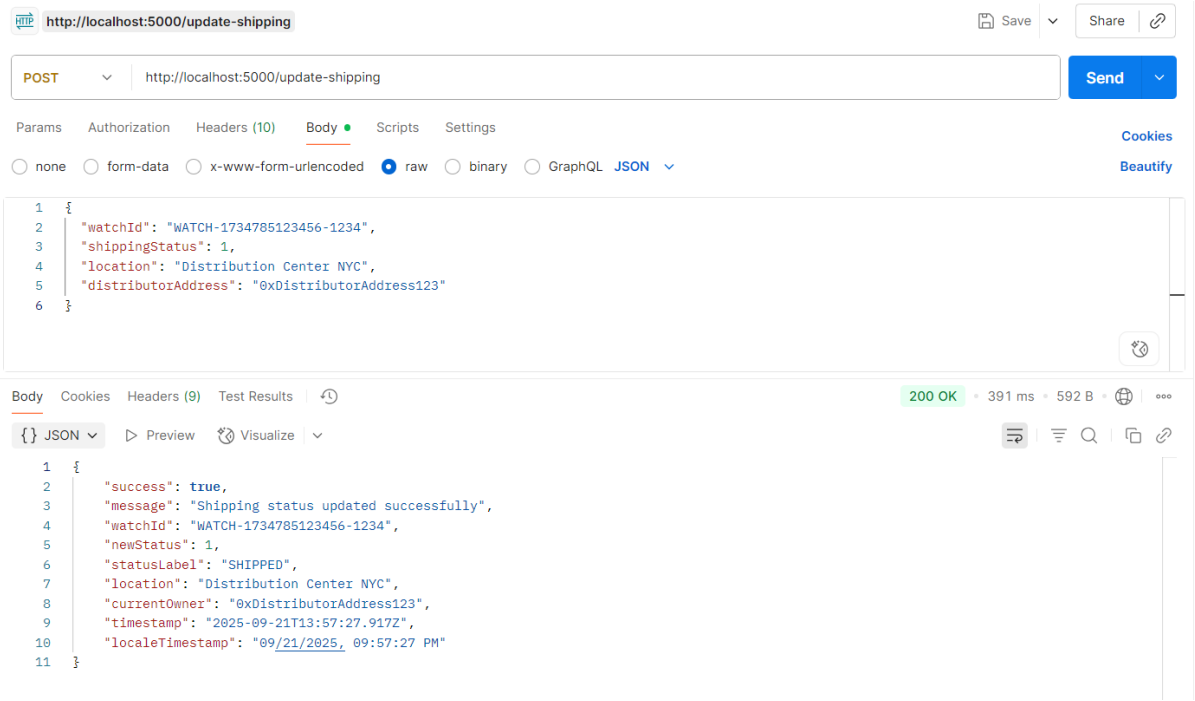

Figure 6.2.34 Postman Testing (Update Shipping Status)

Get Watches for Shipping

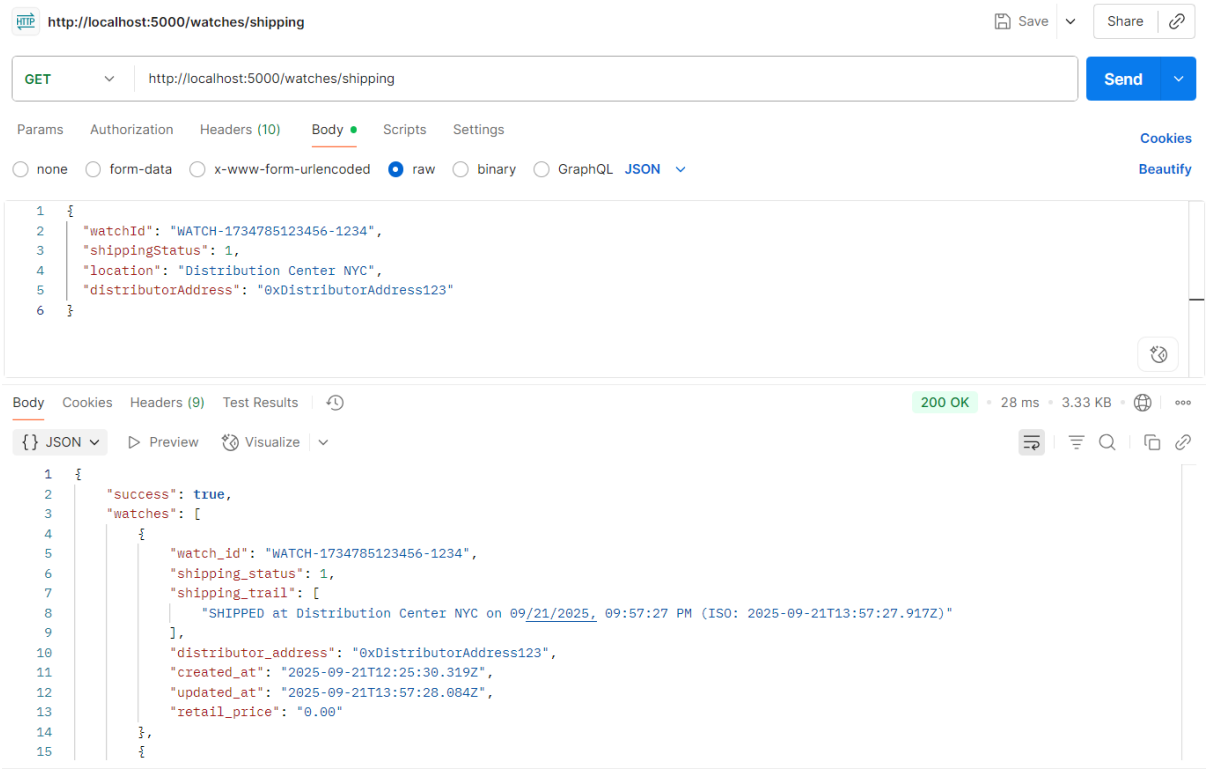

Figure 6.2.35 Postman Testing (Get Watches for Shipping)

Get Shipping Statistics

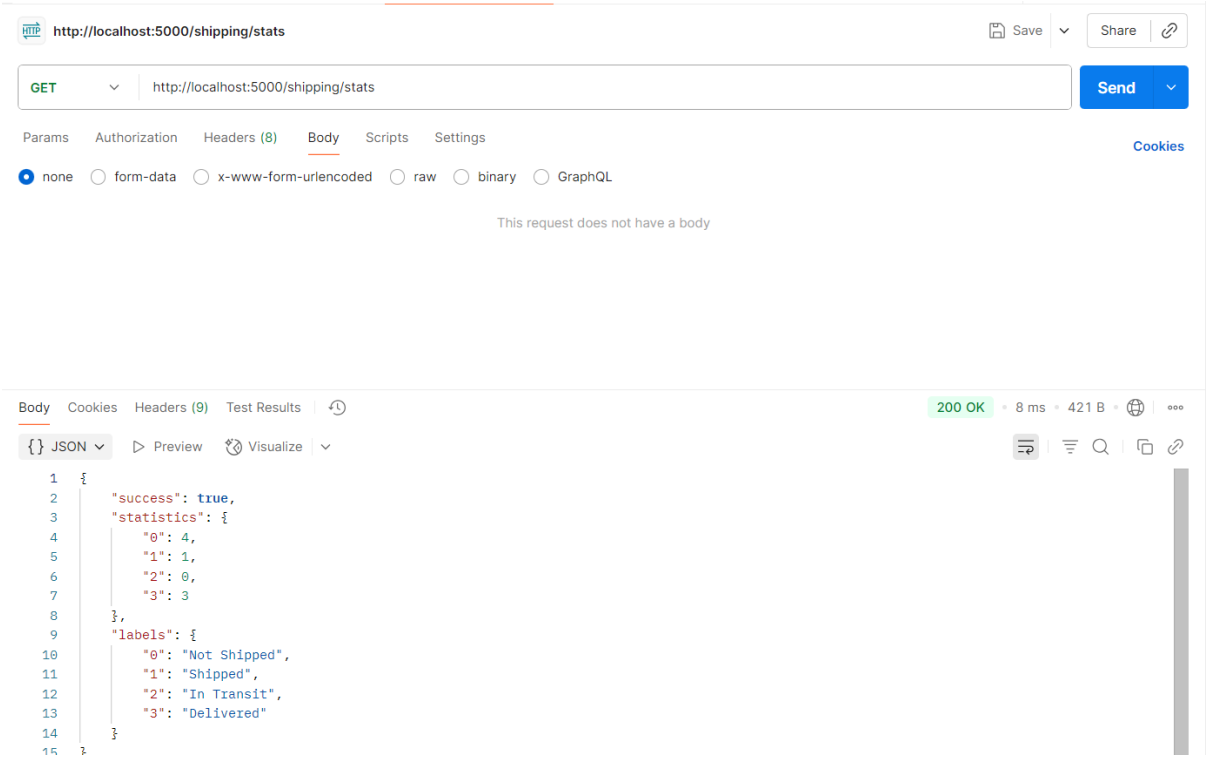

Figure 6.2.36 Postman Testing (Get Shipping Statistics)

