UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

AquaTrace: Secure Federated Identifiers for Product Tracing using Blockchain

Henning Hageli

INF-3981 Master’s Thesis in Computer Science – June 2024

Supervisors

Main supervisor:	Håvard Dagenborg	UiT The Arctic University of Norway, Faculty of Science and Technology, Department of Computer Science

Acknowledgements

I wish to extend my appreciation to Professor Håvard Dagenborg for his guidance, support, and insightful contributions, which have played an important part in completing this thesis. I also acknowledge the assistance of ChatGPT for helping with the grammar and structuring of sentences to enhance the flow and readability of this thesis. I also extend my thanks to my fellow student, Sivert Steinholt, for his collaborative efforts and discussions about our respective theses. Furthermore, I express my appreciation to my friends at the Cyber Security Group (CSG) lab for their companionship, which has made this journey more fulfilling and motivating.

Abstract

This thesis proposes a secure and resilient system for generating and managing unique Identification (ID) number series for tracing food products within the Norwegian fishing industry, without relying on a central authority. Given the context of mutual mistrust among stakeholders and the threat of hostile entities, this project proposes a blockchain-based solution to ensure the uniqueness and security of each ID in a decentralized environment. The core of this research involves designing and implementing a smart contract on an Ethereum Virtual Machine (EVM)-compatible blockchain to manage the ID series efficiently. The thesis begins with an analysis of current industry standards and reviews existing blockchain applications for product tracing, forming the basis for the proposed ID scheme and data models. The practical aspect includes developing a smart contract for ID generation and management, highlighting the application's ability to prevent ID collisions and ensure secure, verifiable IDs. The thesis also includes the design and implementation of a Web3 application, serving as an interface for users to interact with the blockchain system. This application facilitates the generation and management of ID series, and extends to include product tracing functionalities. This research aims to demonstrate the feasibility and effectiveness of using blockchain technology and smart contracts to enhance the traceability and security of food products in the fishing industry.

Contents

List of Figures

List of Tables

List of Abbreviations

ABI Application Binary Interface

API Application Programming Interface

CPU Central Processing Unit

CSG Cyber Security Group

DApp Decentralized Application

DDR4 Double Data Rate 4

EVM Ethereum Virtual Machine

GB Gigabytes

GDPR General Data Protection Regulation

GPU Grahpics Processing Unit

ID Identification

IOT Internet of Things

IPFS Interplanetary File System

JSON JavaScript Object Notation

ML Machine Learning

NVMe Non-Volatile Memory Express

OS Operating System

P2P	Peer-to-Peer
PBFT	Practical Byzantine Fault Tolerance
PII	Personally Identifiable Information
POS	Proof-of-Stake
POW	Proof-of-Work
PSC	Pharmaceutical Supply Chains
RAM	Random Access Memory
RFID	Radio Frequency Identification
RPC	Remote Procedure Call
SED	Self-Encrypting Device
TPS	Transactions Per Second
UI	User-Interface
UiT	University of Tromsø

1 Introduction

The ocean provides 15.3% of the global production of animal proteins, highlighting its importance as a crucial food source worldwide [3]. In Norway, the fishing industry is a significant economic contributor, with seafood exports valued at 43.0 billion NOK and aquaculture exports reaching 128.7 billion NOK in 2023 [10]. Salmon, in particular, stands out as the largest export product, underscoring Norway's prominent position in the global seafood market.

Despite its economic importance, the fishing industry faces severe challenges due to various illicit activities such as illegal fishing, document fraud, and money laundering [48]. These activities not only threaten the economic stability of the industry but also undermine efforts towards sustainability and environmental conservation. Addressing these issues requires innovative solutions that can enhance transparency, traceability, and trust among all stakeholders involved in the seafood supply chain.

One promising approach to combating illegal fishing and related crimes is the implementation of comprehensive product tracing systems. By tracking seafood products from the point of capture to the consumer's table, both control authorities and consumers can verify the authenticity and origin of the products. Blockchain technology, with its immutable ledger and decentralized nature, offers a robust foundation for such traceability systems. This technology can facilitate transparency and trust among various stakeholders in the supply chain. However, effective product tracing also requires physical marking of products with unique digital identification numbers to ensure accurate and reliable tracking throughout the supply chain.

This thesis explores the development of a secure and resilient method for creating and managing unique identification series for tracing food products within the fishing industry. By leveraging blockchain technology, specifically through the implementation of EVM-compatible smart contracts, this research aims to address the challenges of mutual mistrust among organizations and the threat of hostile entities. The project includes the development of a proof-of-concept system to demonstrate the practical application of blockchain for product tracing, ultimately contributing to enhanced transparency and accountability in the seafood supply chain.

1.1 Problem Definition

This project aims to develop a secure and resilient method for creating and managing unique ID number series for tracing food products within the fishing industry without relying on a central authority. The project should address the challenge of mutual mistrust among organizations and the threat of hostile entities aiming to exploit the system. Our thesis is that:

blockchain-based smart contracts can effectively be used to manage unique ID series for product tracing.

We approach our thesis by designing and implementing an EVM-compatible smart contract for the generation and management of these IDs, leveraging existing blockchain-based product tracing methodologies. Additionally, the project encompasses the development of a Web3 application to facilitate user interaction with the blockchain for ID management and product tracing.

1.2 Scope and Limitations

The primary focus of this thesis is on the development and evaluation of a system for securely generating and managing product IDs using blockchain technology. The objectives include implementing a blockchain-based solution to generate unique, tamper-proof product IDs that can be tracked throughout the supply chain. Additionally, a functional proof-of-concept system will be developed to demonstrate the practical application of the blockchain solution. This system will enable users to interact with the blockchain to manage product IDs, ensuring traceability and data integrity.

Several assumptions and constraints define the boundaries of this thesis. It assumes the existence of a private consortium blockchain running on the Avalanche platform, restricted to members who have been securely onboarded through a dedicated onboarding service. It is assumed that the Norwegian Ministry for Trade, Industry, and Fisheries hosts the consortium blockchain, the onboarding service, and the application itself. While the ministry provides the infrastructure, it does not exert control over the blockchain's operations, maintaining the decentralized and autonomous nature of the network.

While the system is designed with security and integrity in mind, detailed exploration of regulatory compliance (e.g., General Data Protection Regulation (GDPR)) is beyond the scope of this thesis. The focus is primarily on the technical implementation and evaluation of the blockchain solution. The proof-of-concept system will be tested in a controlled environment using a testnet, and the results may differ when deployed in a real-world, production environment due to variables such as network conditions and user behavior that are not replicated in the testnet. The security analysis assumes a baseline level of network integrity and does not account for all possible attack vectors, meaning real-world deployments might encounter additional security challenges not fully addressed in this thesis.

Extensive user feedback and usability testing are limited, with the usability evaluation primarily theoretical, and based on own tests, rather than comprehensive user studies. Lastly, the system is built using current blockchain technologies and standards, which may evolve, necessitating adjustments to the implementation presented in this thesis.

By outlining these scopes and limitations, this thesis aims to provide a clear understanding of the project's objectives, the boundaries within which it operates, and the assumptions underlying the research and development process.

1.3 Context

This thesis will be completed in the context of the CSG group at University of Tromsø (UiT). The CSG group is a research group that focuses on key challenges in the realm of large-scale information access applications. Its main goal is to develop effective strategies for designing and implementing systems that can manage and access vast amounts of information efficiently and reliably. This involves breaking down complex applications into smaller, cooperating modules, enhancing the way these modules interact with each other and with users, and determining the best deployment strategies. Additionally, the CSG group emphasizes the importance of maintaining integrity, security, and fault-tolerance within these systems. To achieve these objectives, the CSG group primarily adopts an experimental approach, building prototype middleware systems to address specific research questions.

The CSG group has had considerable focus on how blockchain technology can be used within the fishing industry, especially for product tracing [15]. This thesis will continue working on this use case with the same goal in mind, mitigating crimes within the industry [29]. The thesis is part of an ongoing research interest at the CSG group. Previous works by the research group include studies on low bandwidth communication in remote sensing for fishing vessels [32], transaction latency prediction using deep learning in blockchain systems [46], transaction fees in blockchain systems [45], the development of datasets like Njord [30] for fishing trawler surveillance, and blockchain-based seafood industry applications such as the Áika system [1]. These contributions provide a solid foundation and context for the current research, focusing on enhancing transparency and accountability in the seafood supply chain through secure ID generation and management.

1.4 Method

In their 1989 report, the Task Force on the Core of Computer Science proposed a structured approach to outline the discipline of computing into three major paradigms: theory, abstraction, and design. These paradigms offer a comprehensive framework for the advancement of computer science, each following a distinct process to contribute uniquely to the field [9]. Below is a detailed explanation of each paradigm and its respective methodology.

The theory paradigm is deeply rooted in mathematical principles and follows a rigorous four-step process aimed at developing a coherent and valid theoretical framework

1. **Definition**: This initial step involves characterizing the objects of study to ensure clarity and consistency in the theoretical exploration.

2. **Theorem**: Based on the defined objects, hypotheses are formulated about the potential relationships among them.

3. **Proof**: Evaluating the theorems by proving or disproving them, testing the validity of the proposed hypotheses.

4. **Interpretation**: Results obtained from the proof stage are interpreted.

Rooted in the experimental scientific method, the abstraction paradigm follows a systematic process to model and investigate phenomena:

1. **Hypothesis**: A hypothesis is formulated about the phenomenon under investigation, proposing a tentative explanation or prediction that the study aims to test.

2. **Model and Predict**: A model is constructed based on the initial hypothesis, which is used to predict the behavior or outcomes of the phenomenon.

3. **Experiment and Collect data**: An experiment is designed to test the model's predictions, and data is collected to capture the outcomes of the experiment.

4. **Analyze**: The collected data is analyzed to evaluate the model's accuracy and the validity of the hypothesis.

Drawing on engineering principles, the design paradigm encompasses a structured approach to developing systems that address specific problems:

1. **Requirements**: The process begins with the identification of the system's requirements, detailing the needs and constraints that the system aims to fulfill.

2. **Specifications**: Detailed specifications are outlined, which guide the design and functionality of the system.

3. **Design and Implementation**: Based on the specifications, the system is designed and then implemented.

4. **Testing**: Testing is conducted to ensure that the system operates as intended, meeting the initial requirements and specifications.

AquaTrace falls within the design paradigm. Based on a problem definition, we will design, implement, and evaluate a proof-of-concept system defined by the requirements and specifications we have stated.

1.5 Outline

This section outlines the structure of the thesis.

Section 2 - Background: Provides the necessary context and background information on blockchain technology and other relevant technical details that readers need to understand the thesis

Section 3 - Requirements: Outlines the functional and non-functional requirements for the AquaTrace system, detailing the system's objectives and performance criteria.

Section 4 - Design and Implementation: Describes the architecture, design decisions, and implementation details of the AquaTrace system.

Section 5 - Evaluation: Assesses the performance of AquaTrace against the defined requirements, presenting the results of various tests and analyses to determine how well the system meets its goals.

Section 6 - Discussion: Discusses the evaluation results with regards to non-functional requirements, the effectiveness of the design and implementation, and areas for potential improvement. It also covers the rationale behind the choice of blockchain.

Section 7 - Conclusion: Summarizes the key findings, reflects on the overall success of the project, and suggests directions for future research and development.

2 Background

This chapter provides the knowledge necessary for understanding the thesis that follows. It begins with an explanation of blockchain technology, including its structure, types, and consensus protocols, with a specific focus on Proof-of-Work (POW), Proof-of-Stake (POS), and the Avalanche consensus mechanism. The chapter also explains the EVM and smart contracts, highlighting their role in enabling Decentralized Applications (DApps). Additionally, it covers the Avalanche platform, Web3 principles, the use of Docker for containerization, the importance of cryptographic hashing functions, and key terminology relevant to the thesis. By detailing these concepts, the reader will gain sufficient theoretical knowledge to comprehend the subsequent chapters.

2.1 Blockchain

A blockchain is a distributed database or ledger that is shared among the nodes of a Peer-to-Peer (P2P) computer network [21]. Its most distinctive feature, which sets it apart from a traditional database, is its structure: it organizes data into blocks, which are then chained together using cryptographic principles. This structure provides several unique properties and benefits that are crucial for various applications, especially in areas requiring high security and trust, such as finance, supply chain management, and identity verification [16].

The core of blockchain lies in its decentralization. Unlike conventional databases governed by a central authority, blockchain distributes its data across a network of computers, eliminating single points of failure and diminishing the control of any single entity over the entire system. This decentralization is crucial, not only for enhancing security but also for contributing to a new level of transparency in digital transactions. Every transaction on a blockchain is visible to all participants and is immutable, meaning once it is recorded, it cannot be altered or deleted. Figure 2.1 shows how blocks within a blockchain are cryptographically chained together.

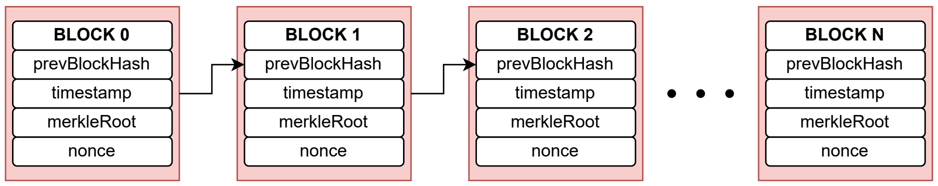


Figure 2.1: Visualization of how blocks are linked together in a blockchain

2.1.1 Types of Blockchain

Each type of blockchain serves different purposes and offers a unique set of advantages and limitations. The choice between them depends on the specific requirements of the application, including considerations of security, privacy, transparency, and control.

Public Blockchain

A public blockchain is a decentralized network that anyone can join and participate in without any restrictions. These blockchains are completely open, allowing anyone to read, write, or participate in the consensus process (i.e., mining or staking). Bitcoin and Ethereum are classic examples of public blockchains. The main advantages of public blockchains include their high level of security, transparency, and immutability. However, they often face challenges related to scalability, privacy, and transaction cost [43, 44], given their open nature.

Private Blockchain

A private blockchain, in contrast, operates within a closed network. It is typically controlled by a single organization or entity that determines who can join the network, submit transactions, or participate in the consensus process. Private blockchains offer greater control over the network, which can lead to improvements in speed and scalability compared to public blockchains.

Permissioned Blockchain

A permissioned blockchain is a broader category that can be either public or private but requires approval for participants to join the network. In a permissioned blockchain, the controlling entity or entities set rules for who can participate in the network, execute transactions, and in some cases, view certain data.

Consortium Blockchain

A consortium blockchain is a specific type of permissioned blockchain that is governed by a group of organizations rather than a single entity. In this setup, multiple organizations collaborate and share the responsibilities of maintaining the blockchain, including managing the consensus process and validating transactions. Consortium blockchains combine the benefits of both private and public blockchains, offering a balance between decentralization, security, and scalability. They are ideal for scenarios where participants need to share data securely and efficiently while maintaining equal control over the network.

2.2 Consensus Protocols

Blockchain operation is fundamentally rooted in consensus mechanisms, such as POW or POS, which ensure all transactions are validated by multiple nodes in the network before being added to the ledger. This validation process not only secures transactions against fraud but also enhances security through advanced cryptographic techniques. Beyond POW and POS, other consensus algorithms like Practical Byzantine Fault Tolerance (PBFT) and Raft are utilized, particularly in permissioned blockchain settings where efficiency and fault tolerance are important. PBFT achieves consensus through a deterministic voting process among nodes, ensuring robustness and reliability, while Raft simplifies consensus through leader election and log replication [6, 31]. The selection of an appropriate consensus algorithm is critical as it directly impacts the integrity, scalability, and performance of the blockchain network, influencing its suitability for various applications.

2.2.1 Proof-of-Work

Proof-of-Work is a consensus algorithm used by some blockchain networks to validate transactions and create new blocks [45]. It requires participants, also called miners, to solve complex mathematical puzzles, a process that demands significant computational power and energy. POW provides a high level of security due to the extremely costly and time consuming process of altering the blockchain. Though secure, it comes with drawbacks such as high energy consumption and scalability concerns [24].

2.2.2 Proof-of-Stake

Proof-of-Stake is another consensus algorithm that tries to improve some of the limiting factors of POW. In a POS network validators are selected based on the amount of cryptocurrency they have placed as a collateral. The higher the stake, the higher the chance of being chosen to validate transactions and add a new block to the blockchain. POS is much more energy-efficient than POW because it eliminates the need for energy-intensive mining activities. Validators are chosen through a deterministic process, depending on their stake, rather than solving cryptographic puzzles [11].

2.2.3 Avalanche consensus

The consensus protocol the Avalanche blockchain uses differs from POW and POS. Utilizing an approach known as repeated sub-sampled voting, the Avalanche protocol ensures rapid and secure transaction validation through a decentralized and democratic process. This protocol boasts features such as speed, scalability, energy efficiency, and adaptive security [17]. In this consensus protocol, when a node seeks to validate a transaction, it does not consult the entire network. Instead, it selects a small, random group of other validator nodes and queries whether the transaction should be accepted. This method allows for a scalable and efficient consensus process without compromising security. Each of these randomly chosen validators then responds based on its current knowledge of the transaction's validity. The inquiring node collects these responses and sides with the majority opinion among the sampled validators. This process is repeated across multiple rounds and among various node. This approach reduces the chance of fraudulent transactions being accepted, as manipulating the consensus would require influencing the majority of responses across numerous, unpredictable subsets of validators [37]. Figure 2.2 depicts the basis of how the Avalanche consensus protocol works.

