University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

ARTIS - Art Tracking with IoT and Blockchains

Jordi Küffer
Zurich, Switzerland
Student ID: 20-714-051

Supervisor: Dr. Eryk Schiller, Dr. Thomas Bocek, Dr. Bruno Rodrigues, Prof. Dr. Burkhard Stiller
Date of Submission: September 6th, 2023

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Kurzfassung

Diese Arbeit befasst sich mit der Kombination von Internet of Things (IoT) und Blockchain-Technologien und konzentriert sich dabei auf die innovative Anwendung dieser Technologien im Bereich des Transports von Kunstwerken. Das zentrale Ziel ist die Einführung eines Systems, welches IoT und Blockchain nutzt, um die Überwachung und Verwaltung von Kunstwerken während des Transports zu verbessern.

Um dieses Ziel zu erreichen, wendet die Studie eine zweigleisige Methodik an. Zunächst wird eine umfassende Literaturrecherche durchgeführt, um ein grundlegendes Verständnis für die zugrundeliegenden Prinzipien aufzubauen. Anschließend wird ein angewandter Forschungsansatz ausgeführt, der die Entwicklung, Implementierung und Evaluierung eines Prototyps beinhaltet. Das Ergebnis dieser Forschungsarbeit ist ein funktionaler Prototyp, der den angestrebten Anwendungsfall unterstützt.

Das Ergebnis dieser Arbeit ist Artis, ein Prototyp, der den angestrebten Anwendungsfall unterstützt. Es wird jedoch festgestellt, dass der Prototyp noch weiter verfeinert werden muss, insbesondere im Hinblick auf den Schutz sensibler Daten und die Optimierung der Sensorgenauigkeit.

Der Wert dieser Arbeit liegt in der innovativen Verschmelzung von IoT- und Blockchain-Technologien, die einen neuen Weg zur Bewältigung von Herausforderungen im Bereich der Kunstwerkstransportation demonstriert. Dies legt die Grundlage für zukünftige Bemühungen, dieses Konzept zu einer produktionsreifen Lösung weiter zu entwickeln.

Abstract

This thesis delves into the convergence of the Internet of Things (IoT) and Blockchain technologies, focusing on the innovative application of these technologies within artwork transportation. The main goal is to introduce a groundbreaking system that capitalizes on IoT and blockchain to enhance the tracking and management of artwork during transportation processes.

In pursuit of this goal, the study adopts a dual-pronged methodology. A comprehensive literature review provides a foundational understanding of the underlying principles. Subsequently, an applied research approach is employed, culminating in designing, implementing, and evaluating a prototype tailored to the intricacies of artwork transportation.

The outcome of this thesis is Artis, a real-world prototype that effectively supports the targeted artwork tracking use case. However, it is acknowledged that further strides are needed to refine the prototype, particularly in safeguarding sensitive data and optimizing sensor accuracy.

The significance of this work lies in its innovative amalgamation of IoT and blockchain technologies, presenting a novel avenue for addressing challenges in the artwork transportation domain. By demonstrating the feasibility of such a system, this thesis lays the groundwork for future endeavors to advance this concept into a production-ready solution.

Acknowledgments

I am profoundly grateful for the invaluable guidance and unwavering support provided by my supervisors, Dr. Eryk Schiller, Dr. Thomas Bocek, and Dr. Bruno Rodrigues. Their profound insights, encouragement, and dedication have been instrumental in shaping the trajectory of this work.

I extend my appreciation to the entire Communication Systems Group (CSG), and Prof. Dr. Burkhard Stiller, whose collective expertise and collaborative spirit have enriched my research journey. Their vibrant exchange of ideas and constructive feedback has significantly contributed to the refinement of this study.

Finally, to my friends and family, your unwavering support has been my constant pillar of strength and has allowed me to thrive and succeed in my educational career.

Contents

Chapter 1

Introduction

The art trade is a diverse market with a wide range of stakeholders, including artists, collectors, auction houses, and art dealers, as well as a variety of middlemen, such as promoters, preservers, archivists, and curators, to name a few. Nevertheless, the entire art world revolves around art objects, which can also come in a large variety, from statues of different materials to canvases or even bananas nailed to walls. While museums own collections, they often display artworks from private collections or a variety thereof. Consequently, requiring trustworthy logistics partners who ensure the artwork is safely transported under optimal conditions and without damage presents an excellent opportunity for Internet of Things (IoT) sensors that can monitor temperature, humidity, vibrations, and other environmental factors. Additionally, blockchain technology can be leveraged to ensure the correctness of the transportation process for all stakeholders and, in combination with art trading, to ensure and document the transfer of ownership virtually and physically, thus bringing a degree of standardization to a largely unregulated and opaque market [10].

1.1 CERTIFY Project

CERTIFY is a multi-partner research project [10], dedicated to achieving a high level of security by developing a novel framework to manage security throughout the lifecycle of IoT devices. The project is scheduled to run from 1st October 2022 for 36 months and involves 13 partners from eight European countries. The Communication Systems Group (CSG) of the Department of Informatics (IfI) at University of Zurich (UZH) is part of CERTIFY and focusing on designing and developing a blockchain-based sharing infrastructure of security information, an Over-The-Air Patch Infrastructure and the pilot for tracking and monitoring of artworks. This thesis is part of the contributions by CSG to the pilot for tracking and monitoring of artworks.

1.2 Motivation

Artwork transportation has proven to be a logistical challenge in many ways [30]. Complying with regulatory laws of both the departure and destination country, taking out insurance on the artworks, using packaging that lowers the risk of damage, and many more. Recording all relevant transactions typically involves administrative paperwork and trust among the many parties involved [30].