11. Retailer Module

Mark Watch Available for Sale

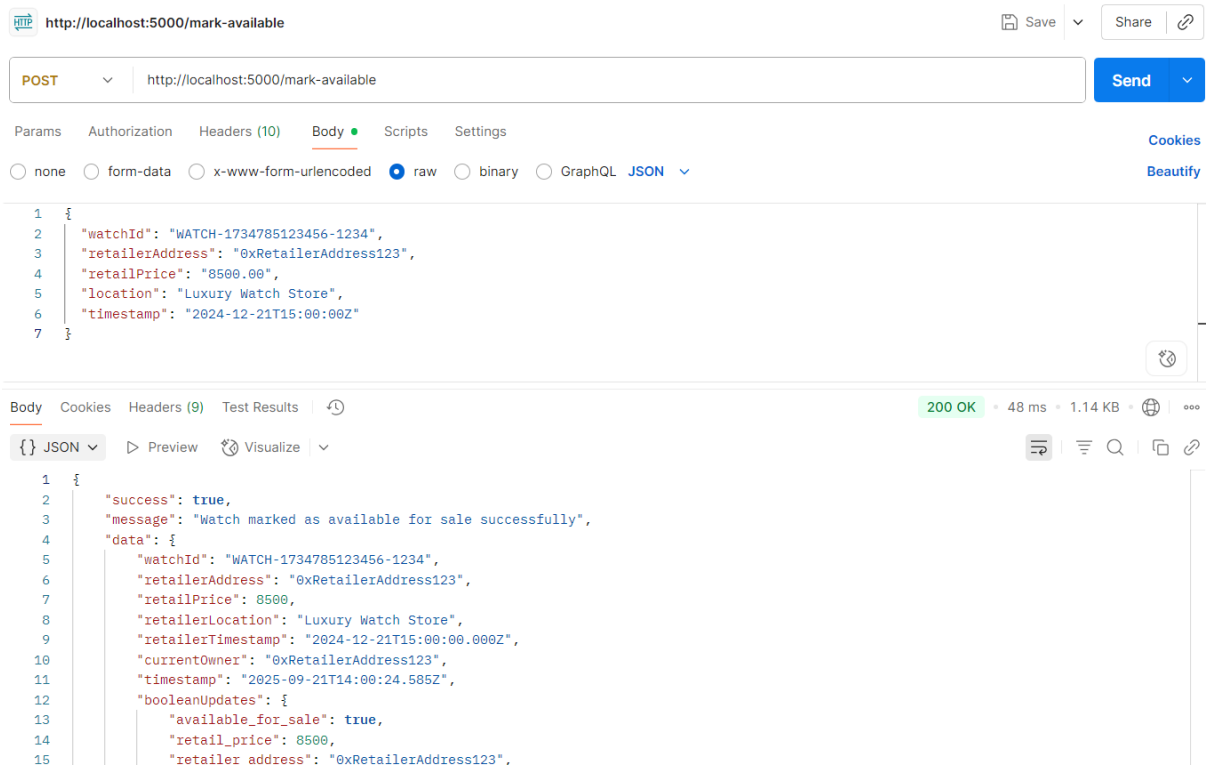

Figure 6.2.37 Postman Testing (Mark Watch Available for Sale)

Get Available Watches

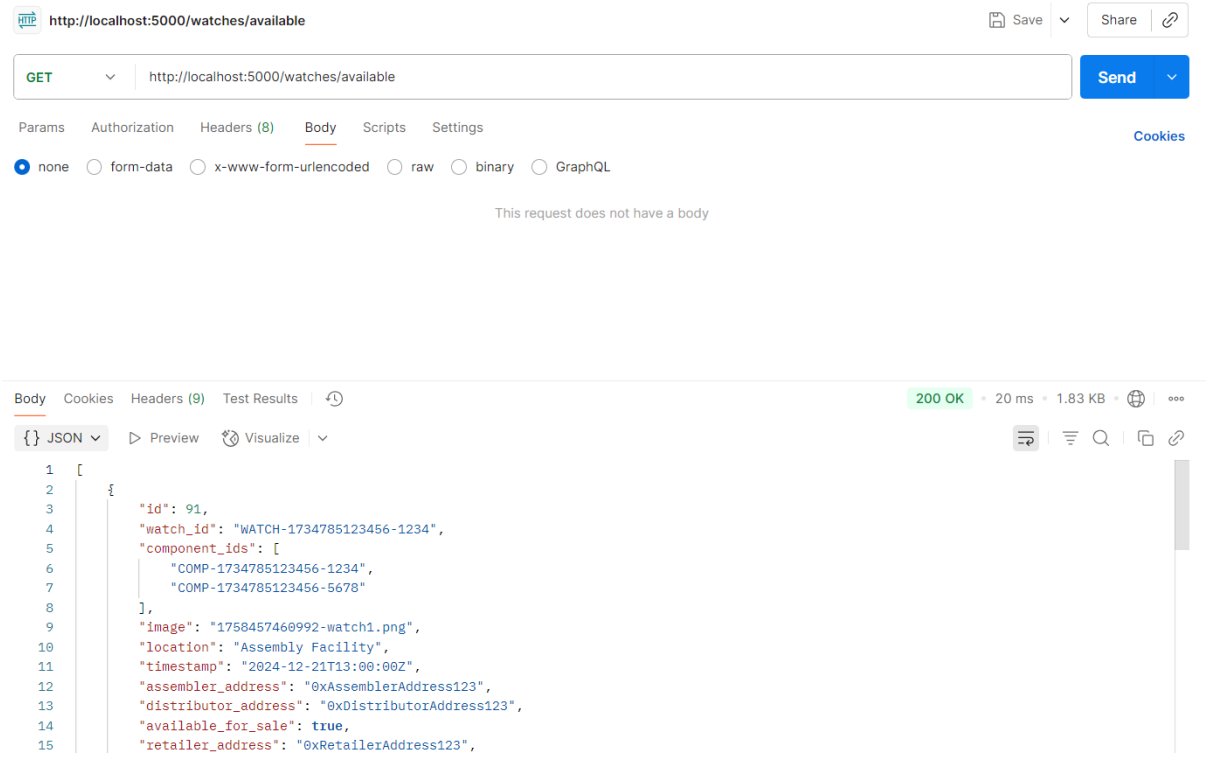

Figure 6.2.38 Postman Testing (Get Available Watches)