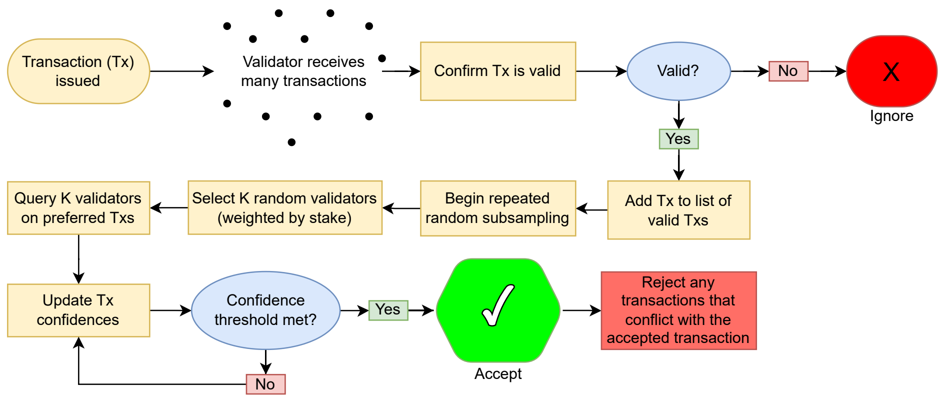

Transaction (Tx) issued
Validator receives many transactions
Confirm Tx is valid
Valid?
No
X
Ignore
Yes
Add Tx to list of valid Txs
Begin repeated random subsampling
Select K random validators (weighted by stake)
Query K validators on preferred Txs
Update Tx confidences
Confidence threshold met?
Yes
No
✓
Accept
Reject any transactions that conflict with the accepted transaction

Figure 2.2: Flowchart of the Avalanche consensus protocol, based on a figure from Ava Labs [17]

2.3 Ethereum Virtual Machine

The Ethereum Virtual Machine serves as the runtime environment for smart contracts on Ethereum and other EVM-compatible platforms like Avalanche [27]. The EVM is Turing-complete, which means it can execute any computation that a Turing machine can, provided sufficient time and resources are available [47]. This environment offers a sandbox for executing smart contracts written in higher-level languages, such as Solidity. Solidity is a statically-typed programming language designed specifically for developing smart contracts that run on the EVM. It is used to implement logic, define state variables, and handle events within smart contracts. Each operation within the EVM consumes a specific amount of gas, which is a unit reflecting computational effort. Gas fees are payable in the blockchain's native cryptocurrency (e.g., ETH for Ethereum, AVAX for Avalanche). Fees help prevent spam and incentivize validators to process transactions. Every operation on the EVM is processed by every node within the network, ensuring consistency and correctness due to the EVM's deterministic nature, meaning identical inputs always result in identical outputs. This mechanism ensures all computations are performed accurately across the network.

2.3.1 Smart Contracts

A smart contract is a program that is running on a blockchain. Both its code (functions) and its data (state) is stored on-chain [28]. Smart contracts can automatically execute transactions and other specified actions when predefined conditions and rules are met. Smart contracts are highly programmable, extending the functionality of blockchains beyond simply recording financial transactions. Executing smart contracts incurs costs in the form of gas fees, which vary based on network demand. Once a smart contract is created and deployed on the blockchain, it cannot be changed. This immutability ensures that no one can alter the contract's terms after it has been deployed. Due to this it is also important to rigorously test the contract to avoid irreversible bugs once deployed [38].

2.4 Avalanche

The Avalanche blockchain platform, developed by Ava Labs, is an open-source platform designed to build Decentralized Applications (DApps). Avalanche has its own unique consensus algorithm as explained earlier which allows for near-instant transaction finality [19]. The platform's compatibility with the EVM allows for seamless migration of smart contracts and DApps from other Ethereum ecosystems. Another core strength of Avalanche is its scalability. The platform can process thousands of Transactions Per Second (TPS), significantly higher than many traditional blockchains like Bitcoin or Ethereum in its initial versions [5]. This makes it an attractive platform for developers looking for a blockchain foundation for high-volume applications. Avalanche allows for the creation of subnets, or sub-networks, which are essentially custom blockchains tailored for specific use cases. These subnets can have their own validators and operate with their unique rules and parameters, providing a high degree of customization. This feature enables various industries and projects to build their blockchain solutions within the Avalanche ecosystem, catering to their specific needs [18].

2.5 Web3

Web3 is perhaps the next phase of the Internet, envisioning a decentralized online ecosystem backed by blockchain technology. This new web model seeks to address the limitations and concerns of the current internet landscape, known as Web2, particularly regarding data privacy, security, and ownership. By leveraging blockchain technology, Web3 enables P2P transactions, interactions, and exchanges without the need for intermediaries [25].

Unlike Web2, where centralized servers store data, Web3 uses blockchain to distribute data across numerous nodes, making it nearly impossible to control or censor information by any single entity. Another key feature of Web3 is that platforms operate on open protocols, allowing anyone to participate. The trustless nature of blockchain ensures transactions and interactions are secure and verifiable, reducing the reliance on trust in third parties. Lastly, cryptocurrencies and tokens are an integral part of Web3, making it easier to exchange assets without the need for traditional financial institutions. This opens up global access to financial services, especially for those in need of a proper banking system. It also removes the reliance on old banking infrastructure.

While Web3 promises a decentralized and user-empowered Internet, it is not without its challenges and drawbacks. One of the primary concerns with Web3, particularly blockchain technology, is scalability. Current blockchain networks can struggle to handle high transaction volumes quickly and efficiently, leading to slower processing times and higher costs compared to traditional Web2 platforms. Another crucial aspect is the usability and accessibility. Web3 technologies often face criticism for their lack of user-friendliness. The complexity of blockchain concepts, along with the need for secure management of cryptographic keys and wallets, can be intimidating for average users.

2.6 Decentralized Applications

A Decentralized Application is a type of digital application that operates not on a single computer or server but across a distributed blockchain network. This decentralized framework ensures that all data associated with the DApp is stored and managed on the blockchain, providing a level of security, transparency, and resistance to censorship that traditional centralized applications cannot match.

While DApps inherit the advantageous characteristics of blockchain technology, such as immutability, decentralization, and transparency, they also face some inherent challenges. Notably, one significant drawback is the difficulty in updating or modifying the application's code after it has been deployed to the blockchain. Unlike traditional software that can be updated or patched relatively easily, making changes to a DApp requires a consensus from the network participants or the deployment of a new smart contract, which can complicate maintenance and agile development [22]. This characteristic underscores the need for thorough testing and validation of DApps before deployment, as any flaws or vulnerabilities in the code become challenging to address post-launch.

2.7 Docker

Docker is an open-source platform designed to facilitate the creation, deployment, and operation of applications using containers. It employs a client-server architecture, where the Docker client communicates with the Docker daemon, which builds, runs, and manages the containers. Containers enable developers to package an application with all necessary components, such as libraries and dependencies, and distribute it as a single package [12]. This modular approach ensures that the application can run on any machine, regardless of specific machine configurations. Docker is particularly effective in microservices architectures, allowing for the independent deployment of individual components in separate containers. Additionally, containers are isolated from each other and from the host system, enhancing security by reducing the risk of accidental or malicious access to system files and resources.

2.8 Hashing Functions

A hash function is a function that can map data of an arbitrary size to a fixed-size string of bytes, or a digest. Cryptographic hash functions are a specialized type hash functions that possess specific properties [23]. These types of hash functions are quick to compute, deterministic (same input always yields the same output), pre-image resistant (computationally infeasible to reverse-engineer the input based on the output), sensitive to input changes (small change in input drastically changes the output), and collision-resistance (unfeasible to find two different inputs that generate the same output). Solidity uses the Keccak256 hashing algorithm, which is part of the SHA-3 family of hashing algorithms [2]. Keccak256 is used for tasks like computing unique hash values that ensure data integrity and generating random numbers.

2.9 Terminology

This section describes terminology used within the thesis that may not necessarily require its own dedicated section but still deserves an explanation.

Remote Procedure Call (RPC)s occur when a program invokes a subroutine or procedure on another computer, typically within a distributed system [26]. RPCs help abstract communication so that developers do not have to directly integrate with the network environment. They can be synchronous, where the calling program waits for a response before continuing, or asynchronous, which allows the program to continue executing while waiting for a response.

JavaScript Object Notation (JSON) is a key/value data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate. Though it originally stems from JavaScript, JSON is now a language-independent format that is universally supported across modern programming environments [4]. It is commonly used for transmitting data in web applications between clients and servers.

Application Programming Interface (API) provides an interface for programs to communicate with each other. APIs are a kind of software interface which helps link programs together [36]. They often send data in JSON format and can vary significantly in their design, such as RESTful APIs, which use HTTP requests to manage data, or SOAP APIs, which use XML for message transmission.

User-Interface (UI) enable humans to interact with computer programs visually through peripherals such as screens, mice, or keyboards. UI design focuses on optimizing the user's experience by ensuring that interactions are intuitive and efficient. Different types of user interfaces include Graphical User Interfaces (GUIs), which allow interaction through graphical icons and visual indicators, Command-Line Interfaces (CLIs), which interact through text commands, and touch interfaces [49].

2.10 Summary

This chapter lays the groundwork for understanding the thesis by explaining key concepts related to blockchain technology. It delves into different blockchain structures, consensus protocols like POW, POS, and Avalanche, and the EVM. It also covers smart contracts, the Avalanche platform, Web3 principles, Docker containerization, cryptographic hashing, and essential terminology.

3 Requirements

This chapter will be dedicated to the formulation and presentation of both functional and non-functional requirements for AquaTrace. These requirements are important as they lay down the specifications that the software must adhere to, ensuring that it functions effectively within the domain of seafood traceability and supply chain management.

3.1 Functional Requirements

Functional requirements in software engineering define the specific behaviors, actions, and functionalities that a software system must provide to its users. These requirements describe what the system should do, detailing the actions the system performs with its inputs. Functional requirements vary between applications and must be tailored to each specific application to meet user needs effectively [39, p. 92].

Requirement 1 (ID Generation). AquaTrace must be able to generate unique identification numbers for each new product entering the supply chain. Each ID should be unique and generated in a manner that prevents duplication.

Requirement 2 (ID Tracking). AquaTrace must track the location and status of each product throughout its journey in the supply chain, from catch to consumer. This includes real-time updates and historical data retrieval.

Requirement 3 (Owner verification). AquaTrace must verify the identity of users attempting to create or modify product IDs. This ensures that only authorized personnel can make changes to the product information.

Requirement 4 (Status updates). Users must be able to update the status of each product (e.g., caught, processed, shipped) through predefined stages in its lifecycle. These updates should reflect immediately in the system.

Requirement 5 (Geo-location tracking). AquaTrace should record and display the geographic coordinates of products at various stages of the supply chain to verify their location and compliance with geographic regulations.

Requirement 6 (Historical data access). Stakeholders should have the ability to access and review the historical data of any product, including its entire transaction history on the blockchain.

Requirement 7 (User Interface). There must be a user-friendly interface that allows various stakeholders to interact with the system easily. This interface should support all other functional requirements, providing access for ID generation, updates, and tracking.

3.2 Non-Functional Requirements

Non-functional requirements define the operational qualities and characteristics of a system that dictate how it must perform and operate, rather than what the system must do (which is defined by functional requirements). These requirements are critical as they often influence the user experience, system efficiency, and maintainability [39, p. 95].

Requirement 8 (Responsiveness). AquaTrace should be able to execute transactions and provide feedback to users within an acceptable timeframe. Given the system's reliance on blockchain technology, responsiveness is influenced by blockchain-specific factors like block time and network congestion. During periods of high transaction volume or network congestion, the system should manage user expectations and system performance effectively. To help with this users could be allowed to up their transaction fees so their transactions are prioritized during peak times.

Requirement 9 (Reliability). AquaTrace must consistently perform its intended functions correctly and dependably over time, even under stress or in unexpected conditions. It should consistently generate and manage IDs without errors or interruptions, ensuring that all operations, such as creating new IDs or updating existing ones, are executed as expected.

Requirement 10 (Availability). Availability in this scenario refers to the ability of the system to be accessible and operational at all times, providing uninterrupted service to its users. AquaTrace should be operational and accessible without unplanned downtime, supporting the 24/7 nature of the global fishing industry.

Requirement 11 (Security)**.** AquaTrace must ensure the confidentiality, integrity, and availability of the information it handles, while protecting against unauthorized access and data tampering. The inherent security features of blockchain technology, such as cryptographic hashing and the chaining of blocks, contribute to these goals. However, additional security measures are necessary to address potential vulnerabilities in the smart contract code. Specifically, the smart contract code must be thoroughly audited and analyzed before deployment to prevent exploits. The system must enforce strict access controls, ensuring that only authorized users can create, access, and modify IDs within the system. While most data will be stored on-chain and therefore immutable, it is essential to maintain transparency and auditability without compromising data security. Furthermore, AquaTrace must comply with data protection regulations, such as GDPR. The system will minimize the amount of data it stores and will not store any Personally Identifiable Information (PII) on-chain.

Requirement 12 (Usability)**.** Usability encompasses the ease of use, intuitiveness, and overall user experience provided by both the Web3 application and the underlying smart contract interactions. Usability is critical to ensure that all stakeholders can efficiently and effectively interact with the system. The user interface needs to be simple, intuitive, and tailored to the needs of users in the fishing industry, including fishermen, distributors, and regulators with various levels of technological competence. The interface should allow users to perform common tasks (e.g., generating IDs, viewing product history) with no more than a few clicks.

Requirement 13 (Maintainability)**.** Maintainability encompasses the ease with which the system can be corrected, improved, updated, or adapted over time. Effective maintainability ensures that the system remains functional and relevant as user needs evolve and the blockchain technology evolves. A well-maintained system could make it easier to migrate to other blockchains if the need ever arises.

Requirement 14 (Resilience)**.** Resilience refers to the system's ability to handle and recover from failures, disruptions, or attacks without significant degradation of performance or loss of data. Due to blockchain's inherent decentralization, both data and processing is distributed across a network of nodes which reduces the risk of single point of failures.

3.3 Summary

This chapter defines the system's functional and non-functional requirements. It specifies what the AquaTrace system must achieve, such as generating unique IDs and tracking product history, and outlines criteria for performance, usability, maintainability, security, and resilience.

4 Design and Implementation

This chapter details the design and implementation of the AquaTrace system. It covers the system architecture, smart contract development, frontend and backend functionalities, and deployment procedures. By breaking down the system into distinct components, we illustrate how each part contributes to the overall functionality, security, and scalability of AquaTrace.

4.1 System Architecture

AquaTrace is structured into several distinct layers and components. This system architecture enhances the system's scalability, maintainability, and security. The layered and component-based architecture of AquaTrace boosts scalability by allowing individual elements to be scaled independently and facilitating easier integration with other systems. This design also enhances maintainability through simpler updates and debugging, as components can be modified or replaced without impacting the entire system. Furthermore, security is strengthened by isolating critical components. The architecture is divided into four main components: the frontend (1), the backend (2), external sources (3), and the Web3 library (4) in Python and JavaScript. The Web3 library facilitates communication with the smart contract hosted on the Avalanche blockchain.

Figure 4.1 depicts how the different components communicate.

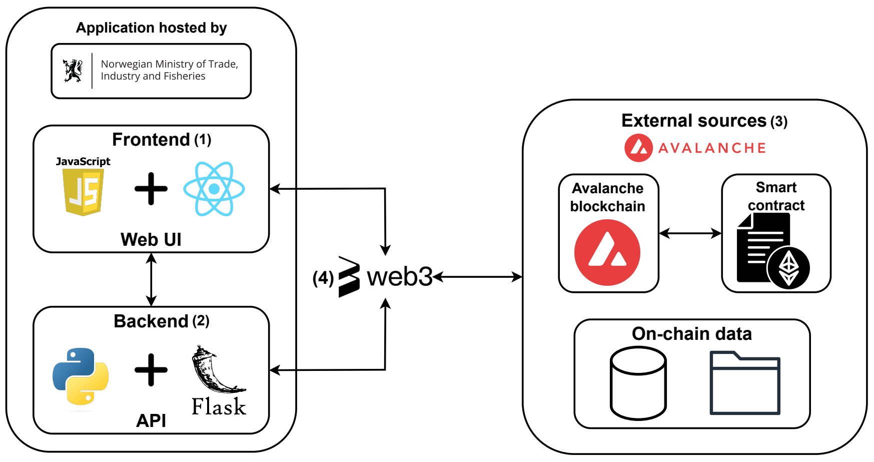


Figure 4.1: AquaTrace system architecture

The Web UI serves as the primary interface for end-users, enabling them to interact with the AquaTrace system. The UI is designed to be user-friendly, providing intuitive access to functionalities such as registering new product IDs, updating existing IDs, and retrieving historical data related to specific IDs. The frontend is implemented using React, which is a popular JavaScript library for building single-page applications. React allows for efficient rendering updates, which is beneficial for real-time data interaction such as when viewing and managing product IDs.

The backend of AquaTrace, while not essential for the core functionalities of the system, which are primarily managed through direct interactions between the frontend and the blockchain via Web3, serves a complementary role. It offers an API that replicates some of the functionalities available on the frontend. This setup could prove valuable for developers or other third-party services that require programmatic access to AquaTrace. The API provided by the backend does not perform any direct data manipulation or storage functions on the blockchain. Instead, it focuses on fetching data from the blockchain via smart contract functions. Table 4.1 shows the available API endpoints and briefly describes them. Python Flask is utilized for the backend framework due to its simplicity and effectiveness in creating lightweight REST APIs. Flask allows for easy setup of route endpoints that can serve data requests by returning JSON formatted responses, which makes integration with other systems easier. Both the frontend and backend utilize their respective Web3 libraries to directly interact with the blockchain. In the frontend it is used to facilitate direct blockchain transactions like generating new IDs, while in the backend it is used to perform read operations from the blockchain.

Endpoint	Method	Description
`/api/owners/<string:address>/ids`	**GET**	Fetches all IDs associated with an address. Returns a list of IDs in hexadecimal format.
`/api/ids/<int:id>/history`	**GET**	Fetches the history of a specific ID. Requires the ID in hexadecimal format.
`/api/ids/<int:id>/metadata`	**GET**	Retrieves metadata for a specific ID. The ID must be in hexadecimal format.
`/api/contract/address`	**GET**	Retrieves the address where the smart contract is deployed. Address is in hexadecimal format.

Table 4.1: API Descriptions for Product ID Management System

The external resources of AquaTrace consists of a smart contract, which is deployed on the Avalanche blockchain. This smart contract handles all interactions with the on-chain data and ensures all operations, such as ID creation and status updates, are executed according to the same rules. All operational data, including ID generation, updates, and historical tracking, is stored directly on the Avalanche blockchain. This approach eliminates the need for separate local data storage and leverages blockchain's inherent data immutability features, securing data against tampering. Having all data on-chain also simplifies the architecture by reducing the amount of components that must be managed.

Although the core data management and processing are performed on the Avalanche blockchain, AquaTrace provides a comprehensive API and user interface, making it accessible to various stakeholders. The application could be hosted and managed by a trusted entity such as the Norwegian Ministry of Trade, Industry, and Fisheries. This hosting ensures that the system adheres to national regulations and standards for data security and privacy while providing centralized oversight of the critical infrastructure.

4.2 Smart Contract

In the design of the AquaTrace system, the implementation of smart contracts is central to the functionality of the blockchain-based ID management system. The following sections outline the design choices and structures used within the Solidity smart contract to manage product metadata and track changes over time.

The smart contract for AquaTrace consists of several functions, data structures, also called structs, and events that are required for its operation. AquaTrace's functions can be seen in Table 4.2 while the structs and events will be described in the following section. These components are crucial for ensuring the integrity, traceability, and accessibility of product data throughout the supply chain.

Function	Type	Description
`generateIDs`	**NONPAYABLE**	Generates unique IDs for new products by using a nonce, timestamp, and sender's address, ensuring uniqueness.
`updateIDStatus`	**NONPAYABLE**	Updates the status and location of an existing ID after verifying user authorization.
`getIDMetadata`	**VIEW**	Returns metadata for a specific ID, including ownership details and product status.
`getIDHistory`	**VIEW**	Retrieves a history of changes for an ID, showing past statuses and locations to support traceability.
`getOwnerIds`	**VIEW**	Lists all IDs registered to a specific owner, providing a quick reference for user-owned products.

Table 4.2: Descriptions of the different functions within the smart contract.

In Solidity functions can be defined with specific types that dictate their behaviour and restrictions [8]. **View** functions promise not to modify the state of the contract. They are read-only functions that can return data and operate on the blockchain without any gas cost when called externally by users. If they are used internally within other functions to alter the state they will incur gas costs. **Pure** functions neither read or modify the state of the contract in any form. They only use function arguments and other local scope variables to compute and return values. **Nonpayable**, while not a keyword in Solidity, is the default function type if no other type is specified. Nonpayable functions has the ability to both read and alter the contract state and will incur gas costs when called. **Payable** functions are the only type of functions that can receive Ether. They are necessary for any function that needs to handle Ether transfers to the contract. This function type must be used if you expect to receive or handle Ether in transactions.

4.2.1 Generating IDs

The **generateIDs** function, seen in Figure 4.2, in the AquaTrace smart contract is a critical component designed to create unique identification numbers for new products as they enter the supply chain, fulfilling Requirement 1. This function takes two parameters: `numIds`, which specifies the number of IDs to be generated, and `productType`, which describes the category of the products. The function first checks that the number of IDs requested is greater than zero and that a product type has been provided, ensuring valid input data. It then enters a loop to generate the specified number of IDs.

```
Function generateIDs(numIds: uint256, productType: string)
    Require numIds > 0
    Require length(productType) > 0

    For i from 0 to numIds - 1 do
        id := hash(block.timestamp, sender address, nonces[
          ↪ sender])
        ownerIds[sender].push(id)

        idToMetadata[id] := {
            ownerId: sender,
            timestamp: block.timestamp,
            productType: productType,
            status: "Generated",
            latitude: 0,
            longitude: 0
        }

        idHistories[id].push({
            timestamp: block.timestamp,
            status: "Generated",
            latitude: 0,
            longitude: 0
        })

        nonces[sender] := nonces[sender] + 1
        Emit IDGenerated(id, sender, block.timestamp)
    End For
End Function
```

Figure 4.2: Pseudo-code for the **generateIDs** function in AquaTrace

Within the loop, each ID is created using the cryptographic hash function **keccak256**, which is the hashing function used in Solidity. This function combines the current blockchain timestamp, the address of the sender, and a nonce. The nonce ensures that each ID is unique even if the other parameters remain constant. This newly created ID is then associated with the sender's address and stored, along with the product's metadata, which includes the type, status, and initial coordinates set to zero, in the `idToMetadata` mapping. The ID and its creation details are also logged in the `idHistories` mapping to keep a history of each ID's state changes. After setting up the metadata and history, the nonce for the sender is incremented to prepare for the next potential ID generation. Finally, the function emits an `IDGenerated` event to signal that a new ID has been successfully created, along with relevant data such as the ID number, owner's address, and timestamp.

4.2.2 Data Structures

Structs allow for the grouping of related data points, which simplifies data management and enhances the efficiency of data retrieval operations. In AquaTrace, structs are particularly useful for organizing data related to product IDs. For example, they can be used to bundle metadata such as product type, status, and coordinates into a single structure. This organization not only makes the code more readable and maintainable but also optimizes the handling and storage of product information within the smart contract.

Product Metadata

The **ProductMetadata** struct, seen in Figure 4.3, is designed to encapsulate all relevant information about a product within a single data structure. It includes the address of the owner of the product which is crucial for tracking ownership throughout the supply chain. Additionally, a Unix timestamp of the last product update, this helps to maintain a chronological record of all changes. The type of product it is such as "King crab or "Salmon" is also included, helpful for managing certain product types or sorting. The current status of the product i.e. "Caught", "In transit", or "Processed" so stakeholders can easily know where in the processing stage the product is. Furthermore latitude and longitude stores the last known geographical location of the product which helps in logistic tracking or for verifying a products location, which fulfills Requirement 2. This struct facilitates easy management throughout a products lifecycle.

ID History

The **IDHistory** struct, highlighted in Figure 4.4, is designed to complement the **ProductMetadata** by recording the historical changes of each product ID within the AquaTrace system. It stores several essential fields that mirror those of the **ProductMetadata** struct, although with a focus on the chronology of events, fulfilling Requirement 6. This struct ensures robust traceability by

```
    struct ProductMetadata {
        address ownerId;
        uint256 timestamp;
        string productType;
        string status;
        int32 latitude;
        int32 longitude;
    }
```

Figure 4.3: Solidity struct for product metadata

maintaining a detailed and immutable log of all changes, which is essential for compliance, auditing, and maintaining the integrity of the supply chain information.

```
    struct IDHistory {
        uint256 timestamp;
        string status;
        int32 latitude;
        int32 longitude;
    }
```

Figure 4.4: Struct for tracking historical changes of product IDs

4.2.3 Events

Events in the AquaTrace smart contract serve as mechanisms for notifying external consumers (like user interfaces or other smart contracts) about specific actions that have occurred within the contract.

ID Generated

The **IDGenerated** event, as introduced in Figure 4.5, functions as a notification mechanism within the AquaTrace system, signaling the creation of new product IDs. This event is designed to work in conjunction with the **ProductMetadata** struct, providing real-time alerts that synchronize with the registration of new product information. When a new ID is successfully generated, the IDGenerated event is emitted. By broadcasting the creation of each new product ID it ensures that all participants in the supply chain are immediately aware of new entries.

```
    event IDGenerated(
        uint256 indexed id,
        address ownerId,
        uint256 timestamp
    );
```

Figure 4.5: Event emitted upon successful generation of a new product ID

ID Updated

The **IDUpdated** event, detailed in Figure 4.6 is used for announcing modifications to the existing product IDs, fulfilling Requirement 4. It functions in conjunction with the **IDHistory** struct, ensuring stakeholders are informed about changes to a product's record. Broadcasting these events in real-time enhances the transparency of the AquaTrace system. The struct enables all involved parties have access to accurate and timely information regarding a product's lifecycle.

Only the owner of an ID is authorized to update its information, ensuring that unauthorized parties cannot make changes. This owner verification mechanism fulfills Requirement 3, as it guarantees that updates to the product ID can only be made by the entity that generated it. This enhances the security and integrity of the data within the AquaTrace system.

```
    event IDUpdated(
        uint256 indexed id,
        uint256 timestamp,
        string newStatus,
        int32 latitude,
        int32 longitude
    );
```

Figure 4.6: Event Emitted When a Product ID's Metadata Is Updated

4.2.4 Gas Optimization

Gas consumption is a critical consideration in the design of the smart contract. Gas is the unit that measures the amount of computational effort required to execute operations on the EVM. Each operation consumes a certain amount of gas, and users must pay for this gas in the blockchain's native cryptocurrency (e.g., ETH for Ethereum, AVAX for Avalanche). Optimizing gas usage is important for several reasons: it directly translates to lower transaction costs for users, particularly important for applications like AquaTrace where frequent interactions with the smart contract are expected. Efficient gas usage also enhances scalability by allowing the system to handle a larger number of transactions without excessive costs. Additionally, by minimizing gas consumption, the overall load on the blockchain network is reduced, contributing to better network performance and reducing the risk of congestion.

To optimize gas usage in AquaTrace, several strategies were employed:

- **Efficient Data Structures:** Using mappings for quick data retrieval and storage helps reduce the computational effort required for operations, thereby saving gas.

- **Minimized Storage Operations:** Storage operations are costly in terms of gas. By minimizing the number of storage writes and utilizing events for logging data that doesn't need to be persistently stored on-chain, we reduce gas costs.

- **Batch Processing:** Where possible, operations such as generating multiple IDs are batched into a single transaction to leverage the inherent efficiency of batch processing.

These strategies collectively ensure that the AquaTrace system operates in a cost-effective and scalable manner, optimizing the use of gas and thereby reducing transaction costs and improving overall system performance.

4.2.5 Error Handling and Security

Security and robust error handling are crucial in the smart contract implementation. Ensuring that only authorized users can perform specific operations and that all inputs are valid is essential for maintaining the integrity and reliability of the system. Implementing these measures helps prevent unauthorized access and common vulnerabilities, safeguarding the contract against potential attacks and ensuring it operates as intended.

Access control is enforced using `require` statements that check the sender's permissions as seen in Figure 4.7, ensuring only authorized users can perform certain operations. Input validation ensures that only valid data is processed by the contract, preventing common vulnerabilities such as buffer overflows. These strategies contribute to the overall efficiency, security, and reliability of the AquaTrace system, ensuring that it can handle the demands of a real-world seafood supply chain while maintaining the integrity and security of product data.

These strategies and mechanisms contribute to the overall efficiency, security,

```
Function updateIDStatus(id: uint256, newStatus: string,
    ↪ latitude: int32, longitude: int32)
    Require idToMetadata[id].ownerId == msg.sender
    Require latitude >= -90 * 1e5 AND latitude <= 90 * 1e5
    Require longitude >= -180 * 1e5 AND longitude <= 180 * 1e5

    newHistory := IDHistory(block.timestamp, newStatus,
        ↪ latitude, longitude)
    idHistories[id].push(newHistory)

    idToMetadata[id].timestamp := block.timestamp
    idToMetadata[id].status := newStatus
    idToMetadata[id].latitude := latitude
    idToMetadata[id].longitude := longitude

    Emit IDUpdated(id, block.timestamp, newStatus, latitude,
        ↪ longitude)
End Function
```

Figure 4.7: Pseudo-code for the **updateIDStatus** function in AquaTrace

and reliability of the AquaTrace system, ensuring that it can handle the demands of a real-world seafood supply chain while maintaining the integrity and security of product data.

4.3 Frontend

The frontend of AquaTrace serves as the primary interface of the application and tries to fulfill Requirement 7. It functions as a Decentralized Application (DApp) that allows users to engage directly with the blockchain-based functionalities of the smart contract. The following sections will describe functionalities provided by the frontend, and how it facilitates interaction between the users and the AquaTrace smart contract.

4.3.1 Interacting with Contract Functions

When users need to interact with a smart contract function on AquaTrace, whether to generate new IDs or to check the history of an existing one, they can do so through the designated Contract page. This page is depicted in Figure 4.8. To interact with payable functions, users are required to have a cryptocurrency wallet browser add-on installed that supports the Avalanche blockchain. Compatible wallets include MetaMask, TrustWallet, and Core, the latter being used in this example. Additionally, users must ensure that their wallet is funded with AVAX, which is the native currency of the Avalanche network.

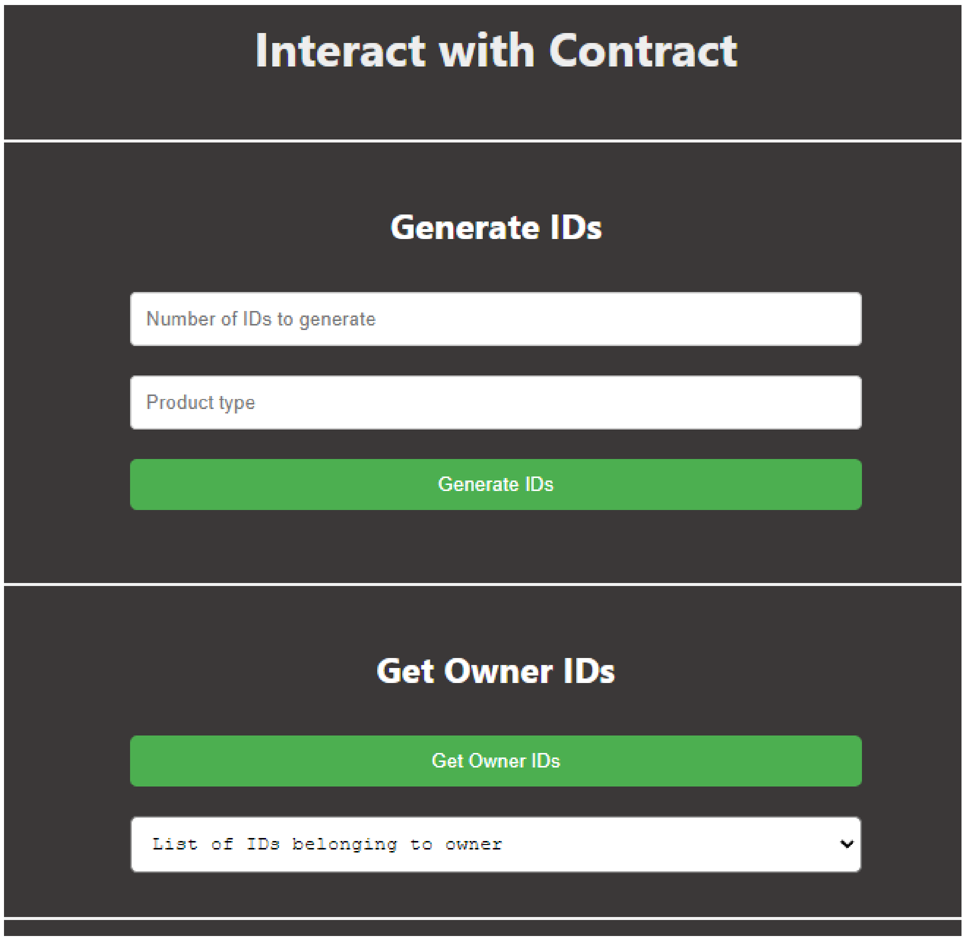
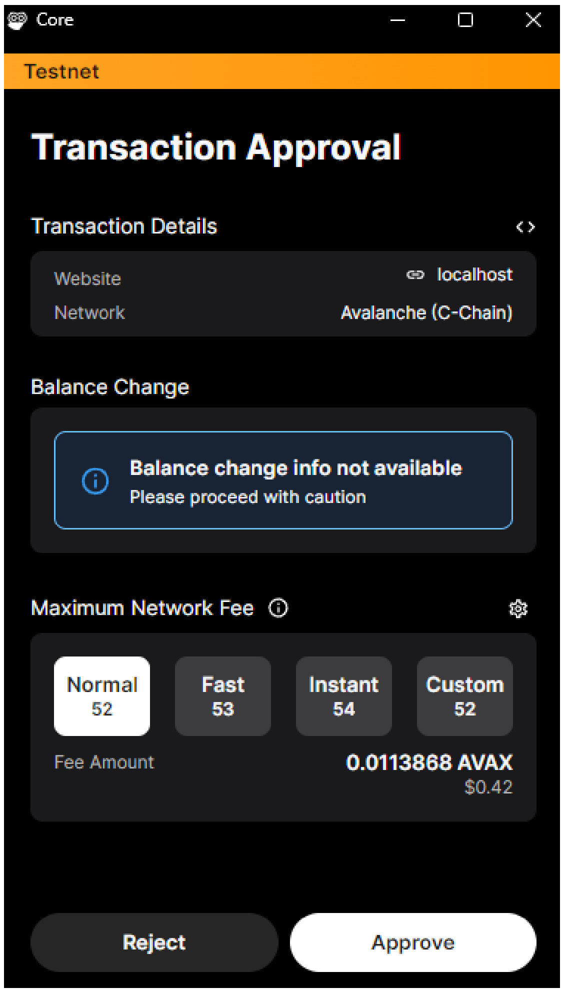


Figure 4.8: Screenshot displaying the page where users can interact with different smart contract functions.