Get Watches for Sale

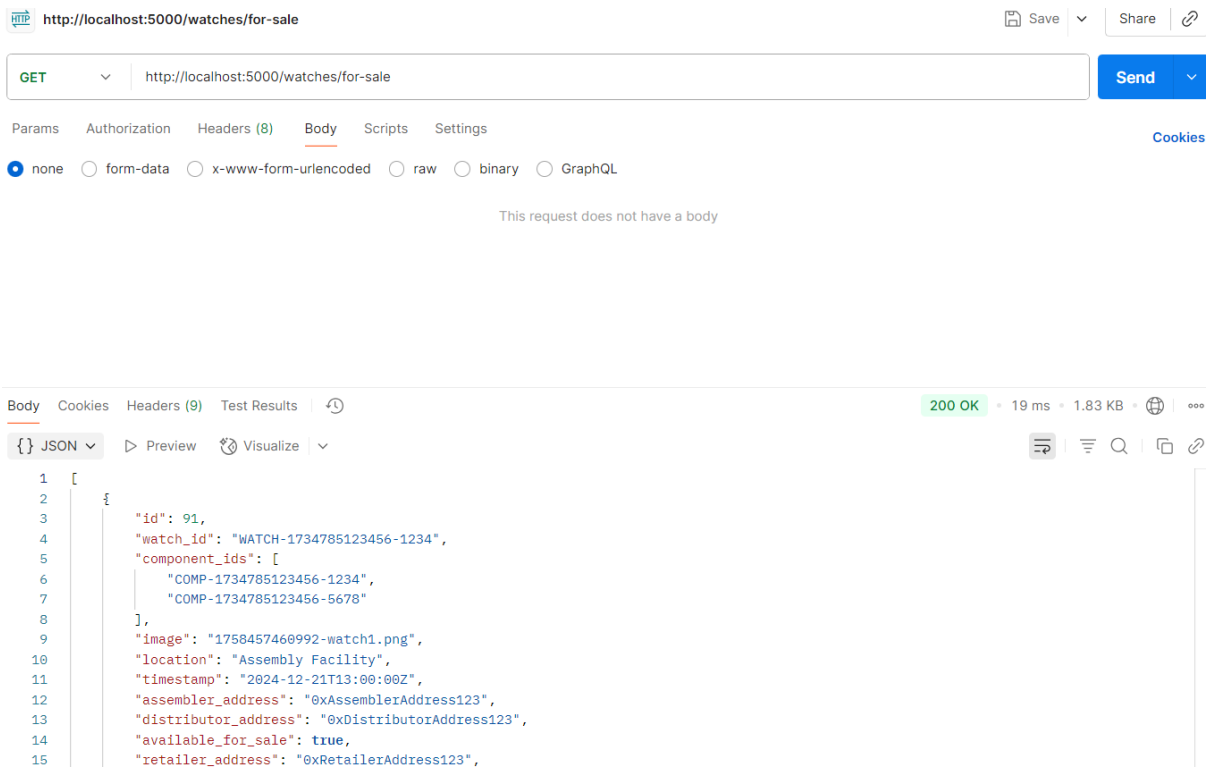

Figure 6.2.39 Postman Testing (Get Watches for Sale)

12. Consumer Module

Purchase Watch

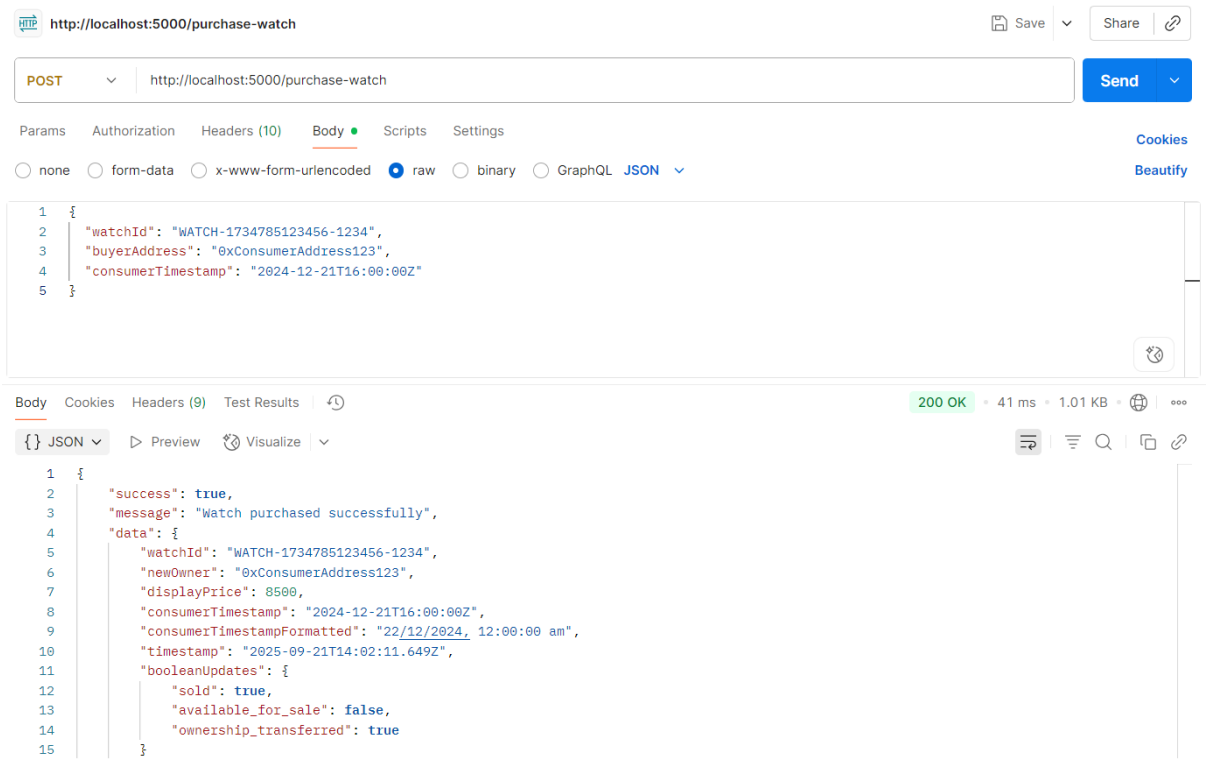

Figure 6.2.40 Postman Testing (Purchase Watch)

Transfer Ownership

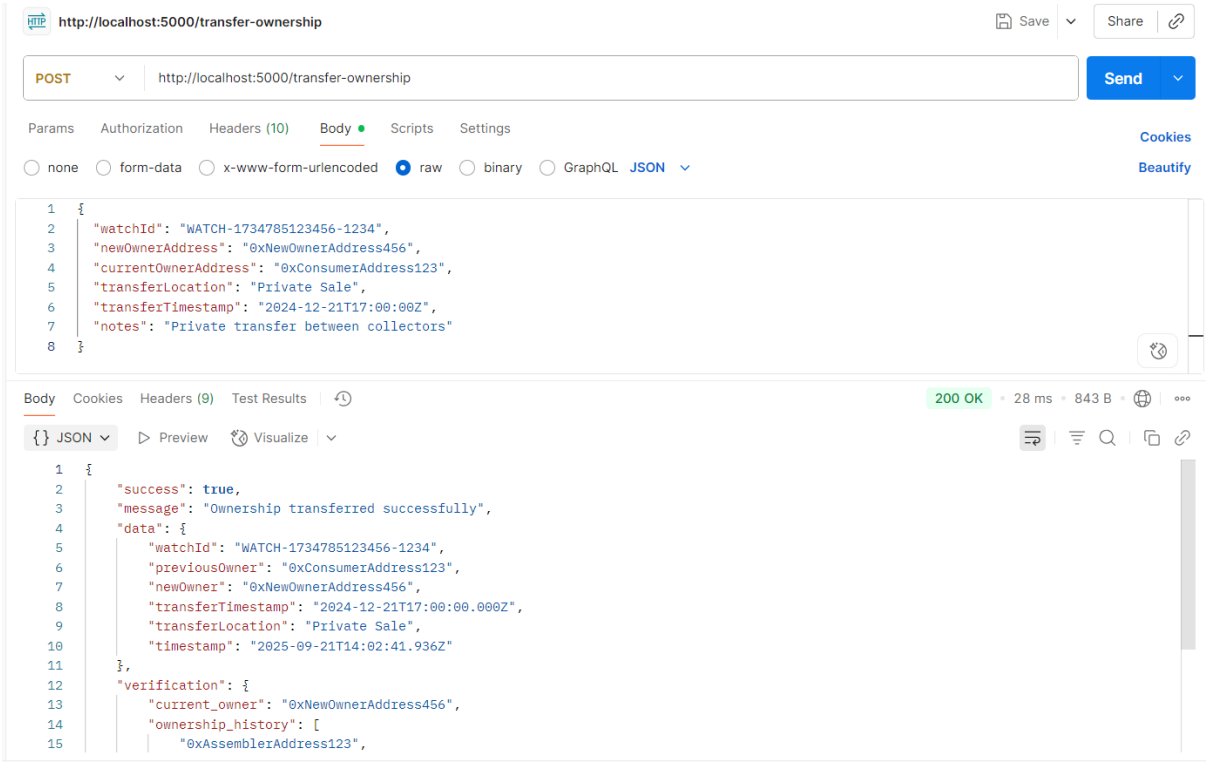

Figure 6.2.41 Postman Testing (Transfer Ownership)