The Contract page facilitates interactions with all types of contract functions, both payable and view. For example, if a user wants to generate new IDs, they simply enter the desired quantity and specify the product type, such as salmon, king crab, or prawns. After entering this information, the user would click the `Generate IDs` button. This action triggers a confirmation screen, as shown in Figure 4.9, which allows the user to review and adjust transaction details such as gas costs and gas limits, and to either approve or deny the transaction.

While view functions are accessible on this page, for user convenience, there is a dedicated tracking page specifically designed for these functions, which offers a tailored user interface that focuses on monitoring and retrieving data. This separation ensures that operational functions and tracking functionalities



Transaction Approval

Transaction Details

Website localhost
Network Avalanche (C-Chain)

Balance Change

Balance change info not available
Please proceed with caution

Maximum Network Fee

Normal 52
Fast 53
Instant 54
Custom 52

Fee Amount 0.0113868 AVAX
$0.42

Figure 4.9: Screenshot displaying the transaction confirmation users are prompted with when calling payable functions.

are intuitively organized, improving navigation and usability. The specific design of the tracking page and its user interface will be discussed in a later section.

After the transaction has been approved, it enters a comprehensive transaction lifecycle [34] where different processes happen behind the scenes. The transaction is encapsulated within an RPC and distributed to various nodes across the network. As nodes receive the transaction they have to validate it to confirm the authenticity of its contents such as signatures, nonce, and gas limits. Once transactions are validated they are added to the individual nodes memory pool, which is a sort of waiting area where transactions wait to be mined or validated further by the network. Depending on the gas prices associated with a transaction it could be prioritized higher or lower. Higher gas prices result in faster processing. When a new block is formed the transaction is executed as part of the block's creation. This includes the execution of any smart contract function called by the transaction, the result is then recorded on the blockchain. The newly formed block is then broadcasted to the network where other nodes verify the validity of it themselves. The network then has to reach consensus using the Avalanche-specific consensus protocol, as described earlier. Once consensus is achieved and the block is added to the blockchain, the transactions contained within are considered fully confirmed.

```
{
    "blockHash": "0x963c173...",
    "blockNumber": "32644960",
    "cumulativeGasUsed": "221067",
    "effectiveGasPrice": "25500000000",
    "from": "0x6169f...",
    "gasUsed": "221067",
    "logs": [ ... ],
    "logsBloom": "0x000800...",
    "status": "1",
    "to": "0x8afd11...",
    "transactionHash": "0xb1d41...",
    "transactionIndex": "0",
    "type": "2",
    "events": {
        "IDGenerated": {
            ...,
            "returnValues":  {
                ...,
                "id": "1805884447...",
                ...,
            }
            ...,
        }
    }
}
```

Figure 4.10: Example of a transaction receipt when generating new IDs. Longer fields and values have been truncated for readability, for the full transaction receipt see A.1. For an explanation of what each field is see Table A.1.

After a transaction has been confirmed, the transaction receipt becomes available, providing a record of the transaction's completion and the effects it had on the blockchain. The transaction receipt is generated by the blockchain network after the transaction has been included in a block and validated by the network nodes. Figure 4.10 displays a partial transaction receipt which notably includes the events emitted during the transaction. This receipt allows users to verify the details of the generated IDs and access other specific details related to the transaction. Once IDs have been generated, users can then proceed to the tracking page to begin managing and monitoring them actively.

4.3.2 Product Tracking Using IDs

Building on the contract functions introduced earlier, this subsection explains how AquaTrace leverages these capabilities for product tracking. By assigning unique IDs to each product, the system enables continuous monitoring from the point of capture to delivery to consumers. This approach provides users and stakeholders with real-time updates about their products, ensuring transparency and reliability throughout the supply chain.

To start tracking and managing products users can navigate to the tracking page within the AquaTrace web application. Users are met by a user-friendly interface featuring a blank world map alongside a dropdown box. This dropdown allows users to select an ID to view its history and interact with a form to update the product's status, as illustrated in Figure 4.11.

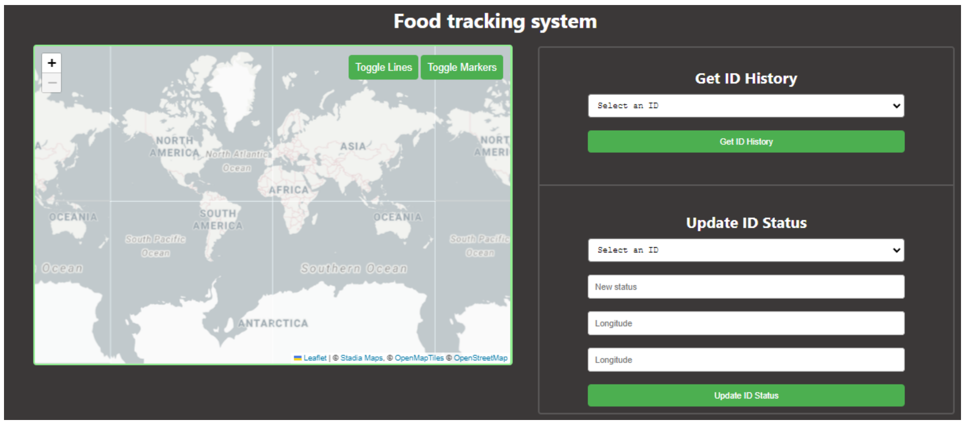
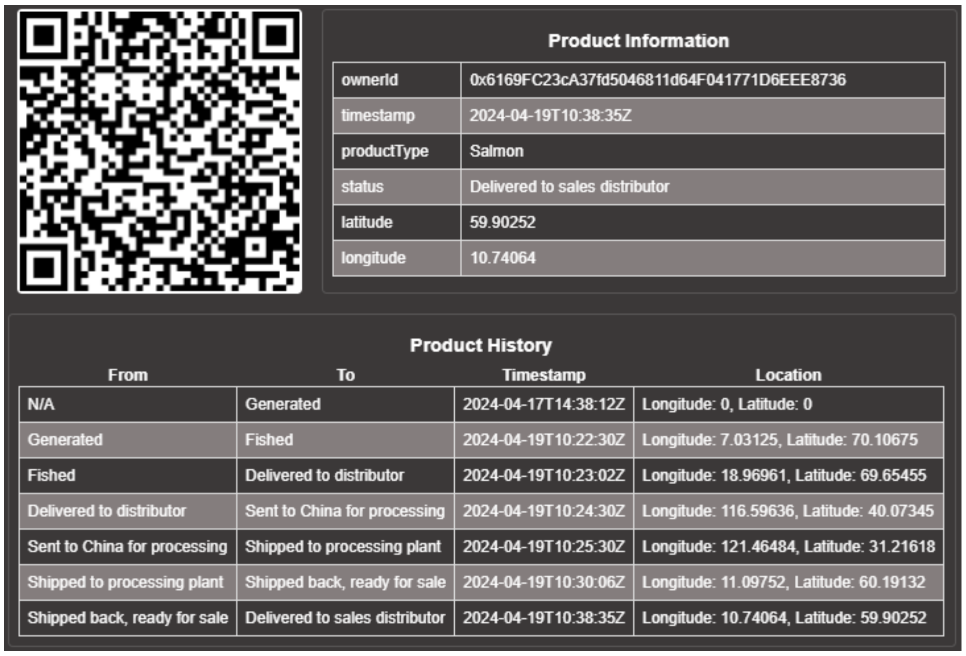


Figure 4.11: Screenshot displaying what the ID tracking page looks like with no IDs selected.

After selecting an ID, users can retrieve detailed information about the product by clicking the `Get ID History` button. The resulting display, shown in Figure 4.12, includes several tables providing essential details such as the current status, location, owner, product type, and the timestamp of the last update. An additional table presents a comprehensive timeline of the product's lifecycle, noting every significant event along with corresponding locations and timestamps. Additionally, each ID is associated with a QR code, enabling quick scanning for convenient status checks or modifications directly from mobile devices.



Product Information

ownerId	0x6169FC23cA37fd5046811d64F041771D6EEE8736
timestamp	2024-04-19T10:38:35Z
productType	Salmon
status	Delivered to sales distributor
latitude	59.90252
longitude	10.74064

Product History

From	To	Timestamp	Location
N/A	Generated	2024-04-17T14:38:12Z	Longitude: 0, Latitude: 0
Generated	Fished	2024-04-19T10:22:30Z	Longitude: 7.03125, Latitude: 70.10675
Fished	Delivered to distributor	2024-04-19T10:23:02Z	Longitude: 18.96961, Latitude: 69.65455
Delivered to distributor	Sent to China for processing	2024-04-19T10:24:30Z	Longitude: 116.59636, Latitude: 40.07345
Sent to China for processing	Shipped to processing plant	2024-04-19T10:25:30Z	Longitude: 121.46484, Latitude: 31.21618
Shipped to processing plant	Shipped back, ready for sale	2024-04-19T10:30:06Z	Longitude: 11.09752, Latitude: 60.19132
Shipped back, ready for sale	Delivered to sales distributor	2024-04-19T10:38:35Z	Longitude: 10.74064, Latitude: 59.90252

Figure 4.12: Screenshot displaying what the ID tracking page looks like when an ID has been selected.

For products with more than one tracking point, users can visually trace the entire journey on a Leaflet map, as depicted in Figure 4.13. This feature enhances user understanding by providing a visual representation of the product's journey, making it more intuitive than deciphering longitudinal and latitudinal coordinates alone. This fulfills Requirement 5 by enabling precise geo-location tracking of products, ensuring transparency and accountability as stakeholders can easily access and review the product's movement through the supply chain.

4.4 Deployment

The deployment of AquaTrace involves several steps that integrate both the blockchain components and the web application components. This section outlines the process through which the smart contract is compiled and deployed to the Avalanche Fuji testnet, and how the frontend and backend are containerized, stored, and deployed using Azure services. A central trusted authority will be hosting the blockchain and other infrastructure such as containers,

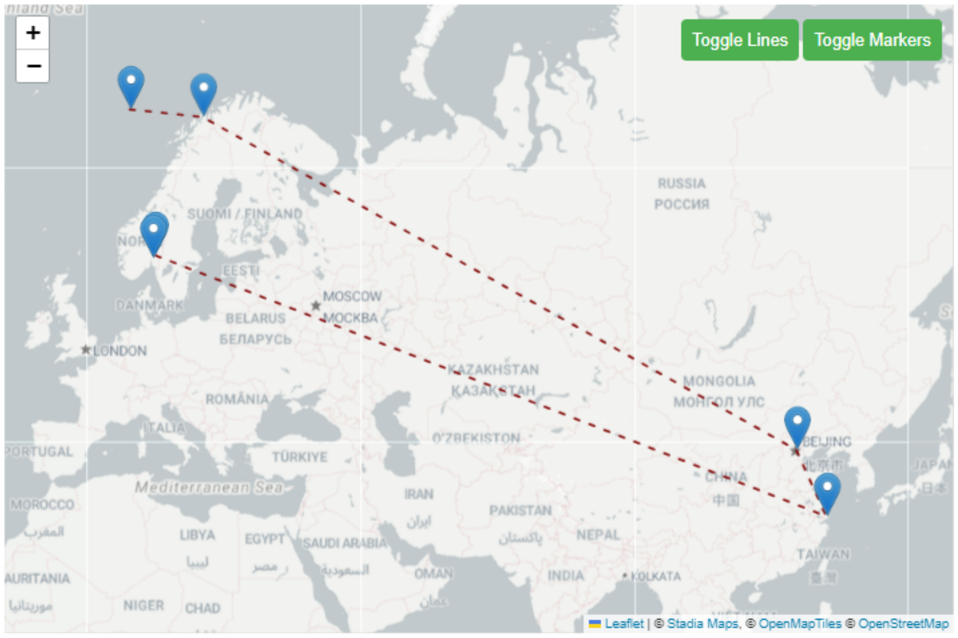


Figure 4.13: Screenshot displaying the tracking of a product using a Leaflet map. This map visually represents the journey of a product that was fished in the Norwegian Sea, unloaded in Tromsø, processed in China, and then shipped back to Norway for distribution to consumers.

source code, and backend servers, ensuring centralized oversight and support. However, it is important to note that while the trusted authority provides the hosting infrastructure, they do not exert control over the blockchain itself, maintaining the decentralized nature of the system.

4.4.1 Smart Contract Compilation and Deployment

The first step in deploying AquaTrace involves compiling the smart contract using the Solidity compiler via the Web3 library, which facilitates interaction with the Ethereum blockchain. The compilation process translates the high-level Solidity code into bytecode that can be executed on the blockchain. Additionally, Web3 is used to generate the contract's Application Binary Interface (ABI), which acts as an interface between the smart contract and the application.

1. **Compiling the Contract:** The Solidity code is compiled using the solc compiler integrated into the Web3 library. This process checks for any syntax or logical errors and ensures the contract is ready for deployment.

2. **Deploying to Avalanche Fuji Testnet:** Once compiled, the contract bytecode and ABI are used to deploy the contract to the Avalanche Fuji testnet. This testnet provides an ideal environment for testing the contract's functionality without incurring the costs associated with the mainnet deployment. The deployment is initiated through a function that specifies the Fuji network details and supplies the necessary gas and contract details.

4.4.2 Containerization and Image Deployment

Both the frontend and backend of AquaTrace are containerized using Docker, which ensures that they can operate reliably across different computing environments. These Docker containers are then pushed to an Azure Container Registry, from where they can be managed and deployed.

1. **Building Docker Images:** Dockerfiles are created for both the frontend and backend, specifying the base images, dependencies, and build commands. The images are built locally and tested to ensure that all components operate correctly.

2. **Uploading to Azure Container Registry:** Once validated, the Docker images are tagged and pushed to a repository in the Azure Container Registry. This registry acts as a single point of access for image management and version control.

4.4.3 Deployment Using Azure Container Apps

The final deployment is managed using Azure Container Apps, which allows for scaling and managing applications efficiently [50]. The deployment process involves the following steps:

1. **Deployment Configuration:** A deployment configuration file called `deployment.bicep` is prepared. This file defines the infrastructure requirements, such as compute resources, scaling rules, and network configurations.

2. **Executing Deployment:** The deployment is executed using the Azure CLI. This command-line tool communicates with Azure to create and manage resources based on the `deployment.bicep` file, effectively rolling out the application to production.

4.5 Summary

This chapter describes the architecture and development of AquaTrace. It covers the system's layered structure, detailing the roles of the frontend, backend, external sources, and Web3 library. The implementation of the smart contract on the Avalanche blockchain and the containerization process using Docker and Azure services are also discussed.

5 Evaluation

This chapter will be focusing on the evaluation of the AquaTrace smart contract implementation. Since the smart contract is the backbone of our system it is important to ensure that all operations like generating, tracking, updating IDs, and retrieving data works efficiently and securely. The evaluation aims to assess that the smart contract meets the requirements and to identify any potential areas for optimization.

5.1 Setup

The evaluation will utilize a suite of tools and libraries to deploy the smart contract, execute tests, collect performance data, and analyze results. The specifics of these tools and their roles in the evaluation process are detailed in Table 5.1.

The hardware setup for conducting the experiments includes a high-performance Dell OptiPlex Tower Plus 7010, this ensures that the evaluation process will not be bottlenecked by resource demands. Specifications of this system are detailed in Table 5.2.