Get Consumer Statistics

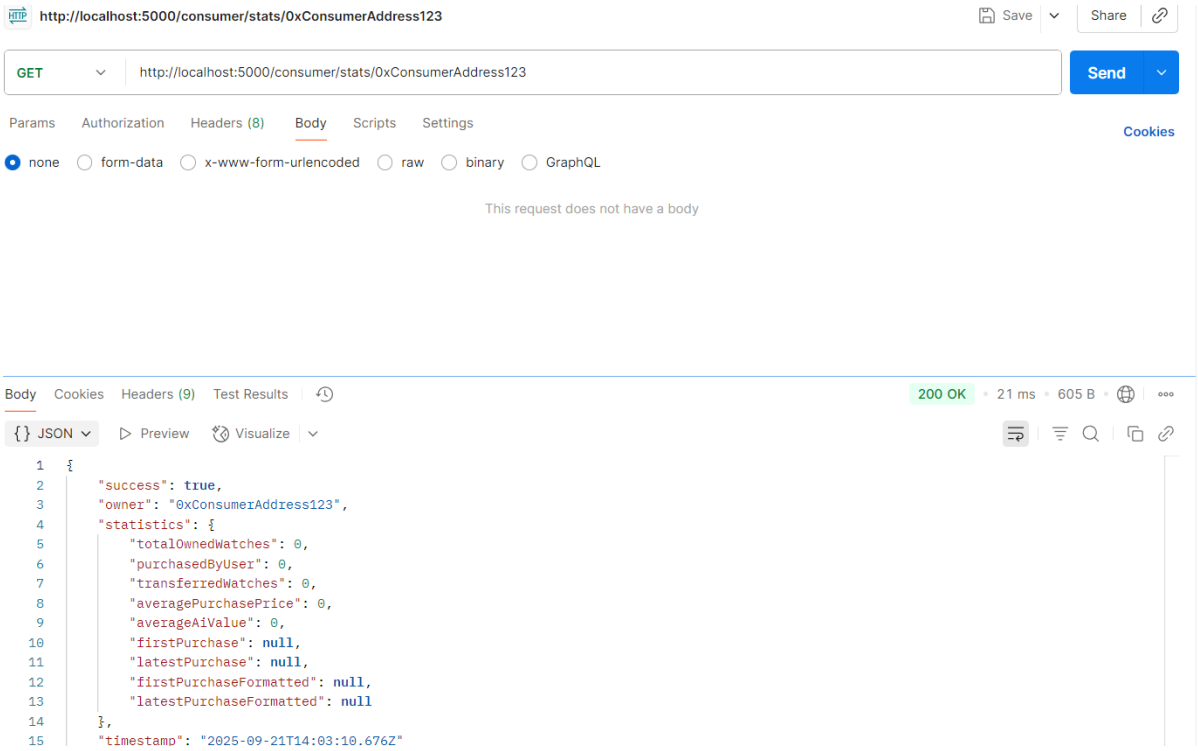

Figure 6.2.42 Postman Testing (Get Consumer Statistics)

13. Traceability and History

Get Watch Traceability

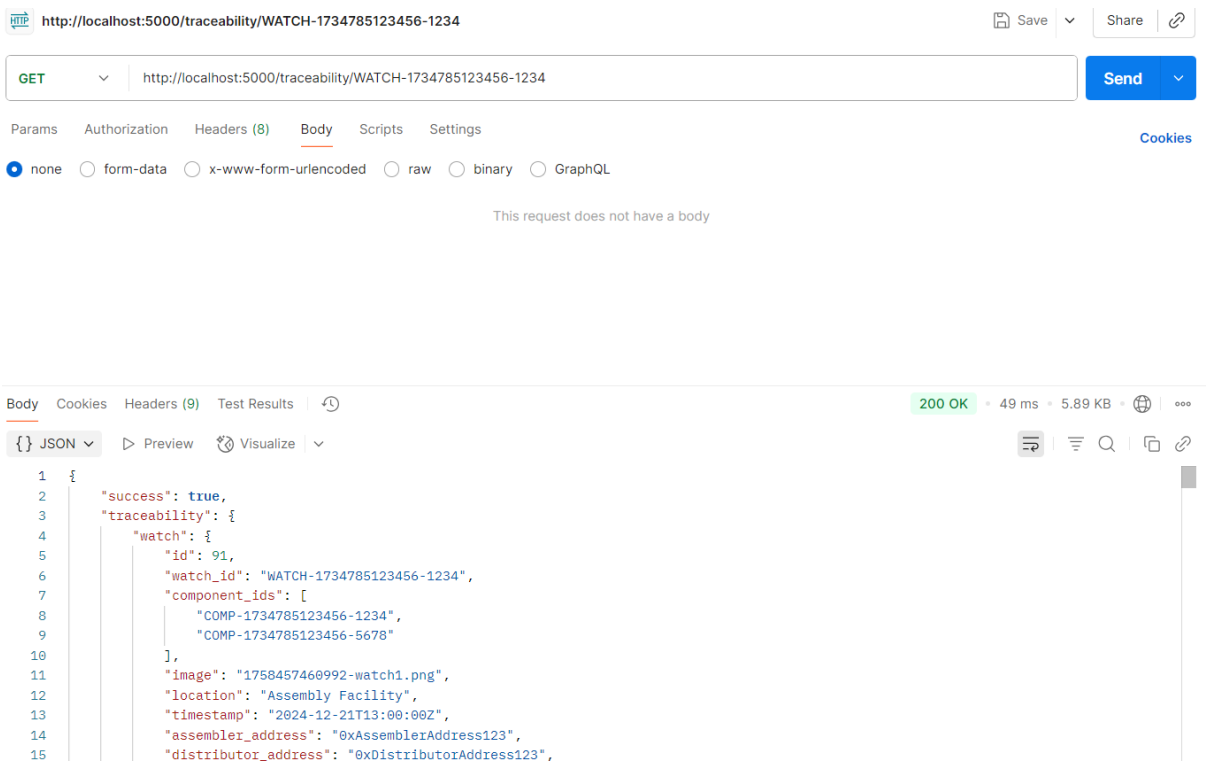

Figure 6.2.43 Postman Testing (Get Watch Traceability)

Get Ownership History

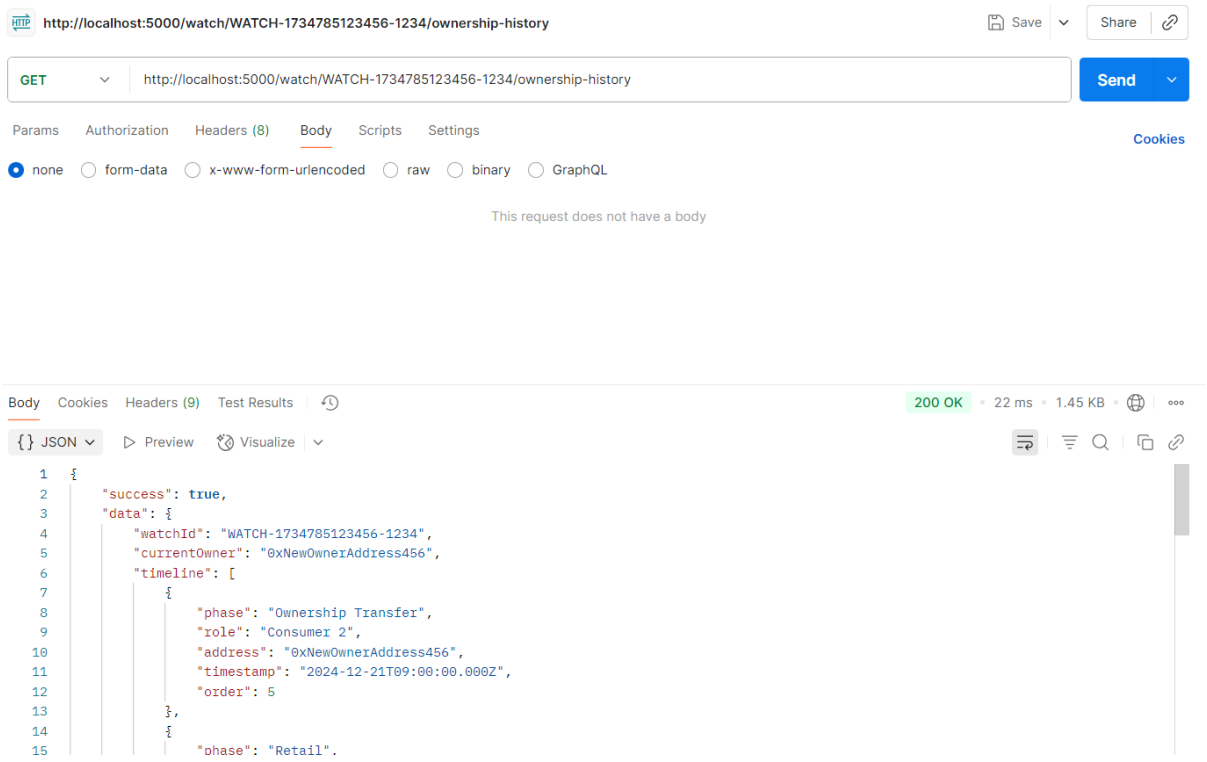

Figure 6.2.44 Postman Testing (Get Ownership History)