After developing and testing the smart contract in Remix IDE, the contract was deployed to an Ethereum Virtual Machine-compatible blockchain, specifically the Avalanche Fuji testnet, for evaluation. The experiment process involved

Tool/Library	Purpose
Python ^3.10.12 [13]	Used for scripting test cases, data handling, and performing data analysis tasks necessary for evaluating system performance.
web3.py [33]	A Python library for interacting with Ethereum-based blockchains, used to connect to and perform actions on the blockchain, such as deploying contracts and making transactions for testing.
matplotlib.py [40]	A plotting library in Python, utilized for creating visualizations of data related to transaction times, costs, and other analyses during the evaluation phase.
Solidity ^0.8.0 [42]	The programming language used to write smart contracts deployed on Ethereum-based blockchains, essential for developing the contract logic that will be evaluated.
Remix IDE [41]	An open-source web and desktop application that simplifies the process of writing, testing, and deploying smart contracts, used for quick iterations during contract evaluation.

Table 5.1: Tools and libraries required for the evaluation of AquaTrace, detailing their purpose in the testing and validation processes.

Component	Specification
Central Processing Unit (CPU)	13th Gen Intel(R) Core(TM) i7-13700 (16C/24T@2.10GHz)
Grahpics Processing Unit (GPU)	NVIDIA GeForce RTX 3070
Random Access Memory (RAM)	128GB DDR4 RAM, running at 3600MHz
Storage	Micron 3400 SED NVMe 1024GB, 6600 MBps (read) / 3600 MBps (write)
Ethernet adapter	Intel(R) Ethernet Connection (17) I219-LM, Link speed 1000/1000 (Mbps)
Operating System (OS)	Windows 11 Enterprise version 22H2

Table 5.2: Hardware specifications of the system used to conduct the experiments.

using the web3.py library to interact with the blockchain, perform necessary transactions, and gather data for analysis. This setup enabled the systematic evaluation of the AquaTrace system's performance under various conditions, ensuring robust and reliable results.

In a real-world situation, gas prices can fluctuate based on network congestion, affecting transaction costs and confirmation times. However, since this evaluation is conducted on a testnet, such fluctuations are minimal and not as noticeable. This controlled environment allows for consistent testing conditions, ensuring that the performance data accurately reflects the smart contract's capabilities without the variability introduced by mainnet traffic.

5.2 Experiments

This section details each experiment conducted to assess the performance and reliability of the AquaTrace smart contract. The experiments are designed to examine various aspects of the smart contract's operations, from its efficiency and cost-effectiveness to its responsiveness and security. By analyzing each experiment, this section aims to provide a comprehensive view of the system's operational efficiency and responsiveness. There are many objectives with these experiments. Primarily we seek to verify that the AquaTrace smart contract adheres to its design specifications and meets the standards necessary for real-world applications. This evaluation will help ensure that AquaTrace can be confidently deployed with a clear understanding of its capabilities and limitations. It also aims to lay a foundation for ongoing improvements and provide a benchmark for future enhancements. By the end of this section, readers should have a thorough understanding of how AquaTrace has been tested, the results of these tests, and the implications for its future deployment. This comprehensive approach ensures that every side of the AquaTrace smart contract has been examined carefully, providing stakeholders with the assurance that the technology is not only innovative but also robust and reliable for its intended use.

5.2.1 Cost of Executing Different Smart Contract Functions

Our first experiment focus on quantifying and analyzing the gas costs associated with various functions of the AquaTrace smart contract to optimize cost-efficiency and economic viability. By systematically invoking each function within the smart contract and recording the gas consumed, the objective is to identify which functions may require further optimization to reduce operational expenses [7]. This evaluation is critical as it directly impacts the overall cost-effectiveness of deploying the smart contract in a real-world scenario.

To gather the results for this test, each function of the AquaTrace smart contract was executed once, and the gas cost for each operation was recorded. This process included deploying the smart contract, generating IDs, updating ID statuses, and retrieving data. The gas consumed for each function was measured to provide a clear picture of the operational costs associated with each smart contract action. These measurements were performed on the Avalanche Fuji testnet, which provides a controlled environment for testing without the variability and congestion often encountered on the mainnet.

Table 5.3 outlines the cost of invoking each of the contract functions, including deploying the contract it self onto the blockchain. The cost of generating IDs (marked with *) is the cost of generating 1 ID. Further cost analysis in **Table 5.4**, which provides a detailed breakdown of costs when generating IDs at scale. Pure and view functions (marked with **) do not incur gas costs and are free to call when called externally from a transaction, thereby not affecting the blockchain state. If they are called within other functions they will incur a gas cost as usual. On the Avalanche Fuji testnet, the base fee is set at 25 nAVAX per gas unit. To determine the cost of a transaction in AVAX, multiply this base fee by the number of gas units used. The conversion from AVAX to USD is calculated by multiplying the AVAX amount by the current exchange rate, which is approximately 1 AVAX ≈ $33.30 as of May 13, 2024.

Function	Costs		
	Gas units	AVAX	USD
Deploy smart contract	1,910,414	0.047760	$1.59
Generating IDs*	255,387	0.006384	$0.21
Updating ID status	98,905	0.002472	$0.08
Get ID metadata**	36,107	0.000902	$0.03
Get ID history**	33,254	0.000831	$0.03
Getting all owned IDs**	26,695	0.000667	$0.02

Table 5.3: Gas costs for executing different smart contract actions on the Avalanche blockchain.

Analyzing the table we see that deploying the smart contract is the most expensive operation. Though it is relatively costly compared to the other functions, this is a one-time operation and should be considered a fixed startup expense. This is typical for smart contract where the initial deployment incurs a higher cost due to the storage of the bytecode on the blockchain. Generating IDs is one of the core functionalities of AquaTrace and is relatively costly in terms of gas usage. Since this function will be used frequently, especially as new products are added to the system, its cost efficiency is crucial. Efforts to optimize this function could include minimizing state changes or restructuring how data is stored and retrieved. Further cost analysis regarding ID generation will performed in another experiment.

Updating the status of IDs is also a function that will be used frequently as products move through different stages of the supply chain. The cost is moderate, and optimizing this function could prove hard due to it already storing the minimum amount of information necessary to track a product. The rest of the functions can be invoked to read data from the blockchain without any cost when called externally. These functions do not modify the blockchain state and are crucial for providing stakeholders with access to data without incurring transaction costs. The reported costs apply only when these functions cause state changes, such as during internal calls made by other functions within the smart contract.

This detailed cost analysis helps in identifying potential optimizations and provides a benchmark for stakeholders to evaluate the economic feasibility of implementing the AquaTrace system on the Avalanche platform. Understanding these costs is essential for managing the financial aspect of running decentralized applications and ensuring sustainable operations.

5.2.2 Cost Analysis of Generating IDs

The operational expenses associated with generating IDs in the AquaTrace system are examined in this experiment to assess how scaling affects costs. The gas consumption necessary to generate varying numbers of IDs under simulated real-world conditions is quantified, providing insights into the scalability and financial feasibility of the AquaTrace system.

To measure the gas costs associated with generating IDs, the `estimate_gas` function in the web3.py library was used. This function provides an estimate of the gas required to execute a given function on the blockchain. For this experiment, IDs were generated in batches ranging from 1 to 100, and the gas costs for each batch size were recorded. Table 5.1 outlines the cost for a select number of IDs, while Table A.2 provides the cost for all recorded values from 1 to 100.

Number of IDs	1	5	10	25	50	75	100
Cost (Gas units)	255,387	1,009,941	1,952,968	4,782,230	9,498,278	14,215,077	18,932,643

Table 5.4: Gas costs for generating a given amount of IDs.

To quantify the relationship between the number of IDs generated and the associated gas costs, we can use linear regression analysis. The steps of the analysis are:

1. Calculate the means of the number of IDs (x_i) and the corresponding gas costs (y_i):

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

2. Compute the slope (m) of the best-fit line:

$$m = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

3. Determine the y-intercept (c):

$$c = \bar{y} - m\bar{x}$$

4. The resulting first-degree polynomial describing the cost model is:

$$y = mx + c$$

Applying the derived linear regression model to the data from **Table 5.4**, we obtain an equation that effectively predicts the gas costs associated with generating varying numbers of IDs:

$$y = 188656x + 66308$$

This model equation reveals that for each additional ID generated, the incremental gas cost is 188,656 units. This cost, when converted using the current base fee of 25 nAVAX per gas unit, equates to 0.0047164 AVAX. Based on the prevailing exchange rate of 1 AVAX ≈ $33.30, this translates to approximately $0.15 USD per ID. Figure 5.1 shows the linear correlation between the amount of IDs generated and gas cost. This mathematical model and the associated visual representation allow us to verify the linear relationship between the number of IDs and their generation costs.

This cost breakdown is useful for stakeholders in assessing the economic implications of scaling operations within the AquaTrace system. It provides a clear metric for budgeting and financial planning as the deployment scales, ensuring that the system remains economically viable as the number of IDs generated increases.

5.2.3 Time to Generate IDs Based on Gas Price

In the next experiment we explore the correlation between gas price and the time required for ID generation within AquaTrace, demonstrating that increased gas prices can reduce the transaction confirmation time.

The methodology involves performing ID generation transactions at different gas prices, starting from the network's average rate to 20% and 50% above

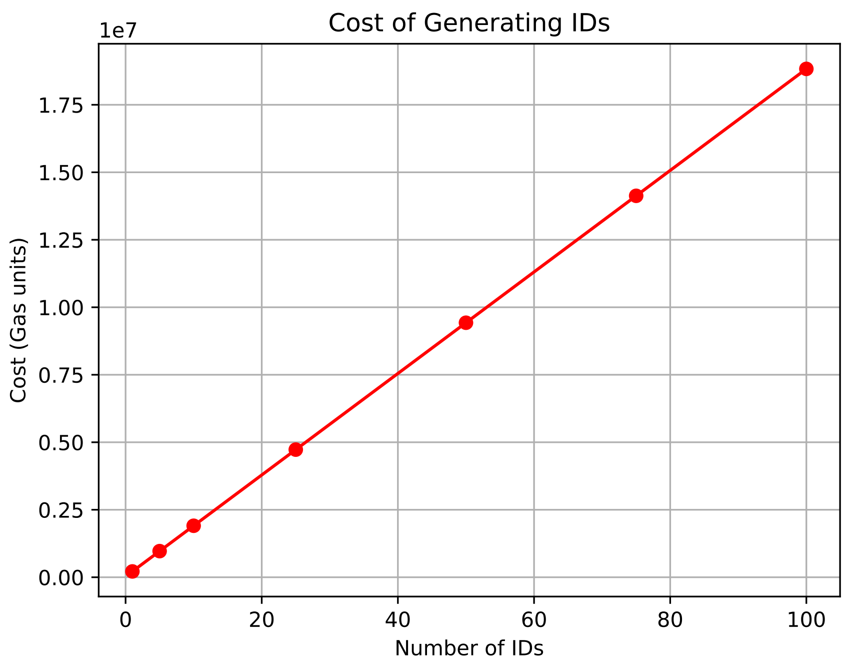


Figure 5.1: Graphical representation showing that the cost of generating IDs rises linearly with the number of IDs produced.

it, and measuring the time until transaction confirmation. Each trial was conducted with a consistent number of IDs to ensure that the data comparison remains valid across different gas price settings. The results indicate that increasing the gas price by 20% and 50% reduces the time taken to generate IDs, as shown in Figures 5.2, 5.3, and 5.4. This improvement occurs because higher gas prices incentivize miners to prioritize these transactions, leading to quicker inclusion in blocks and thus faster confirmations. However, the improvement in speed must be balanced against the increased cost [45], as higher gas fees could significantly inflate operational expenses. Additionally, the figures reveal that generating a larger batch of IDs does not take more time than generating a smaller batch, showcasing the efficiency of the AquaTrace system in handling bulk operations.

The figures 5.2, 5.3, 5.4, and 5.5 provide a visualization of how minor adjustments in gas price can affect operational efficiency. For instance, the time reductions observed at 20% and 50% increases are not linear, suggesting diminishing returns at higher fee levels. This behavior highlights the need for strategic management of gas prices, especially in scenarios where transaction speed is crucial but must be weighed against cost constraints.

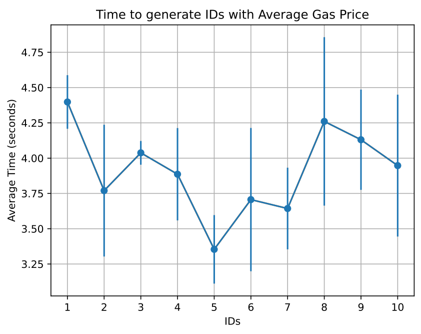


Figure 5.2: Graph showing the time it takes to generate different amounts of IDs using average Gas Price with standard deviations.

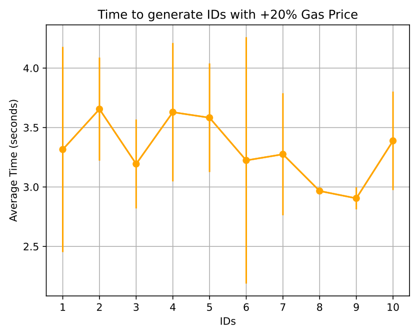


Figure 5.3: Graph showing the time it takes to generate different amounts of IDs using a 20% increased Gas Price with standard deviations.

For AquaTrace, these findings are important in optimizing the gas expenditure versus transaction latency trade-off. The system can be configured to dynamically adjust gas prices based on desired confirmation times and budgetary allowances. For critical operations where delay could affect the supply chain's integrity or lead to logistical complications, opting for a higher gas price may be justified. On the other hand, for less time-sensitive operations, it may be more cost-effective to accept longer wait times at lower gas prices. Also, the fact that generating a larger batch of IDs does not increase the time required further underscores the system's efficiency, making AquaTrace a robust solution for large-scale implementations.

5.2.4 Time Taken to Fetch Data from Blockchain

This experiment investigates the response times for retrieving data from the blockchain, specifically focusing on the efficiency of data retrieval operations using the `getIdMetadata` function. The goal is to understand the impact of repeated access and varying delays between requests on the latency of data retrieval, which is crucial for assessing user experience and system performance in AquaTrace.

The methodology involves repeatedly calling the `getIdMetadata` function and measuring the time taken for each request to be fulfilled. The experiment was conducted under three different conditions: no delay between requests, a 3-second delay between requests, and a 5-second delay between requests. This setup helps in identifying how network latency and blockchain response times vary under different conditions. Each trial consisted of ten consecutive

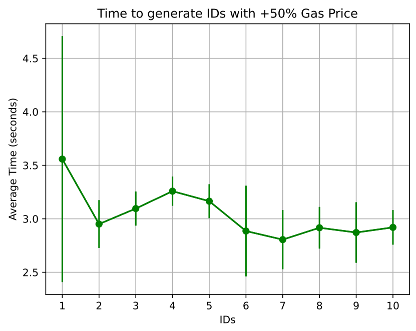


Figure 5.4: Graph showing the time it takes to generate different amounts of IDs using a 50% increased Gas Price with standard deviations.

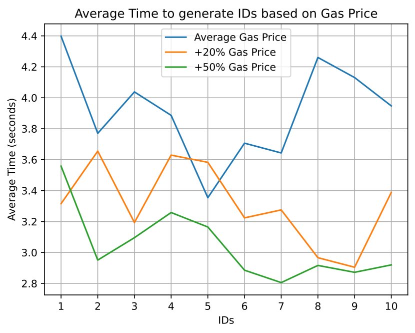


Figure 5.5: Graph comparing the average time it takes to generate different amounts of IDs depending on Gas Price. All associated data can be found in Table A.3, A.4, and A.5.

requests, and each trial was conducted five times to ensure consistency.

The results, as shown in Figures 5.6, 5.7, and 5.8, reveal interesting patterns. With no delay between requests, the first request generally takes longer, while subsequent requests are processed much faster. This behavior may be attributed to initial network overhead or caching mechanisms that speed up subsequent requests. On the other hand, when a delay of 3 seconds and 5 seconds is introduced, the response times for all requests increase significantly, suggesting that the initial optimization effect is negated with increased intervals between requests. This behavior could be due to the lack of continuous connection or session persistence, resulting in each request being processed as a fresh transaction.

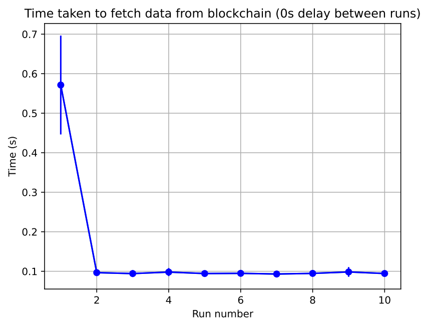


Figure 5.6: Time it takes to fetch data from the blockchain with no delay between runs, with standard deviations.

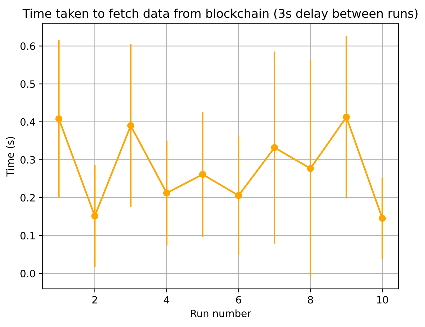


Figure 5.7: Time it takes to fetch data from the blockchain with 3 seconds delay between runs, with standard deviations.