Get Transfer History

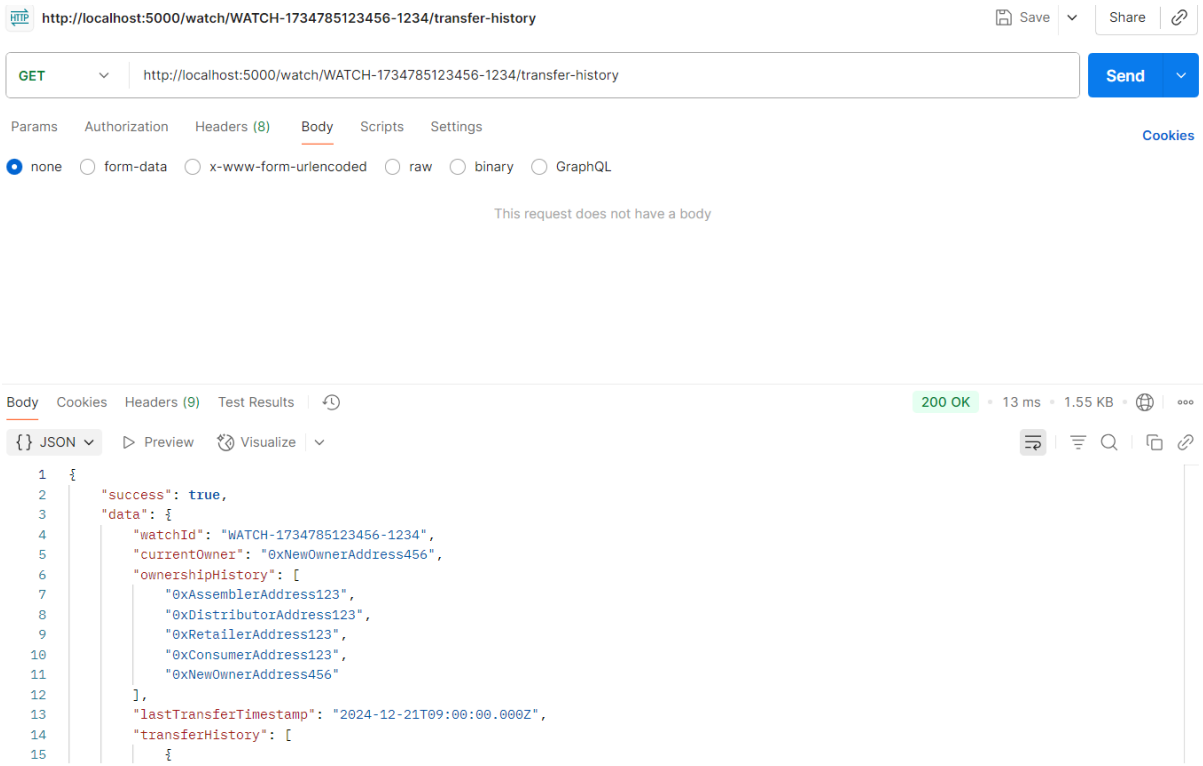

Figure 6.2.45 Postman Testing (Get Transfer History)

Figure 6.2.46 Chai and Mocha Backend Testing 1

```
FIXED: Detailed verification result:
=== FIXED: PURCHASE WATCH REQUEST ===
FIXED: Purchase validation data:
FIXED: All validations passed - proceeding with purchase
FIXED: No consumer timestamp provided, generating current time
FIXED: Final timestamp processing:
FIXED: Purchase completed successfully with transaction
FIXED: Purchase verification:
FIXED: Fetching watch with enhanced image debugging:
FIXED: Watch found with enhanced image analysis:
  Image file exists: true at C:\Users\USER\Desktop\Testing\final\FYP\backend\public\uploads\watch\test-image.png
FIXED: Sending enhanced response with image debugging
      ✓ should follow complete end-to-end supply chain process (51ms)
Test data cleared
Test database disconnected
Mocks cleaned up


  78 passing (4s)
```

Figure 6.2.47 Chai and Mocha Backend Testing 2

Effective backend system testing requires a comprehensive approach integrating both manual and automated methodologies. Postman facilitates manual API endpoint validation and exploratory testing through its intuitive interface, enabling rapid prototyping and collaborative documentation during development phases. Chai and Mocha provide a robust automated testing framework, where Mocha serves as the test runner while Chai offers expressive assertion capabilities supporting behavior-driven and test-driven development paradigms. This dual methodology addresses the limitations of singular testing approaches by combining Postman's flexibility for immediate development needs with Chai-Mocha's systematic, repeatable automation suitable for CI/CD pipeline integration. Such comprehensive testing strategies are particularly essential for complex backend architectures involving blockchain integration, IPFS data management, and NFT functionalities, where multiple service layer

interdependencies require rigorous validation to ensure system reliability, data integrity, and functional correctness across diverse operational scenarios.

Frontend Testing

The implementation of effective frontend testing requires a multi-tiered architectural approach that addresses distinct aspects of application functionality through specialized testing strategies. Unit testing forms the foundational layer, focusing on isolated component verification using React Testing Library and Jest frameworks. This approach enables developers to validate individual component rendering, state management, and user interaction handling in controlled environments. The methodology emphasizes testing user behavior rather than implementation details, aligning with contemporary best practices that prioritize maintainable and meaningful test coverage.

Integration testing constitutes the intermediate layer, examining component interactions with external dependencies, particularly API services and state management systems. Through the implementation of Mock Service Worker (MSW) technology, developers can simulate realistic API responses while maintaining test isolation and reliability. This approach facilitates comprehensive validation of asynchronous operations, error handling mechanisms, and data flow patterns without dependence on external service availability. The integration testing strategy proves particularly valuable for applications involving complex workflows such as blockchain transactions and IPFS data management, where external service reliability can significantly impact testing consistency.

```
PASS src/components/__tests__/WatchManager.integration.test.jsx (9.845 s)
  WatchManager Integration Tests
    √ renders initial form correctly (193 ms)
    √ complete watch assembly and NFT minting workflow (215 ms)
    √ handles assembly errors gracefully (53 ms)
    √ handles NFT minting errors (52 ms)
    √ component management (add/remove) works correctly (86 ms)
    √ loading states are handled correctly (29 ms)
    √ reset functionality works correctly (68 ms)
    √ button is disabled when all components are empty (31 ms)
    √ displays error alert when API returns error (34 ms)

Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        14.62 s
Ran all test suites matching /WatchManager.integration.test.jsx/i.
```

Figure 6.2.48 Chai and Mocha Frontend Testing 1

```
C:\Users\USER\Desktop\Testing\final\FYP\frontend>npm test useWatchAPI.test.js

> my-app@0.1.0 test
> react-scripts test useWatchAPI.test.js
PASS src/hooks/__tests__/useWatchAPI.test.js
  useWatchAPI Hook
    √ assembleWatch succeeds with valid components (21 ms)
    √ assembleWatch fails with empty components (3 ms)
    √ mintNFT succeeds for valid watch (4 ms)
    √ mintNFT handles blockchain errors (3 ms)
    √ getWatchDetails returns complete watch information (5 ms)
    √ loading state works correctly (103 ms)
    √ clearError function works (3 ms)

Test Suites: 1 passed, 1 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        3.719 s, estimated 10 s
Ran all test suites matching /useWatchAPI.test.js/i.
```

Figure 6.2.49 Chai and Mocha Frontend Testing 2

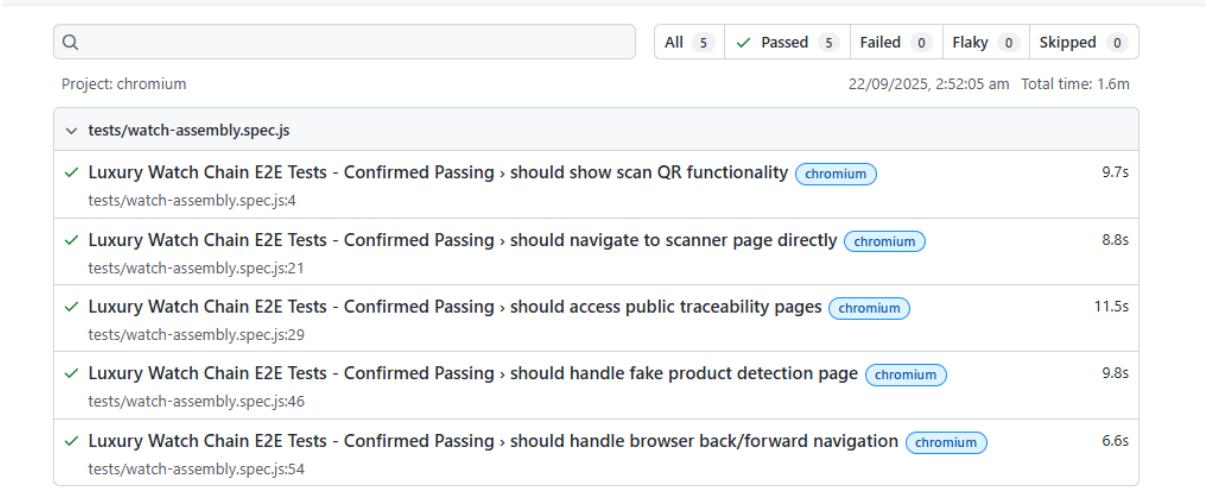

Figure 6.2.50 Playwright Frontend Testing

6.3 Project Challenges

First of all, the primary technical challenge encountered was the complex integration between multiple decentralized technologies, particularly the IPFS (InterPlanetary File System) storage mechanism and blockchain smart contracts. The system demonstrated considerable difficulty in maintaining consistent connectivity with Pinata IPFS services, requiring the implementation of sophisticated fallback mechanisms that utilize local storage when IPFS upload operations fail. This challenge was compounded by the need to maintain data integrity across multiple storage layers, including PostgreSQL databases, local file systems, and distributed IPFS networks, creating potential points of failure that required extensive error handling and recovery procedures.

Besides, the blockchain integration presented substantial challenges related to transaction management and gas fee optimization. The system frequently encountered issues with Ethereum network connectivity, particularly regarding MetaMask wallet integration and smart contract interactions. Gas fee volatility posed significant usability concerns, as users experienced transaction failures due to insufficient gas limits, necessitating the implementation of dynamic gas estimation mechanisms and comprehensive error messaging systems. Additionally, the asynchronous nature of blockchain transactions created complex state management challenges, requiring sophisticated loading states and progress indicators to maintain user experience while transactions were being mined and confirmed on the network. Performance optimization emerged as a critical challenge across multiple system components, particularly in the AI analysis integration using Google's Gemini API for luxury watch model identification and market price analysis. The system experienced significant latency issues when processing high-resolution watch images, generating NFT metadata, and performing real-time price trend analysis across twelve-month periods. The Material-UI frontend components required extensive optimization to reduce animation overhead and improve rendering performance, leading to the removal of complex gradient animations and the implementation of simplified styling approaches. Database query performance also presented challenges when handling complex joins between watches, components, raw materials, and AI analysis tables, requiring careful indexing strategies and query optimization.

Moreover, the testing infrastructure using Playwright introduced additional complexity challenges, particularly in end-to-end testing scenarios that required coordination between multiple technology stacks including React frontend, Node.js backend, PostgreSQL database, and Ethereum blockchain interactions. Test flakiness emerged as a significant issue due to the asynchronous nature of blockchain transactions and IPFS operations, requiring the implementation of sophisticated retry mechanisms and timeout handling. Furthermore, the multi-browser testing approach using Chromium introduced compatibility challenges that necessitated extensive cross-platform validation and error handling to ensure consistent functionality across different execution environments.

6.4 Objectives Evaluation

The project was designed to achieve several key objectives, and the results reflect significant progress towards these goals.

Objective 1: Implement Blockchain-Based Traceability for Luxury Watches

Regarding the first objective of the project, the system successfully establishes a robust infrastructure utilizing Ethereum smart contracts and ERC-721 NFT standards to create immutable records throughout the supply chain lifecycle. The implementation demonstrates comprehensive traceability capabilities through the integration of raw material registration, component manufacturing, quality certification, watch assembly, and ownership transfer processes, all recorded on the blockchain with corresponding NFT metadata stored on IPFS.

Objective 2: Utilize Blockchain for Authenticity Verification Using NFTs

The project achieves substantial success through the implementation of unique digital certificates that encapsulate comprehensive product information, manufacturing data, and ownership history within standardized NFT metadata structures. The system successfully integrates AI-powered watch model identification using Google's Gemini API, providing market value analysis and trend predictions that enhance the authenticity verification process beyond traditional methods. The QR code scanning infrastructure enables stakeholders to verify watch authenticity in real-time by accessing blockchain-stored information and associated NFT metadata.

Objective 3: Implement Blockchain to Enhance Transparency and Collaboration in SCM

The project demonstrates significant progress through the development of a comprehensive multi-stakeholder platform that provides role-based dashboards for suppliers, manufacturers, certifiers, assemblers, distributors, retailers, and consumers. The system successfully implements smart contract automation for critical processes including component certification, ownership transfers, and payment verification, thereby reducing manual intervention and increasing operational efficiency. Real-time tracking capabilities and comprehensive audit trails provide stakeholders with unprecedented visibility into product lifecycle management.

6.5 Concluding Remark

In conclusion, this project successfully integrated blockchain technology and Non-Fungible Token (NFT) frameworks into luxury watch supply chain management to enhance traceability, authenticity verification, and stakeholder transparency. Despite encountering significant technical challenges, the project comprehensively achieved its fundamental objectives. The implementation of ERC-721 smart contracts coupled with IPFS-based metadata storage demonstrated substantial potential for revolutionizing luxury goods authentication and supply chain provenance tracking. The system successfully established end-to-end traceability from raw material sourcing through component manufacturing, quality certification, watch assembly, distribution, and final consumer purchase, while providing real-time verification capabilities through QR code integration and AI-powered watch model identification. The multi-stakeholder platform encompassing suppliers, manufacturers, certifiers, assemblers, distributors, retailers, and consumers validated the viability of blockchain-based collaborative frameworks in complex supply chain ecosystem. The findings of this research demonstrate that the convergence of blockchain immutability, NFT-based digital certificates, and AI-enhanced authentication mechanisms offers a robust foundation for combating counterfeiting in luxury goods markets while establishing unprecedented levels of supply chain transparency.

Chapter 7

7.1 Conclusion

In conclusion, this project presents a pioneering implementation of blockchain-enabled supply chain management specifically tailored for luxury watch authentication and traceability, demonstrating significant advancement in the convergence of distributed ledger technology, non-fungible tokens, and artificial intelligence for combating counterfeiting in high-value goods markets. The project successfully addresses three critical objectives through the development of a sophisticated multi-stakeholder ecosystem that leverages Ethereum smart contracts, ERC-721 NFT standards, and IPFS decentralized storage to create immutable provenance records from raw material sourcing through final consumer ownership. The implementation of the LuxuryWatchNFT smart contract, coupled with a comprehensive backend infrastructure utilizing PostgreSQL databases and Node.js API services, establishes a robust framework capable of supporting complex supply chain operations while maintaining data integrity and security across seven distinct stakeholder roles including suppliers, manufacturers, certifiers, assemblers, distributors, retailers, and consumers.

Besides, the technical architecture demonstrates exceptional innovation through its integration of artificial intelligence capabilities using Google's Gemini API for automated watch model identification and market value analysis, providing stakeholders with unprecedented insights into luxury goods authentication and investment potential. The system's ability to generate unique digital certificates that encapsulate comprehensive product information, manufacturing data, and ownership history within standardized NFT metadata structures represents a significant advancement over traditional authentication methods, while the implementation of QR code scanning infrastructure enables real-time verification capabilities that enhance both consumer confidence and supply chain transparency. The project achieved comprehensive end-to-end functionality with successful testing validation across smart contract operations, API endpoints, and frontend user interfaces.

Moreover, the project contributions extend beyond mere technical implementation to establish a foundational framework that addresses fundamental challenges in luxury goods markets including counterfeiting, supply chain opacity, and ownership verification through the creation of an immutable digital thread that tracks each product's complete lifecycle. The findings demonstrate that the strategic integration of blockchain immutability, NFT-based digital certificates, and AI-enhanced authentication mechanisms provides a scalable solution

for enhancing transparency, trust, and value proposition in high-value merchandise transactions. Furthermore, the project's comprehensive testing methodology, encompassing smart contract validation, backend API performance assessment, and multi-browser frontend testing, establishes rigorous quality assurance standards suitable for enterprise deployment while providing valuable insights for future research in decentralized authentication systems. This work represents a significant academic contribution to the intersection of blockchain technology and supply chain management, demonstrating practical application of theoretical concepts in real-world contexts while establishing a blueprint for transforming luxury goods authentication practices across multiple industries beyond horology, ultimately advancing the field toward more transparent, secure, and efficient supply chain management paradigms in the digital economy.