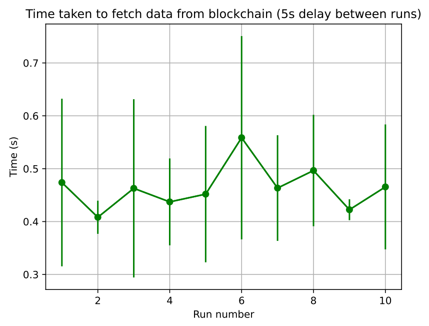


Figure 5.8: Time it takes to fetch data from the blockchain with 5 seconds delay between runs, with standard deviations.

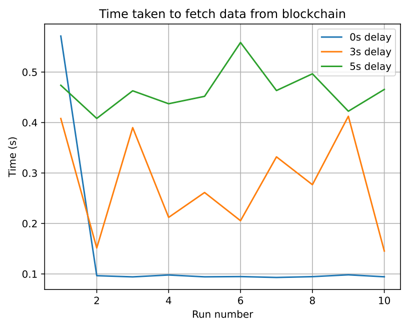


Figure 5.9: Comparison of time it takes to fetch data from blockchain depending on delay between runs.

The faster response times for subsequent requests with no delay highlight the presence of initial overhead. For operations requiring rapid, consecutive data retrievals, minimizing delay between requests is crucial. However, for less time-sensitive operations, introducing a delay will be less resource intensive, even if it increases individual request times. Balancing these factors will be key to ensuring AquaTrace's efficiency and user satisfaction.

5.2.5 Time Taken to Update the Status of an ID

This experiment focuses on measuring the efficiency and timeliness with which the AquaTrace smart contract processes ID status updates on the blockchain. The goal is to evaluate the smart contract's responsiveness to state changes, providing insights into the transaction processing speed and overall performance of the AquaTrace system.

The methodology involved updating the status of an ID and recording the time taken for each transaction to process. This was done 10 times per trial, and each trial was run 5 times to ensure consistency and reliability of the results. The average gas price was used for all transactions to maintain standard testing conditions. The latency times for these operations were recorded and analyzed to understand the variations and overall performance.

The results, illustrated in Figure 5.10, show the time taken to update the status of an ID across different trials. The times are relatively consistent, with small deviations observed across different runs. This consistency indicates a stable performance of the AquaTrace smart contract in handling state updates.

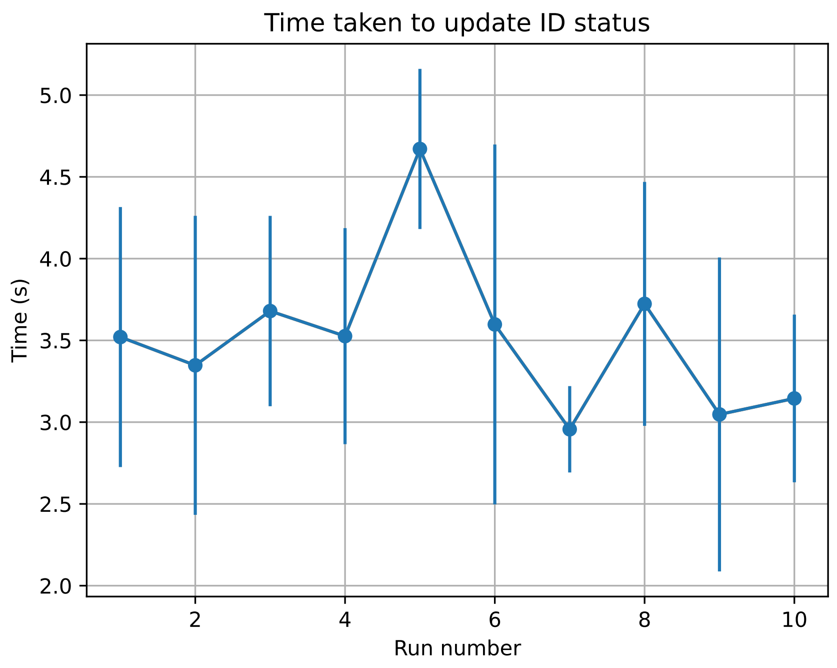


Figure 5.10: Time it takes to update the status of an ID with average gas prices. Extensive results can be seen in A.9.

The experiment reveals that the time taken to update the status of an ID is generally consistent, with small deviations across different trials. The average time per update was similar to the time taken to generate IDs, suggesting that both operations have comparable performance characteristics. The observed deviations in latency times are minor and can be attributed to typical network fluctuations and minor variances in blockchain processing times. This consistency can be attributed to the efficient handling of transactions by the AquaTrace smart contract and the underlying blockchain network.

The consistent performance in updating ID statuses demonstrates the reliability of the AquaTrace smart contract in handling frequent state changes. This reliability is crucial for maintaining accurate and up-to-date information in the food traceability system. The comparable performance in generating IDs and updating statuses suggests that the system is efficient in managing different types of transactions.

5.3 Summary

This chapter evaluates the AquaTrace system by focusing on key performance metrics, such as transaction latency and gas costs. Through testing, the evaluation provides insights into the system's efficiency and scalability. The results of these tests highlight the operational performance of AquaTrace, identifying both strengths and potential areas for improvement.

6 Discussion

This chapter provides an in-depth discussion about the AquaTrace system, focusing on the non-functional requirements outlined in Chapter 3 and the reasons behind the choice of blockchain. Additionally, it includes a discussion about real-world deployment considerations, covering aspects such as integration with existing systems, regulatory compliance, scalability and performance testing, and infrastructure and hosting strategies.

6.1 Non-Functional Requirements

In this section, we will focus on how our non-functional requirements are fulfilled and identify areas for potential improvement.

6.1.1 Responsiveness

The AquaTrace system features a responsive UI that immediately reacts to user actions. Users can interact with the application seamlessly, and the UI updates promptly to reflect these interactions. However, certain operations involving blockchain transactions and data fetching introduce some latency. Evaluation results indicate that performing transactions can take up to 3.5 seconds, while fetching data from the blockchain typically takes about 0.5 seconds. These delays are inherent to the blockchain validation process and the time required for RPC responses.

The primary challenge in improving responsiveness lies in the nature of blockchain operations. Transactions must undergo a comprehensive validation process before being confirmed on-chain, which takes time. Similarly, fetching data involves waiting for RPC calls to return the required information. To address these challenges, several improvements can be considered.

As mentioned earlier, one approach could be to dynamically adjust gas fees for transactions based on wanted transaction confirmation time (slow, medium, fast). Higher gas fees can prioritize their transactions, potentially reducing confirmation times during peak network usage. Additionally, implementing batch processing for data retrieval operations where feasible can reduce the overall wait time for users by fetching multiple pieces of data in a single RPC call. Lastly, introducing caching for frequently accessed data would also reduce the need for repeated RPC calls for the same information, thereby speeding up the retrieval process.

These improvements could help AquaTrace enhance its responsiveness and provide a more seamless and efficient user experience without compromising the integrity and security of the system.

6.1.2 Reliability

AquaTrace has proven to perform reliably under testing conditions on the Avalanche Fuji testnet. The system can consistently generate and manage IDs without errors or interruptions. The reliability in generating unique IDs is ensured by the sequential execution of transactions on the EVM [5]. This sequential execution guarantees that even if two users generate an ID at the same time, the transactions will still result in unique IDs.

The robustness of AquaTrace's reliability is theoretically supported by its design and the underlying blockchain technology. Although stress testing on a testnet posed challenges related to testnet funds, the principle of sequential execution of transactions on the EVM ensures that transactions are handled correctly and efficiently. The use of the Avalanche blockchain, known for its high throughput and low latency, further supports AquaTrace's reliability by ensuring that the system can handle a large number of transactions without significant delays or failures.

The inherent reliability provided by the blockchain's transaction handling also helps AquaTrace. Each transaction is validated before being processed, ensuring that only legitimate transactions are executed. This reduces the likelihood of errors and contributes to the overall reliability of the system.

These measures ensure that AquaTrace remains reliable, providing users with a dependable system for generating and managing product IDs, even under varying loads and conditions.

6.1.3 Availability

The AquaTrace system aims for high availability, theoretically achieving 100% uptime as long as the underlying blockchain is operational. Since the application primarily serves as a frontend interface for blockchain data, its availability is closely tied to the availability of the blockchain. However, if the frontend becomes unreachable, it can impact user access, even though the blockchain itself remains available.

Currently, there are no redundancy measures in place. However, given that AquaTrace is containerized using Docker, it is feasible to implement redundancy and scaling solutions using cloud services like Azure. This could involve deploying multiple instances of the application and using load balancing to ensure continuous availability even during high traffic periods or when some instances fail.

By leveraging cloud services for redundancy and scaling, AquaTrace can enhance its availability, providing a more reliable and consistent user experience.

6.1.4 Security

Security is a fundamental aspect of AquaTrace, which is inherent in blockchain technology. Blockchain provides robust security features that ensure the integrity, confidentiality, and availability of the data managed by AquaTrace.

AquaTrace's smart contract is designed to protect against unauthorized tampering. Only the individual who generates an ID can change or manage it, ensuring that control remains with the rightful owner. This access control is enforced through the smart contract's rules, which are immutable once deployed, further enhancing security.

In terms of confidentiality, while the data on-chain is publicly accessible, this transparency is intentional and beneficial for a food tracing system. It allows for easy and transparent audits, ensuring that all stakeholders can verify the authenticity and history of the products. This openness supports the goal of traceability and accountability within the food supply chain. From an integrity standpoint, the data's security is guaranteed by the blockchain's inherent properties. Each transaction is cryptographically secured, and the sequential addition of blocks ensures that once data is recorded, it cannot be altered without consensus from the network. This immutability protects the data from unauthorized changes and ensures its reliability. Although availability has been discussed previously, it is worth noting again that blockchain technology contributes to this aspect by ensuring continuous access to data as long as the blockchain network is operational.

A crucial aspect of AquaTrace's security is its collision resistance when generating IDs. IDs are generated using the Keccak256 hashing function, which ensures unique hash values. When generating IDs, each user has a personal nonce that is combined with other inputs such as the current timestamp and the user's address. This combination of inputs guarantees unique hash values for each ID generated. Given the vast number of possible outputs (2^256), the likelihood of generating two identical hashes (collision) is astronomically low. Even with a large number of IDs, the probability of a collision remains negligible. This number is unimaginably large, far beyond any practical scale of ID generation in the food traceability system. The collision resistance provided by Keccak256 ensures that each ID is unique and secure, which is vital for maintaining the integrity and traceability of the product information within the AquaTrace system.

By leveraging the security features of blockchain technology, AquaTrace ensures that its data remains secure, tamper-proof, and transparent, providing a trustworthy platform for food product tracing.

6.1.5 Usability

The AquaTrace system is designed to be easy to use and navigate, with clear menus and buttons. The UI employs consistent UI elements across all pages, helping users become familiar with the system quickly. This consistency ensures that once users learn how to use one part of the application, they can easily apply that knowledge to other parts. The system also includes features to help users input the correct data types. Input boxes are labeled and restricted to accept only the appropriate data types (e.g., string inputs for text and number inputs for numerical values). Dropdown boxes are also used to assist users in selecting the correct values, reducing the likelihood of input errors. Although AquaTrace currently lacks specific accessibility features, it does offer a permanent dark mode, which can reduce eye strain for some users. Future improvements could include developing a more comprehensive accessibility mode to support users with disabilities.

From a performance perspective, the frontend is responsive, providing immediate feedback to user actions. However, as mentioned earlier, RPCs and transactions can introduce delays ranging from 0.5 seconds to 3.5 seconds. Despite these delays, the overall user experience remains functional and efficient due to the design of the application.

Currently, there is no mobile version of AquaTrace. However, given that users would typically use this application on the go, porting it to mobile or even to an embedded device would be highly beneficial. Developing a mobile application, particularly for iOS, would require significant effort and resources, including an Apple developer account. Nonetheless, providing a mobile or embedded solution would enhance usability by allowing users to access the system flexibly and conveniently in various settings.

By maintaining a consistent design, AquaTrace ensures that users can interact with the system efficiently and effectively. Future enhancements, such as improved accessibility features, comprehensive documentation, and mobile or embedded device support, will further improve the usability of the system.

6.1.6 Maintainability

The AquaTrace system is designed with maintainability in mind, ensuring that it can be efficiently managed and updated over time. To achieve this, the system is split into separate containers for the frontend and backend, allowing each component to be deployed and maintained independently. This modular approach simplifies updates and debugging, as changes can be made to one component without affecting the others.

Despite these efforts, maintaining the smart contract presents a unique challenge. Once deployed, the smart contract is immutable, meaning it cannot be altered. This immutability ensures the integrity and security of the contract but also necessitates thorough testing and auditing before deployment to avoid any potential issues.

To enhance the maintainability of AquaTrace, logging and metrics could be implemented for all services. This would enable continuous monitoring of system health and facilitate easier debugging. Tools such as Docker logging drivers and monitoring solutions like Prometheus can be integrated to collect and analyze logs and metrics, providing valuable insights into the system's performance and potential areas for improvement.

By adopting these practices, AquaTrace can improve its maintainability, ensuring that the system remains robust and can easily adapt to future changes.

6.1.7 Resilience

AquaTrace leverages the inherent properties of blockchain technology to enhance its resilience. The decentralized nature of blockchain mitigates the risk of single points of failure, ensuring that data remains accessible and secure even if some nodes in the network go offline. By storing data on-chain, AquaTrace benefits from the robustness of blockchain, which guarantees data integrity and availability through consensus mechanisms.

The use of Docker containers further contributes to the system's resilience. Containerization allows AquaTrace to create additional instances of itself, enhancing redundancy and ensuring continuous operation even if some instances fail. This capability is crucial for maintaining service availability during peak loads or partial system failures.

To further enhance resilience, future improvements could include the implementation of error handling and recovery mechanisms. By integrating error handling, AquaTrace can ensure that any issues are detected and managed promptly, reducing the impact on users. Additionally, implementing recovery mechanisms would enable the system to restore functionality quickly after a failure.

Another potential enhancement is the introduction of backup data storage. While the primary data is securely stored on-chain, maintaining backup copies of critical information could provide an additional layer of protection. These backups could be used to quickly restore data in case of unexpected issues, ensuring that the system remains reliable and robust.

By leveraging the decentralization of blockchain technology and the flexibility of Docker containerization, AquaTrace achieves a high level of resilience. The suggested future enhancements will further strengthen the system's ability to provide continuous and reliable service under various conditions.

6.2 Choice of Blockchain

The choice of Avalanche as the blockchain for AquaTrace was motivated by several factors, including scalability, speed, efficiency, and cost-effectiveness. The most important criterion when selecting a blockchain was smart contract compatibility. This compatibility allowed for the seamless use of existing tools designed for developing and testing smart contracts, significantly easing the integration process. Additionally, Avalanche offers a public testnet, which was invaluable during the development phase for deploying and testing smart contracts without incurring high costs or risking real assets.

The discussions on scalability, speed, and efficiency are based on findings from a comparative study [51], which evaluates several blockchains. According to this study, Avalanche can handle approximately 5000 TPS per subnet. This level of scalability is crucial for AquaTrace, as it ensures that the system can handle high transaction volumes efficiently, particularly during peak usage periods. In comparison, Ethereum's maximum TPS is around 15. Such a limitation would likely result in heavy network congestion and long transaction times, making it less suitable for AquaTrace's needs.

Furthermore, Avalanche's speed and efficiency are significant advantages. The study reports that Avalanche has a block time of around 2 seconds and a time to finality of approximately 1 second. Time to finality refers to the time it takes for a transaction to be irreversibly committed to the blockchain, meaning it can no longer be altered or reversed. In contrast, Ethereum has a block time of 12-14 seconds and a time to finality of 60 seconds. These metrics highlight Avalanche's superior performance in processing transactions quickly and finalizing them almost instantaneously, which is essential for the real-time requirements of AquaTrace.

Cost considerations also played a vital role in the decision. As of May 18, 2024, the value of 1 ETH is equivalent to approximately 85 AVAX. Given that the base gas price fee fluctuates with network congestion, performing operations on Ethereum can be up to 85 times more expensive than on Avalanche, assuming equivalent gas prices. This cost efficiency makes Avalanche a more economical choice for AquaTrace, particularly for frequent transactions.

Both Ethereum and Avalanche boast large and active developer communities, which is beneficial given that both ecosystems use Solidity for developing smart contracts. This overlap facilitates the transfer of knowledge and resources. While Ethereum's community is larger due to its longer establishment, Avalanche's development team is actively producing articles and guides to support new developers. This proactive approach helps to bridge the gap and provides strong community support for those working on the platform.

Both Ethereum and Avalanche are under continuous development, ensuring they remain at the forefront of blockchain technology. Ethereum's recent transition from POW to POS has improved transaction throughput and reduced energy consumption. Similarly, Avalanche is continually enhancing its platform, promising ongoing improvements in performance and functionality.