7.2 Recommendations

1. Infrastructure Optimization and Scalability Enhancement

The primary recommendation focuses on addressing the significant infrastructure challenges identified during system evaluation, particularly the IPFS connectivity instabilities and Ethereum mainnet scalability limitations. Future development should prioritize the implementation of Layer 2 scaling solutions such as Polygon or Arbitrum to reduce transaction costs and improve processing speeds, while establishing redundant IPFS infrastructure through multiple pinning services including Pinata, Infura, and Fleek to ensure consistent metadata availability. The current system's reliance on local fallback mechanisms, while functional, compromises the decentralized architecture's integrity and should be supplemented with distributed storage networks and Content Delivery Networks (CDNs) for enhanced reliability. Additionally, implementing dynamic gas fee estimation algorithms and transaction batching mechanisms would significantly improve user experience by reducing both costs and transaction confirmation times. The integration of enterprise-grade blockchain infrastructure providers and the establishment of dedicated nodes for high-volume operations would ensure the system's capability to handle commercial-scale deployments while maintaining the security and immutability benefits of blockchain technology.

2. Enhanced Testing and Quality Assurance Framework

The second recommendation addresses the critical need for a more robust testing infrastructure to ensure system reliability and stability across all components. The current Playwright-based testing framework, while comprehensive, experiences significant flakiness due to the asynchronous nature of blockchain transactions and multi-system dependencies, necessitating the development of specialized testing methodologies for blockchain applications. Future iterations should implement comprehensive mocking frameworks for blockchain interactions during unit testing, establish dedicated testnets with controlled environments for integration testing, and develop automated contract auditing tools to ensure smart contract security. The testing suite should incorporate stress testing scenarios that simulate high-volume transactions, network congestion, and partial system failures to validate system resilience under adverse conditions. Furthermore, implementing continuous integration and continuous deployment (CI/CD) pipelines with automated smart contract deployment, database migration testing, and cross-browser compatibility validation would significantly enhance the system's reliability and reduce the risk of production deployment failures.

3. User Experience and Interface Optimization

The third recommendation emphasizes the imperative need for comprehensive user experience enhancement to facilitate broader adoption among stakeholders with varying technical expertise. The current Material-UI implementation, while functional, suffers from performance bottlenecks and complex navigation structures that may impede user adoption, particularly among traditional luxury goods industry participants who may lack blockchain familiarity. Future development should prioritize the implementation of progressive web application (PWA) capabilities to enable offline functionality and mobile optimization, while developing intuitive guided workflows that abstract complex blockchain operations behind simplified interfaces. The integration of comprehensive onboarding tutorials, contextual help systems, and multi-language support would significantly improve accessibility for global stakeholders. Additionally, implementing adaptive user interfaces that adjust complexity based on user roles and experience levels, coupled with real-time transaction status indicators and simplified wallet integration processes, would enhance user confidence and reduce the learning curve associated with blockchain technology adoption in traditional supply chain environments.

4. Advanced Security and Authentication Mechanisms

The fourth recommendation focuses on strengthening the security framework beyond the inherent blockchain protections to address enterprise-level security requirements and regulatory compliance standards. While the current system leverages blockchain's immutability and cryptographic security, additional security layers are essential for handling sensitive commercial data and protecting against sophisticated attack vectors. Future implementations should incorporate multi-factor authentication (MFA) systems, hardware security module (HSM) integration for key management, and zero-knowledge proof mechanisms for privacy-preserving verification of sensitive supply chain information. The development of comprehensive audit logging systems, real-time threat detection algorithms, and automated security scanning for smart contracts would provide additional protection against vulnerabilities and malicious activities. Furthermore, implementing role-based access control (RBAC) with granular permissions, encrypted communication channels between all system components, and regular security assessments following industry standards such as NIST Cybersecurity Framework would ensure the system meets enterprise security requirements and regulatory compliance mandates for handling luxury goods authenticity data.

5. Integration with Industry Standards and Regulatory Compliance

The fifth recommendation addresses the critical need for alignment with existing luxury goods industry standards and regulatory frameworks to facilitate real-world adoption and legal

compliance. The current system, while technically robust, operates independently of established certification bodies, international trade regulations, and industry-specific authentication standards that govern luxury watch markets globally. Future development should prioritize integration with recognized certification authorities such as the Swiss Official Chronometer Testing Institute (COSC), Fondation de la Haute Horlogerie (FHH), and relevant ISO standards for luxury goods authentication. The system should incorporate compliance mechanisms for international trade regulations including customs documentation, anti-money laundering (AML) requirements, and consumer protection laws across different jurisdictions. Additionally, establishing partnerships with existing luxury goods authentication services, insurance providers, and established auction houses would create a comprehensive ecosystem that bridges traditional and blockchain-based verification methods. The implementation of standardized data exchange formats compatible with existing Enterprise Resource Planning (ERP) systems and Supply Chain Management (SCM) platforms would facilitate seamless integration with established business processes, thereby reducing barriers to adoption and enabling the system to complement rather than replace existing industry infrastructure.

REFERENCES

[1] GeeksForGeeks, "Types of Blockchain," GeeksforGeeks, Apr. 27, 2022. https://www.geeksforgeeks.org/types-of-blockchain/

[2] D. Adam, D. B. Matellini, and A. Kaparaki, "Benefits for the bunker industry in adopting blockchain technology for dispute resolution," Blockchain: Research and Applications, vol. 4, no. 2, p. 100128, Jan. 2023, doi: https://doi.org/10.1016/j.bcra.2023.100128.

[3] R. Reddy, "Proof of Work vs. Proof of Stake: Why Their Differences Matter," Global X ETFs, Oct. 05, 2022. https://globalxetfs.com.br/en/proof-of-work-vs-proof-of-stake-why-their-differences-matter/ (accessed Apr. 30, 2025).

[4] A. Joury, "Proof of Stake Versus Proof of Work: Understanding the Differences | Built In," builtin.com, Mar. 13, 2025. https://builtin.com/blockchain/proof-of-stake-versus-proof-of-work

[5] N. Kumar, K. Kumar, A. Aeron, and F. Verre, "Blockchain Technology in Supply Chain Management: Innovations, Applications, and Challenges," *Telematics and Informatics Reports*, vol. 18, no. 18, p. 100204, Apr. 2025, doi: https://doi.org/10.1016/j.teler.2025.100204.

[6] S. Saberi, M. Kouhizadeh, J. Sarkis, and L. Shen, "Blockchain Technology and Its Relationships to Sustainable Supply Chain Management," *International Journal of Production Research*, vol. 57, no. 7, pp. 2117–2135, Oct. 2019, doi: https://doi.org/10.1080/00207543.2018.1533261.

[7] GeeksforGeeks, "Smart Contracts in Blockchain," *GeeksforGeeks*, Jan. 07, 2019. https://www.geeksforgeeks.org/smart-contracts-in-blockchain/

[8] S. E. Chang, Y.-C. Chen, and M.-F. Lu, "Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process," *Technological Forecasting and Social Change*, vol. 144, no. 1, pp. 1–11, Jul. 2019, doi: https://doi.org/10.1016/j.techfore.2019.03.015.

[9] V. Sugumaran and S. Jafar Ali Ibrahim, "Rough set based on least dissimilarity normalized index for handling uncertainty during E-learners learning pattern recognition," *International Journal of Intelligent Networks*, vol. 3, pp. 133–137, 2022, doi: https://doi.org/10.1016/j.ijin.2022.09.001.

[10] A. (Jasmine) Chang, N. El-Rayes, and J. Shi, "Blockchain Technology for Supply Chain Management: A Comprehensive Review," *FinTech*, vol. 1, no. 2, pp. 191–205, Jun. 2022, doi: https://doi.org/10.3390/fintech1020015.

[11] G. Rai Goyal and S. Vadhera, "Solution to uncertainty of renewable energy sources and peak hour demand in smart grid system," *Measurement: Sensors*, vol. 33, p. 101129, Mar. 2024, doi: https://doi.org/10.1016/j.measen.2024.101129.

[12] A. I. Sanka and R. C. C. Cheung, "A systematic review of blockchain scalability: Issues, solutions, analysis and future research," *Journal of Network and Computer Applications*, vol. 195, no. 1, p. 103232, Dec. 2021, doi: https://doi.org/10.1016/j.jnca.2021.103232.

[13] Dario, "Polygon Review: Beginner's Guide to POL," *99Bitcoins*, Apr. 18, 2025. https://99bitcoins.com/cryptocurrency/polygon-crypto-review/ (accessed May 01, 2025).

[14] M. K. Lim, Y. Li, C. Wang, and M.-L. Tseng, "A literature review of blockchain technology applications in supply chains: A comprehensive analysis of themes, methodologies and industries," *Computers & Industrial Engineering*, vol. 154, no. 107133, p. 107133, Apr. 2021, doi: https://doi.org/10.1016/j.cie.2021.107133.

[15] D. Shree, Muskan, N. Srinivasa, R. Pokala, and P. Harish, "Blockchain-Powered Product Authentication and Anti-Counterfeiting System," *Blockchain-Powered Product Authentication and Anti-Counterfeiting System*, vol. 13, no. 1, pp. 2320–2882, 2025, Accessed: May 01, 2025. [Online]. Available: https://ijcrt.org/papers/IJCRT2501546.pdf

[16] H. Han, R. K. Shiwakoti, and W. Chen, "Unlocking enterprise blockchain adoption: A R3 Corda case study," *Journal of General Management*, vol. 4, Oct. 2024, doi: https://doi.org/10.1177/03063070241292701.

[17] T. Sharrow, "Is R3 Dead? The Current Status of R3 and Its Future - SoftHandTech," *SoftHandTech*, Mar. 12, 2025. https://softhandtech.com/has-r3-been-discontinued/ (accessed May 01, 2025).

[18] Sutibu Kanemochi, "Ethicoin (ETHIC+) on Binance Smart Chain: A Scientific Analysis," *ResearchGate*, Aug. 06, 2024. https://www.researchgate.net/publication/3828

88189_Ethicoin_ETHIC_on_Binance_Smart_Chain_A_Scientific_Analysis

[19] "An Introduction to BNB Smart Chain (BSC)," *Binance Academy*, Feb. 14, 2024. https://academy.binance.com/en/articles/an-introduction-to-bnb-smart-chain-bsc

[20] N. Willing, “What Is Binance Smart Chain? BSC Explained In Detail,” *Techopedia*, Mar. 28, 2024. https://www.techopedia.com/definition/binance-smart-chain-bsc (accessed May 01, 2025).

[21] G. Kaur, “Polygon blockchain explained: A beginner’s guide to MATIC,” *Cointelegraph*, 2023. https://cointelegraph.com/learn/articles/polygon-blockchain-explained-a-beginners-guide-to-matic

[22] OpenSea, “Polygon (MATIC) Explained: Complete Guide to Ethereum’s Scaling Solution,” *Opensea.io*, 2022. https://opensea.io/learn/blockchain/what-is-polygon (accessed May 01, 2025).

[23] TheKnowledgeAcademy, “Polygon Blockchain: Beginner’s Guide to MATIC,” *Theknowledgeacademy.com*, 2025. https://www.theknowledgeacademy.com/blog/polygon-blockchain/ (accessed May 01, 2025).

[24] Janne Kaisto, Teemu Juutilainen, and Joona Kauranen, “Non-fungible tokens, tokenization, and ownership,” *Computer Law & Security Review*, vol. 54, no. 2, pp. 105996–105996, Sep. 2024, doi: https://doi.org/10.1016/j.clsr.2024.105996.

[25] A. Cantu Moreno, “NFT as a proof of Digital Ownership-reward system integrated to a Secure Distributed Computing Blockchain Framework,” *uis.brage.unit.no*, 2022. https://uis.brage.unit.no/uis-xmlui/handle/11250/3032538

[26] V. Nadkarni, “Introduction to tokens and the Corda token SDK - Vardan Nadkarni - Medium,” *Medium*, Dec. 28, 2021. https://medium.com/@vardan.nadkarni/introduction-to-tokens-and-the-corda-token-sdk-4fec85c0c045 (accessed May 01, 2025).

[27] H. Sehgal, “SoluLab - Blockchain Development Company,” *Blockchain Technology, Mobility, AI and IoT Development Company USA, Canada*, Jul. 18, 2023. http://solulab.com/bsc-nft-marketplace-development/ (accessed May 01, 2025).

[28] Siddhant Kejriwal, “Polygon Review 2025: Updates You Don’t Want to Miss!,” *Coin Bureau*, Apr. 18, 2023. https://coinbureau.com/education/polygon-review/ (accessed May 01, 2025).

[29] N. Kumar, “Ethereum Gas Fees: Challenges and Solutions in 2024,” *Analytics Insight*, Jan. 27, 2024. https://www.analyticsinsight.net/blockchain/ethereum-gas-fees-challenges-and-solutions-in-2024

[30] D. Ravi, S. Ramachandran, R. Vignesh, V. R. Falmari, and M. Brindh, “Privacy preserving transparent supply chain management through Hyperledger Fabric,” *Blockchain: Research and Applications*, vol. 3, no. 2, p. 100072, Mar. 2022, doi: https://doi.org/10.1016/j.bcra.2022.100072.

[31] Aslan Alwi, Nugroho Adi Sasongko, None Suprapto, Yaya Suryana, and Hendro Subagyo, “Blockchain and big data integration design for traceability and carbon footprint management in the fishery supply chain,” *Egyptian Informatics Journal/Egyptian Informatics Journal* , vol. 26, no. 1, pp. 100481–100481, Jun. 2024, doi: https://doi.org/10.1016/j.eij.2024.100481.

[32] A. Rizzardi, S. Sicari, J. F. Cevallos, and A. Coen-Porisini, “IoT-driven blockchain to manage the healthcare supply chain and protect medical records,” *Future Generation Computer Systems*, vol. 161, no. 2, Jul. 2024, doi: https://doi.org/10.1016/j.future.2024.07.039.

[33] L. Stopfer, A. Kaulen, and T. Purfürst, “Potential of blockchain technology in wood supply chains,” *Computers and Electronics in Agriculture*, vol. 216, no. 6, p. 108496, Jan. 2024, doi: https://doi.org/10.1016/j.compag.2023.108496.

[34] J. Liu, P. Jiang, and J. Zhang, “A blockchain-enabled and event-driven tracking framework for SMEs to improve cooperation transparency in manufacturing supply chain,” *Computers & Industrial Engineering*, vol. 191, no. 9, pp. 110150–110150, May 2024, doi: https://doi.org/10.1016/j.cie.2024.110150.

[35] L. Jiang, A. Yao, W. Li, and Q. Wei, “Blockchain technology empowers the cross-border dual-channel supply chain: Introduction strategy, tax differences, optimal decisions,” *Computers & Industrial Engineering*, vol. 195, no. 2, pp. 110431–110431, Aug. 2024, doi: https://doi.org/10.1016/j.cie.2024.110431.

[36] Y. Kang, P. Dong, Y. Ju, and T. Zhang, “Differential game theoretic analysis of the blockchain technology investment and carbon reduction strategy in digital supply chain with government intervention,” *Computers & Industrial Engineering*, vol. 189, no. 2, p. 109953, Mar. 2024, doi: https://doi.org/10.1016/j.cie.2024.109953.

[37] M. W. Akram, N. Akram, F. Shahzad, K. U. Rehman, and S. Andleeb, “Blockchain technology in a crisis: Advantages, challenges, and lessons learned for enhancing food supply chains during the COVID-19 pandemic,” *Journal of Cleaner Production*, vol. 434, p. 140034, Jan. 2024, doi: https://doi.org/10.1016/j.jclepro.2023.140034.

[38] Geeksforgeeks, “What is Secure Software Development Life Cycle (SSDLC)?,” *GeeksforGeeks*, Jan. 03, 2024. https://www.geeksforgeeks.org/what-is-secure-software-development-life-cycle-ssdlc/

[39] F. Dietrich, L. Louw, and D. Palm, “Blockchain-Based Traceability Architecture for Mapping Object-Related Supply Chain Events,” *Sensors*, vol. 23, no. 3, p. 1410, Jan. 2023, doi: https://doi.org/10.3390/s23031410.

[40] A. Kumare, “AI based NFT Minting using Blockchain,” *ISJEM Journal*, May 02, 2025. https://isjem.com/download/ai-based-nft-minting-using-blockchain/ (accessed Sep. 20, 2025).

[41] J. Davies, H. Sharifi, A. Lyons, R. Forster, and O. K. S. M. Elsayed, “Non-fungible tokens: The missing ingredient for sustainable supply chains in the metaverse age?,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 182, no. 5, p. 103412, Feb. 2024, doi: https://doi.org/10.1016/j.tre.2024.103412.

POSTER


UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
EFFICIENT TRACKING OPERATION FOR SUPPLY CHAIN MANAGEMENT BASED ON BLOCKCHAIN TECHNOLOGY
Introduction
This research presents an innovative blockchain-enabled supply chain management system that integrates Ethereum smart contracts, NFT tokenization, and AI-powered analysis to establish immutable provenance tracking and combat counterfeiting in the luxury watch industry through comprehensive multi-stakeholder authentication workflows.
Objectives
Implement Blockchain-Based Traceability for Luxury Watches
Utilize Blockchain for Authenticity Verification Using NFTs
Implement Blockchain to Enhance Transparency and Collaboration in SCM
Student: Lim Wei Ching
Sepervisor: Ts Dr. Ooi Joo On