In conclusion, the choice of Avalanche for AquaTrace is driven by its technical advantages, cost efficiency, and strong support ecosystem. Its EVM compatibility and public testnet facilitated a smooth integration and development process. The high scalability, speed, and economic benefits position Avalanche as a robust platform to support the demands of AquaTrace, ensuring efficient, secure, and cost-effective traceability solutions in the seafood supply chain.

6.3 Real-world Deployment Considerations

Deploying AquaTrace in a real-world environment involves addressing several practical challenges to ensure its effective operation within the seafood supply chain. This section discusses the key considerations and steps required for a successful deployment.

6.3.1 Integration with Existing Systems

A significant aspect of real-world deployment is integrating AquaTrace with other systems developed by the CSG group. Given that AquaTrace is part of an ongoing research interest within the CSG group, it is essential to ensure interoperability with these systems to enhance overall functionality and data consistency.

The CSG group has developed several systems that focus on different aspects of supply chain management, particularly within the seafood industry. For instance, SeaChain [15] aims to provide robust traceability within the fishing industry, sharing a similar objective with AquaTrace, though with a different focus. Integrating SeaChain with AquaTrace can enhance traceability features and provide a more comprehensive view of the supply chain.

To address the challenge of limited connectivity at sea, AquaTrace can implement edge computing strategies by processing data closer to its source on fishing vessels. The CSG group has developed the Dorvu file system [32], part of the Dutkat framework, designed for efficient, secure, and compliant data management in weakly connected environments. Integrating Dorvu with AquaTrace can enhance data management by ensuring that only the most relevant and critical information is transmitted over low-bandwidth satellite links to central servers. This integration will address bandwidth limitations while ensuring data privacy and regulatory compliance.

6.3.2 Regulatory Compliance

Compliance with local and international regulations is another critical aspect of real-world deployment. AquaTrace must adhere to data protection laws such as GDPR, ensuring that all personal and sensitive data is handled securely and transparently. Additionally, the system must comply with industry-specific regulations governing the seafood supply chain, such as traceability requirements and food safety standards.

The Dutkat framework [29], developed by the CSG group, is specifically designed to address regulatory compliance within the fishing industry. Dutkat focuses on maintaining compliance with various regulatory requirements while ensuring the privacy and security of the data collected. By integrating AquaTrace with the Dutkat framework, the system can leverage Dutkat's robust compliance mechanisms to ensure that all activities within the supply chain adhere to the necessary regulations.

6.3.3 Scalability and Performance Testing

Before full-scale deployment, extensive scalability and performance testing are essential. This involves simulating high transaction volumes and varying network conditions to evaluate how AquaTrace performs under different scenarios. Such testing helps identify potential bottlenecks and areas for optimization, ensuring that the system can handle the demands of a real-world environment.

To enhance the realism of these tests, the Njord dataset [30] developed by the CSG group can be utilized. The dataset provides surveillance videos from fishing trawlers, which include detailed annotations of activities and objects on board. By integrating this dataset into the testing environment, AquaTrace can simulate real-world conditions more accurately. This includes handling large volumes of data generated from onboard cameras and sensors, processing this data efficiently, and ensuring that the system remains responsive and reliable under heavy load.

The Njord dataset can also help in testing the system's ability to handle edge computing scenarios, where data is processed locally on the vessel before being transmitted over low-bandwidth connections. This is crucial for ensuring AquaTrace's performance and scalability in remote and bandwidth-constrained environments commonly found in the fishing industry.

6.3.4 Infrastructure and Hosting

The hosting infrastructure for AquaTrace needs to be robust and reliable to ensure continuous operation. Utilizing cloud services like Azure for containerized deployment provides scalability, security, and ease of management. Additionally, a central trusted authority, such as a governmental or industry body, can oversee the hosting infrastructure to ensure compliance with industry standards and provide centralized support without compromising the decentralized nature of the blockchain.

6.4 Summary

This chapter discusses how well the system meets non-functional requirements. It explains the rationale behind choosing the Avalanche blockchain, considering its technical and economic benefits. Furthermore, the chapter explores real-world deployment considerations, including integration with other CSG-developed systems, compliance with regulatory standards, and ensuring robust infrastructure.

7 Conclusion

This chapter provides an overview of the key findings and contributions of the thesis. It begins with a review of related work, situating this research within the broader context of blockchain applications in supply chain management. The chapter then details the specific contributions of this thesis, particularly in the secure generation and management of unique IDs within the seafood industry. It also outlines potential areas for future work, including the development of a mobile application, real-world deployment, and the implementation of logging and monitoring solutions. The concluding remarks reiterate the thesis statement and summarize the implications of the research findings, highlighting the potential of blockchain technology to enhance transparency and accountability in supply chains.

7.1 Related Work

In exploring the use of blockchain technology for supply chain management, it is essential to review existing literature and studies that have addressed similar challenges and solutions. This section provides an overview of recent and relevant work that has investigated the application of blockchain in various supply chain contexts. While the primary focus of this thesis is on the seafood and aquaculture sector, we will also examine studies from different sectors to gather a comprehensive understanding of blockchain's potential and its diverse applications. By selecting recent papers, we aim to incorporate the latest advancements and insights that inform and support the development of this thesis.

The paper titled *Agricultural Food Supply Chain Traceability using Blockchain* by Rajput et al. [35] explores the use of blockchain technology to enhance traceability in the agricultural food supply chain. The authors identify significant issues within current agricultural supply chains, such as the involvement of numerous participants, poor communication, distrust among members, and centralized control, which can lead to inefficiencies and opportunities for fraudulent activities. To address these challenges, the proposed system utilizes blockchain technology to create a decentralized, immutable, and tamper-proof ledger that tracks product information throughout the supply chain. The system integrates various technologies, including sensors to gather data during the production stages, the Interplanetary File System (IPFS) for secure data storage, and Radio Frequency Identification (RFID) tags for easy access to information by consumers. By storing data such as temperature, humidity, and other quality indicators in IPFS and linking it to the blockchain, the system ensures that all relevant information is securely recorded and accessible. The authors emphasize the benefits of blockchain in enhancing transparency, reducing management costs, and increasing the credibility of information within the supply chain. Consumers can verify the quality and origin of agricultural products through RFID tags, which access data stored on the blockchain. This approach not only helps in maintaining trust among all stakeholders but also supports the integrity and traceability of products from farm to fork. Overall, Rajput et al. demonstrates the potential of blockchain technology to transform the agricultural food supply chain by providing a robust and transparent traceability system, ultimately benefiting both producers and consumers by ensuring product quality and authenticity

Zhang et al.'s [52] paper, titled *A Blockchain-Based Traceability Model for Grain and Oil Food Supply Chain*, presents a comprehensive blockchain-based traceability model specifically designed for the grain and oil food supply chain. This model tackles significant challenges such as data centralization, tampering, and storage limitations by combining blockchain technology with Machine Learning (ML) to ensure the authenticity and reliability of source data. The system employs a lightweight blockchain-storage method and a data-recovery mechanism to alleviate storage pressures and enhance fault tolerance. Built on Hyperledger Fabric, an open-source blockchain framework designed for enterprise use, the architecture allows for efficient multi-source heterogeneous data uploading and secure data management. Key innovations include the use of Internet of Things (IOT) devices for real-time data collection and the integration of a three-layer anomaly-detection model to filter outliers, ensuring only accurate data is recorded. The results demonstrate that public data queries have an average latency of 0.42 seconds, private data queries 0.88 seconds, and data recovery 1.2 seconds, illustrating the model's efficiency and reliability. This study underscores the potential of blockchain to improve transparency, security, and efficiency in the grain and oil food supply chain, offering a robust framework for other agricultural sectors to follow.

The paper titled *Blockchain Implementation in Pharmaceutical Supply Chains: A Review and Conceptual Framework* by Ghadge et al. [14] reviews the potential of blockchain technology in addressing challenges within Pharmaceutical Supply Chains (PSC). It highlights the pressing issues of drug counterfeiting, lack of transparency, and inefficiencies in PSCs. Through a systematic literature review of 65 articles published between 2010 and 2021, the authors identify key drivers, barriers, and stages of blockchain adoption in PSCs. The study proposes a conceptual framework that outlines the initiation, adoption, and implementation stages for blockchain integration. The framework emphasizes blockchain's potential to enhance traceability, data security, and operational efficiency within the pharmaceutical sector. Additionally, the authors discuss the regulatory and technological hurdles that must be overcome for successful implementation. The paper underscores the need for further empirical research and provides directions for future studies, aiming to bridge the gap between theoretical exploration and practical application in the pharmaceutical industry.

The paper titled *Toward an Intelligent Blockchain IoT-Enabled Fish Supply Chain: A Review and Conceptual Framework* by Ismail et al. [20] explores the integration of blockchain technology with the IOT to enhance traceability and transparency in the fish supply chain. The authors review current research efforts and existing solutions, discussing both traditional and smart supply chains enabled by blockchain and IoT technologies. They propose an intelligent blockchain IoT-enabled framework designed to track and trace fish products through various stages of the supply chain, from harvesting to final delivery. This framework leverages distributed ledger technology to build trustworthy and decentralized traceability systems, providing timely and valuable information to verify the authenticity and quality of fish products. The integration of Machine Learning (ML) is also considered to enhance fish quality assessment, freshness evaluation, and fraud detection. The paper highlights Hyperledger Fabric for its role in ensuring data security and trust within the proposed framework. This platform allows for controlled access, ensuring that only authorized participants can validate transactions and access sensitive information, thus enhancing the overall security and efficiency of the supply chain system. By addressing significant challenges in the fish industry, including illegal, unreported, and unregulated activities, the proposed system aims to provide a robust solution for real-time monitoring and verification of fish products throughout the supply chain.

These studies underscore the versatility of blockchain technology in addressing various challenges across different sectors, from agriculture and pharmaceuticals to the fish industry. The integration of blockchain with other technologies like IoT and ML further enhances its potential, providing robust solutions for improving transparency, security, and efficiency in supply chain management.

7.2 Contributions

To recap, the thesis statement is:

Blockchain-based smart contracts can effectively be used to manage unique ID series for product tracing.

This thesis specifically focuses on the secure generation and management of IDs within the seafood industry, addressing challenges such as mutual mistrust among organizations and the threat of hostile entities. Unlike related works that often look at the supply chain management aspect as a whole, this research focuses on the issue of ID uniqueness and security. By leveraging the properties of blockchain technology, including its immutable ledger and decentralized nature, this work ensures that IDs are tamper-proof and verifiable. The system's design minimizes the risk of ID collisions and enhances the traceability of seafood products from capture to consumer.

7.3 Future work

Future enhancements to AquaTrace could focus on several key areas to improve functionality and user experience. The most significant improvement would be to develop a mobile application, making the system accessible for users on the go. This would enhance the usability and convenience of the system for stakeholders in the seafood supply chain. Another critical step would be to deploy the system in a real-world application to validate its effectiveness, scalability, and robustness under actual operational conditions, providing valuable insights and guiding further refinements. Finally, implementing logging and monitoring solutions, such as Docker logging drivers and Prometheus, would provide deeper insights into system performance and facilitate easier debugging. While these latter improvements are not necessary for a minimum viable product, they would contribute to the overall robustness and maintainability of the system.

7.4 Concluding Remarks

This thesis has explored the hypothesis that blockchain-based smart contracts can effectively manage unique ID series for product tracing. By leveraging the decentralized and immutable properties of blockchain technology, AquaTrace aims to address the challenges of security, trust, and tamper-resistance in the seafood industry. The thesis demonstrates how blockchain can potentially eliminate the need for a central authority while ensuring the integrity and uniqueness of product IDs.

The deployment of an EVM-compatible smart contract on the Avalanche blockchain and the development of a Web3 application have been central to this thesis. This approach aims to enable seamless interaction with the blockchain, allowing for efficient and secure management and tracing of product IDs. The practical application of these technologies highlights the potential of blockchain to enhance transparency and accountability throughout the supply chain.

In conclusion, AquaTrace represents a step in exploring the application of blockchain technology for supply chain management. While the system has not yet been tested in a real-life environment, the research outcomes suggest that blockchain can be an effective tool in the seafood industry and potentially other sectors. The system developed through this thesis provides a framework for future innovations aimed at improving the security and traceability of products in complex supply chains.

References

[1] Joakim Aalstad Alslie et al. "Áika: A Distributed Edge System for AI Inference." In: *Big Data and Cognitive Computing* 6.2 (2022). ISSN: 2504-2289. DOI: 10.3390/bdcc6020068. URL: https://www.mdpi.com/2504-2289/6/2/68.

[2] Guido Bertoni et al. "Keccak." In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 313–314. ISBN: 978-3-642-38348-9.

[3] Claude E Boyd, Aaron A McNevin, and Robert P Davis. "The contribution of fisheries and aquaculture to the global protein supply." In: *Food security* 14.3 (2022), pp. 805–827. DOI: 10.1007/s12571-021-01246-9.

[4] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. Dec. 2017. DOI: 10.17487/RFC8259. URL: https://www.rfc-editor.org/info/rfc8259.

[5] Vitalik Buterin. *Ethereum: A next-generation smart contract and decentralized ...* 2014. URL: https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf. (accessed: 17.03.2024).

[6] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance." In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. OSDI '99. New Orleans, Louisiana, USA: USENIX Association, 1999, pp. 173–186. ISBN: 1880446391.

[7] Ting Chen et al. "Under-optimized smart contracts devour your money." In: (Feb. 2017). DOI: 10.1109/saner.2017.7884650.

[8] chriseth. *Types - Solidity 0.8.26 documentation*. Accessed on 07/05/2024. 2015. URL: https://docs.soliditylang.org/en/latest/types.html#function-types.

[9] D. E. Comer et al. "Computing as a discipline." In: *Commun. ACM* 32.1 (Jan. 1989), pp. 9–23. ISSN: 0001-0782. DOI: 10.1145/63238.63239. URL: https://doi.org/10.1145/63238.63239.

[10] The Norwegian Seafood Council. *Nøkkeltall*. Accessed on 18/05/2024. 2024. URL: https://nokkeltall.seafood.no/.

[11] Victor Dusart. *Proof-of-stake (POS)*. Sept. 2023. URL: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/. (accessed: 17.03.2024).

[12] dvdksn. *Docker overview*. Accessed on 22/04/2024. 2023. URL: https://docs.docker.com/get-started/overview/.

[13] Python Software Foundation. *Python 3.10.12*. Accessed on 10/05/2024. 2023. URL: https://www.python.org/downloads/release/python-31012/.

[14] Abhijeet Ghadge et al. "Blockchain implementation in pharmaceutical supply chains: A review and conceptual framework." In: *International Journal of Production Research* 61.19 (2023), pp. 6633–6651.

[15] Erik Godtliebsen. "Product Tracing in the Norwegian Fishing Industry Supply Chain Utilizing GoQuorum Blockchain and Smart Contracts." MA thesis. UiT Norges arktiske universitet, 2023.

[16] IBM. *What is blockchain?* URL: https://www.ibm.com/topics/blockchain. (accessed: 17.03.2024).

[17] Ava Labs Inc. *Avalanche Consensus*. URL: https://docs.avax.network/learn/avalanche/avalanche-consensus. (accessed: 17.03.2024).

[18] Ava Labs Inc. *INTRODUCTION TO AVALANCHE SUBNETS*. URL: https://www.avax.network/subnets. (accessed: 17.03.2024).

[19] Ava Labs Inc. *What Is Avalanche?* URL: https://docs.avax.network/intro. (accessed: 17.03.2024).

[20] Shereen Ismail et al. "Toward an Intelligent Blockchain IoT-Enabled Fish Supply Chain: A Review and Conceptual Framework." In: *Sensors* 23.11 (2023). ISSN: 1424-8220. DOI: 10.3390/s23115136. URL: https://www.mdpi.com/1424-8220/23/11/5136.

[21] Håvard D. Johansen et al. "Fireflies: A Secure and Scalable Membership and Gossip Service." In: *ACM Trans. Comput. Syst.* 33.2 (May 2015). ISSN: 0734-2071. DOI: 10.1145/2701418. URL: https://doi.org/10.1145/2701418.

[22] Lodovica Marchesi, Michele Marchesi, and Roberto Tonelli. "ABCDE—agile block chain DApp engineering." In: *Blockchain: research and applications* 1.1-2 (Dec. 2020), pp. 100002–100002. DOI: 10.1016/j.bcra.2020.100002.

[23] Alfred J. Menezes et al. *Handbook of applied cryptography*. London, England: CRC Press, 2018. ISBN: 9780429881329.

[24] minimalsm. *Proof-of-work (POW)*. Sept. 2023. URL: https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/. (accessed: 17.03.2024).

[25] minimalsm. *What is web3 and why is it important?* Aug. 2023. URL: https://ethereum.org/en/web3. (accessed: 17.03.2024).

[26] Bruce Jay Nelson. *Remote procedure call*. Carnegie Mellon University, 1981.

[27] nhsz. *ETHEREUM VIRTUAL MACHINE (EVM)*. Accessed on 22/04/2024. 2023. URL: https://ethereum.org/en/developers/docs/evm/.

[28] nhsz. *INTRODUCTION TO SMART CONTRACTS*. Accessed on 22/04/2024. 2023. URL: https://ethereum.org/en/developers/docs/smart-contracts/.

[29] Tor-Arne S. Nordmo et al. "Dutkat: A Multimedia System for Catching Illegal Catchers in a Privacy-Preserving Manner." In: *Proceedings of the 2021 Workshop on Intelligent Cross-Data Analysis and Retrieval*. ICDAR '21. Taipei, Taiwan: Association for Computing Machinery, 2021, pp. 57–61. ISBN: 9781450385299. DOI: 10.1145/3463944.3469102. URL: https://doi.org/10.1145/3463944.3469102.

[30] Tor-Arne Schmidt Nordmo et al. "Njord: a fishing trawler dataset." In: *Proceedings of the 13th ACM Multimedia Systems Conference*. MMSys '22. Athlone, Ireland: Association for Computing Machinery, 2022, pp. 197–202. ISBN: 9781450392839. DOI: 10.1145/3524273.3532886. URL: https://doi.org/10.1145/3524273.3532886.

[31] Diego Ongaro and John Ousterhout. *In search of an understandable consensus algorithm (extended ... - raft*. June 2014. URL: https://raft.github.io/raft.pdf. (accessed: 17.03.2024).

[32] Aril Bernhard Ovesen et al. "File System Support for Privacy-Preserving Analysis and Forensics in Low-Bandwidth Edge Environments." In: *Information* 12.10 (2021). ISSN: 2078-2489. DOI: 10.3390/info12100430. URL: https://www.mdpi.com/2078-2489/12/10/430.

[33] pipermerriam. *gm - web3.py*. Accessed on 10/05/2024. 2023. URL: https://web3py.readthedocs.io/en/stable/.

[34] Karthik Sai Puranam et al. "Anatomy and Lifecycle of a Bitcoin Transaction." In: *SSRN Electronic Journal* (Jan. 2019). DOI: 10.2139/ssrn.3355106.

[35] SatpalSing Rajput et al. "Agricultural Food supply chain Traceability using Blockchain." In: *2023 4th International Conference on Innovative Trends in Information Technology (ICITIIT)*. 2023, pp. 1–6. DOI: 10.1109/ICITIIT57246.2023.10068564.

[36] M. Reddy. *API Design for C++*. Elsevier Science, 2011. ISBN: 9780123850041. URL: https://books.google.no/books?id=IY29LylT85wC.

[37] Team Rocket et al. *Scalable and Probabilistic Leaderless BFT Consensus through Metastability*. 2020. arXiv: 1906.08936 [cs.DC].

[38] Sarwar Sayeed, Hector Marco-Gisbert, and Thomas Caira. "Smart Contract: Attacks and Protections." In: *IEEE Access* 8 (Jan. 2020), pp. 24416–24427. DOI: 10.1109/access.2020.2970495.

[39] Ian Sommerville. *Engineering Software Products: An Introduction to Modern Software Engineering*. Global Edition. Pearson, 2020. ISBN: 1292376341,9781292376349.

[40] Matplotlib Development Team. *Matplotlib: Visualization with Python*. Accessed on 10/05/2024. 2012. URL: https://matplotlib.org/.

[41] Remix Project Team. *Remix - Ethereum IDE*. Accessed on 10/05/2024. 2015. URL: https://remix.ethereum.org/.

[42] Solidity Team. *Solidity Programming Language*. Accessed on 10/05/2024. 2020. URL: https://soliditylang.org/.

[43] Enrico Tedeschi, Håvard D. Johansen, and Dag Johansen. "Trading Network Performance for Cash in the Bitcoin Blockchain." In: *the 8th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER*. INSTICC. SciTePress, 2018, pp. 643–650. DOI: 10.5220/0006805906430650.

[44] Enrico Tedeschi et al. "Mining Profitability in Bitcoin: Calculations of User-Miner Equilibria and Cost of Mining." In: *24th International Conference on Distributed Applications and Interoperable Systems (D AIS 2024)*. Vol. 14677. LNCS. 2024.

[45] Enrico Tedeschi et al. "On Optimizing Transaction Fees in Bitcoin Using AI: Investigation on Miners Inclusion Pattern." In: *ACM Trans. Internet Technol.* 22.3 (July 2022). ISSN: 1533-5399. DOI: 10.1145/3528669. URL: https://doi.org/10.1145/3528669.

[46] Enrico Tedeschi et al. "Predicting Transaction Latency with Deep Learning in Proof-of-Work Blockchains." In: *2019 IEEE International Conference on Big Data (Big Data)*. 2019, pp. 4223–4231. DOI: 10.1109/BigData47090.2019.9006228.

[47] A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem." In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. DOI: https://doi.org/10.1112/plms/s2-42.1.230.

[48] UNODC. "Fisheries Crime: transnational organized criminal activities in the context of the fisheries sector." In: (2016).

[49] Jean Vanderdonckt and Xavier Gillo. "Visual techniques for traditional and multimedia layouts." In: (June 1994). DOI: 10.1145/192309.192334.

[50] Mario Villamizar et al. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud." In: (Nov. 2015). DOI: 10.1109/columbiancc.2015.7333476.

[51] Jan Werth et al. "A Review of Blockchain Platforms Based on the Scalability, Security and Decentralization Trilemma." In: Mar. 2023. DOI: 10.5220/0011837200003467.

[52] Yuan Zhang et al. "A Blockchain-Based Traceability Model for Grain and Oil Food Supply Chain." In: *Foods* 12.17 (2023). ISSN: 2304-8158. DOI: 10.3390/foods12173235. URL: https://www.mdpi.com/2304-8158/12/17/3235.

A

Appendix A

A.1 Full transaction receipt

Field	Description
blockHash	The hash of the block in which the transaction was included.
blockNumber	The block number in which the transaction was recorded.
cumulativeGasUsed	Total amount of gas used in the block up until this transaction.
effectiveGasPrice	The gas price set by the sender, which may be adjusted by the network.
from	The address from which the transaction was sent.
gasUsed	The amount of gas used by this specific transaction.
logs	Array of log objects generated by this transaction.
logsBloom	Bloom filter for the logs of the block.
status	Transaction status, where '1' typically indicates success.
to	The address to which the transaction was sent.
transactionHash	Unique identifier for this transaction.
transactionIndex	The index position of the transaction in the block.
type	Type of the transaction (e.g., legacy, EIP-1559, etc.).
events	Detailed events triggered during the execution of the transaction.

Table A.1: Description of fields in a blockchain transaction receipt

```
{
    "blockHash": "0x963c173eec47e5c2dba50e75...",
    "blockNumber": "32644960",
    "cumulativeGasUsed": "221067",
    "effectiveGasPrice": "25500000000",
    "from": "0x6169fc23ca37fd5046811d64f041771d...",
    "gasUsed": "221067",
    "logs": [
        {
            "address": "0x8afd11bd6f56af0506...",
            "topics": [
                "0x54f0f7c20d24cc292c0d60...",
                "0x27ecf19a3173f4db4..."
            ],
            "data": "0x006169fc23ca...",
            "blockNumber": "32644960",
            "transactionHash": "0xb1d414fa5c8...",
            "transactionIndex": "0",
            "blockHash": "0x963c173eec47e5cd...",
            "logIndex": "0",
            "removed": false
        }
    ],
    "logsBloom": "0x00080000000...",
    "status": "1",
    "to": "0x8afd11bd6f56af0506784f6ed07c12fc0...",
    "transactionHash": "0xb1d414fa5c822ddf...",
    "transactionIndex": "0",
    "type": "2",
    "events": {
        "IDGenerated": {
            "address": "0x8afd11bd6f56af050...",
            "topics": [
                "0x54f0f7c20d24cc292c...",
                "0x27ecf19a3173..."
            ],
            "data": "0x006169fc23ca37fd...",
            "blockNumber": "32644960",
            "transactionHash": "0xb1d414fa5...",
            "transactionIndex": "0",
            "blockHash": "0x963c173e...",
            "logIndex": "0",
            "removed": false,
            "returnValues": {
                "0": "1805884447...",
                "1": "0x6169FC23c...",
                "2": "1714997190",
                "__length__": 3,
                "id": "180588444...",
                "ownerId": "0x6169FC2...",
                "timestamp": "1714997190"
            },
            "event": "IDGenerated",
            "signature": "0x54f0f7c20...",
            "raw": {
                "data": "0x006169...",
                "topics": [
                    "0x54f0f7c20d24c...",
                    "0x27ecf19a..."
                ]
            }
        }
    }
}
```

Figure A.1: Full example of a transaction receipt when generating new IDs. Very long values have been truncated, but not fields. For an explanation of what each field is see Table A.1.

A.2 Gas cost for generating IDs

Number of IDs	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Gas cost	255,387	444,139	632,736	821,338	1,009,941	1,198,542	1,387,148	1,575,751	1,764,359	1,952,968
Number of IDs	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Gas cost	2,141,575	2,330,186	2,518,799	2,707,410	2,896,025	3,084,638	3,273,255	3,461,875	3,650,491	3,839,112
Number of IDs	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Gas cost	4,027,735	4,216,355	4,404,980	4,593,602	4,782,230	4,970,859	5,159,484	5,348,116	5,536,749	5,725,378
Number of IDs	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
Gas cost	5,914,013	6,102,644	6,291,282	6,479,921	6,668,556	6,857,197	7,045,840	7,234,478	7,423,123	7,611,764
Number of IDs	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**
Gas cost	7,800,411	7,989,060	8,177,705	8,366,356	8,555,009	8,743,656	8,932,311	9,120,961	9,309,619	9,498,278
Number of IDs	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
Gas cost	9,686,931	9,875,592	10,064,255	10,252,912	10,441,577	10,630,236	10,818,904	11,007,573	11,196,235	11,384,907
Number of IDs	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**
Gas cost	11,573,579	11,762,245	11,950,920	12,139,589	12,328,267	12,516,945	12,705,617	12,894,298	13,082,981	13,271,657
Number of IDs	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**	**80**
Gas cost	13,460,341	13,649,020	13,837,707	14,026,395	14,215,077	14,403,768	14,592,460	14,781,145	14,969,840	15,158,528
Number of IDs	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**
Gas cost	15,347,225	15,535,923	15,724,614	15,913,315	16,102,017	16,290,712	16,479,417	16,668,113	16,856,820	17,045,529
Number of IDs	**91**	**92**	**93**	**94**	**95**	**96**	**97**	**98**	**99**	**100**
Gas cost	17,234,229	17,422,940	17,611,652	17,800,356	17,989,071	18,177,777	18,366,494	18,555,212	18,743,922	18,932,643

Table A.2: Gas cost for generating a different amount of IDs

A.3 Time to generate IDs based on Gas Price

Number of IDs	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Average**
Trial 1 - Time to generate IDs	4.30144	3.80096	4.07456	3.50992	3.47541	3.94581	3.53066	4.24715	4.06776	4.47861	**3.94323**
Trial 2 - Time to generate IDs	4.27566	3.28848	3.94156	4.04565	3.07451	4.04990	3.42715	4.86356	3.81089	3.88359	**3.86610**
Trial 3 - Time to generate IDs	4.61674	4.22037	4.09705	4.10302	3.51208	3.12337	3.97184	3.67041	4.51272	3.47959	**3.93072**
Average											**3.91335**

Table A.3: Time it took to generate a given amount of IDs from 1 to 10 using the base Gas Price of 25 nAvax on the Avalanche Fuji Testnet. Each time is rounded to 5 decimals and the test was done 3 times in total.

Number of IDs	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Average**
Trial 1 - Time to generate IDs	4.29332	4.13112	3.62570	4.23832	4.10993	2.61762	3.00923	2.95560	3.00949	2.98973	**3.49801**
Trial 2 - Time to generate IDs	2.66042	3.55103	2.97006	3.07858	3.30163	4.42033	2.94918	2.94597	2.83191	3.81706	**3.25262**
Trial 3 - Time to generate IDs	2.99225	3.28235	2.98525	3.57060	3.33617	2.63315	3.86704	2.99755	2.87252	3.35773	**3.18947**
Average											**3.31337**

Table A.4: Time it took to generate a given amount of IDs from 1 to 10 using a 20% increased Gas Price (30nAvax) from the base price of 25 nAvax on the Avalanche Fuji Testnet. Each time is rounded to 5 decimals and the test was done 3 times in total.

Number of IDs	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Average**
Trial 1 - Time to generate IDs	4.78492	2.98059	3.24754	3.25913	3.02481	3.11442	2.68376	3.06382	2.59348	3.06554	**3.18181**
Trial 2 - Time to generate IDs	2.50447	3.15845	2.92925	3.39483	3.13172	3.14611	2.60998	2.69517	2.86260	2.94864	**2.93813**
Trial 3 - Time to generate IDs	3.38503	2.71354	3.11042	3.12073	3.33756	2.39629	3.12239	2.99020	3.15958	2.74531	**3.00811**
Average											**3.04268**

Table A.5: Time it took to generate a given amount of IDs from 1 to 10 using a 50% increased Gas Price (37.5nAvax) from the base price of 25 nAvax on the Avalanche Fuji Testnet. Each time is rounded to 5 decimals and the test was done 3 times in total.

A.4 Time to fetch data from blockchain

Run number	1	2	3	4	5	6	7	8	9	10	**Average**
Trial 1 times	0.77634	0.09574	0.09140	0.09847	0.10417	0.09258	0.09059	0.09958	0.09679	0.09022	**0.16359**
Trial 2 times	0.53037	0.09788	0.08968	0.09117	0.09123	0.09576	0.09329	0.09582	0.08448	0.09391	**0.13636**
Trial 3 times	0.47618	0.09672	0.09146	0.10693	0.09181	0.09257	0.09710	0.09780	0.11716	0.09818	**0.13660**
Trial 4 times	0.47575	0.09836	0.08892	0.08482	0.08971	0.09679	0.09095	0.08812	0.09188	0.08563	**0.12910**
Trial 5 times	0.59788	0.09408	0.10954	0.10839	0.09452	0.09628	0.09329	0.09193	0.10114	0.10393	**0.14910**
Average											**0.14295**

Table A.6: Time to fetch data from blockchain with 0 seconds delay between requests, rounded to 5 decimals. For each of the 5 trials data was fetched 10 times from the blockchain.

Run number	1	2	3	4	5	6	7	8	9	10	**Average**
Trial 1 times	0.49416	0.09672	0.58593	0.09971	0.34419	0.09013	0.69143	0.09417	0.63916	0.10461	**0.32403**
Trial 2 times	0.38104	0.08289	0.60686	0.09821	0.38193	0.09947	0.39412	0.08423	0.33810	0.09427	**0.25612**
Trial 3 times	0.42308	0.08981	0.33962	0.33420	0.08216	0.41141	0.09009	0.72986	0.09402	0.33669	**0.29310**
Trial 4 times	0.08631	0.39228	0.08522	0.38794	0.08341	0.33939	0.08485	0.39287	0.41785	0.09282	**0.23630**
Trial 5 times	0.65504	0.09545	0.33211	0.14024	0.41514	0.08686	0.39955	0.08308	0.57309	0.09855	**0.28792**
Average											**0.27949**

Table A.7: Time to fetch data from blockchain with 3 seconds delay between requests, rounded to 5 decimals. For each of the 5 trials data was fetched 10 times from the blockchain.

Run number	1	2	3	4	5	6	7	8	9	10	**Average**
Trial 1 times	0.75017	0.36597	0.75755	0.40817	0.39744	0.76540	0.45282	0.43768	0.42820	0.43353	**0.51970**
Trial 2 times	0.44188	0.43871	0.43060	0.43526	0.67587	0.43154	0.38067	0.42512	0.45133	0.67331	**0.47843**
Trial 3 times	0.43423	0.38648	0.36453	0.57127	0.40594	0.42357	0.44562	0.61383	0.42389	0.43169	**0.45011**
Trial 4 times	0.38810	0.41490	0.34467	0.34783	0.34753	0.40016	0.40372	0.60796	0.39987	0.37994	**0.40347**
Trial 5 times	0.35479	0.43460	0.41691	0.42360	0.43267	0.77214	0.63400	0.39806	0.40872	0.40935	**0.46849**
Average											**0.46404**

Table A.8: Time to fetch data from blockchain with 5 seconds delay between requests, rounded to 5 decimals. For each of the 5 trials data was fetched 10 times from the blockchain.

A.5 Times taken to update status of IDs

Run number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Trial 1 times	3.94398	4.82210	3.16608	3.00885	4.74808	3.39571	2.79121	4.97620	2.62141	2.89268
Trial 2 times	2.72568	2.83859	3.54909	3.29411	4.68096	5.30088	2.86999	3.20916	2.08913	3.80802
Trial 3 times	4.56723	2.62864	4.66912	3.09266	5.06401	2.60206	2.73324	3.85052	2.75525	2.69498
Trial 4 times	3.62597	3.64994	3.63901	4.63560	5.01426	3.97030	2.99171	3.29634	3.14408	3.58011
Trial 5 times	2.73909	2.79801	3.37384	3.59904	3.84594	2.71881	3.39579	3.28422	4.62597	2.74939

Table A.9: Time taken to update the status of an ID in AquaTrace. This table showcases the measured times across 50 runs, illustrating the responsiveness of the smart contract to status update operations under various conditions.