The IoT has revolutionized how we interact with technology and data, allowing us to connect devices, sensors, and networks seamlessly. At the same time, blockchain technology has emerged as a powerful tool for securely managing data and transactions in a decentralized, tamper-proof way. Combined, these technologies offer exciting possibilities for creating new forms of digital art that are both secure and transparent.

IoT devices, such as sensors, can record data that can be used to monitor the environment around an artwork. By using blockchain technology to manage and store this data securely, stakeholders can ensure the integrity of artwork even in a trustless environment. The blockchain ledger can additionally improve documentation of ownership and custody by indefinitely storing transactions on the chain.

In this context, the thesis proposes a system for artwork tracking in combination with hardware and software deployed to provide automatic monitoring and management of artwork in a transportation scenario.

1.3 Description of Work

The work involves creating a secure and transparent system for tracking and monitoring the movement of a unique item, represented by a Non-Fungible Token (NFT) [49], using a combination of blockchain technology and IoT devices. The goal is to develop a system for monitoring and logging some environmental parameters, such as temperature and humidity, while transporting artwork. The system should be able to define alert thresholds for these parameters and notify the artwork sender, carrier, and recipient of any anomalies. The NFT is registered in a Smart Contract (SC) by the item owner or holder and contains relevant information and regulations, such as ObjectID [22], as required by the International Council of Museums (ICOM).

The IoT Board Administrator sets the board to its initial state, ready to receive data from the sensors. A private and public key profile is generated by creating a blockchain account and stored on the device, ensuring the private key remains confidential and cannot be leaked. The account address is associated with the NFT by the owner. This profile can update the state of the NFT by issuing transactions signed with the private key to the smart contract. This allows the NFT to be updated with events such as ”pick up at origin” and ”delivery at destination,” which are logged with relevant timestamps. Multi-approval operations certify changes in responsibilities between different actors, such as when the Sender transfers responsibility for the artwork to the Carrier. This helps to ensure the secure and transparent transfer of custody of the item.

Finally, with the transfer of custody, the behavior of the sensor is set from standby mode to constant monitoring mode, allowing for real-time tracking and monitoring of the item as it moves from one location to another. The system provides a secure and transparent way to track and monitor unique items' movement while ensuring their authenticity and ownership.

1.3.1 Thesis Goals

Research: The research report should include a review of the relevant literature concerning the technologies used and include existing solutions for blockchain-based artwork tracking.

Design on Solution Architecture: The report should include detailed specifications and requirements for the system, including the backend and frontend.

Solution Prototyping: This goal covers the implementation of a prototype of the artwork tracking system. This involves working with IoT devices, sensors, and other suitable technologies. The prototype should be functional and demonstrate the key features of the system.

Evaluation: The system evaluation involves testing the prototype in a (simulated) real-world artwork transportation scenario. The evaluation report includes an analysis of the system's performance and an assessment of its real-world usability.

1.4 Methodology

The methodology adopted in this thesis is strategically designed to comprehensively address the objectives outlined in Section 1.3.1. This approach can be categorized into two primary phases: a literature review phase and an applied research phase.

Literature review phase: This involves conducting a literature review to gather essential knowledge about fundamental concepts and related work in artwork tracking and blockchain and IoT. This review provides an overview of related work on tracking solutions, summarized in Section 3.6. The knowledge gained will serve as a basis for the design and development of the proposed system.

Applied research phase: The applied research phase is composed of creating and evaluating a prototype for the proposed system. It includes the design, implementation, and evaluation described below.

Design: The system architecture design is based on a simplified artwork tracking scenario. The prototype is required to support this scenario concerning the goals defined in Section 1.3.1. The architecture includes all necessary components and interactions to support the use case. This phase delivers a documentation of this design presented in Chapter 4.

Implementation: The output of the previous phase needs to be implemented as a prototype system. The implementation process and details regarding the implementation of features are reported in Chapter 5.

Evaluation: In the final phase, the delivered prototype of the implementation phase is evaluated. The evaluation includes a cost and performance analysis outlined in Section 6.1, a security analysis in Section 6.2, and a test run in a simulated artwork tracking scenario in Section 6.3. The results are discussed in Section 6.4.

1.5 Thesis Outline

Chapter 1 has provided introductory and motivational information. Chapter 2 gives a high-level overview of the theoretical background of the thesis. In Chapter 3, we discuss existing papers on the topic and elaborate on the current state-of-the-art regarding artwork tracking. This information is built upon in Chapter 4 when we present the architecture and design of our solution in detail. The implementational aspects of the solution are presented in Chapter 5. The implemented system is then evaluated in Chapter 6, and a summary and conclusions are given in Chapter 7. Furthermore, it addresses the potential for future work and suggests opportunities for improvement of the proposed system.

Chapter 2

Fundamentals

This chapter aims to introduce valuable background knowledge of the fundamentals built upon in this thesis.

2.1 Internet of Things

IoT technology does not have a single unique definition. However, Madakam et al. [27, p. 165] defines the term Internet of Things as follows:

> "An open and comprehensive network of intelligent objects that have the capacity to auto-organize, share information, data, and resources, reacting and acting in face of situations and changes in the environment"

While the internet in the traditional sense is about the data created by people, IoT is about data created by things. A practical example of this would be a smart heating system for a vacation home. Such a system is able to record and store data about environmental parameters such as temperature or humidity and make this data accessible in real-time on a smartphone through the internet. It is also possible to interact with this system through your phone. For example, raising the temperature of a winter vacation home the night before arrival would be possible. This thesis aims to utilize an IoT device capable of recording and storing environmental parameters to monitor an artwork in transit.

2.2 Blockchain

Blockchain technology has long been an evolution and promising advancement in distributed and decentralized systems. It describes a ledger that can be either distributed (permissioned) or decentralized (permissionless), tamper-evident, tamper-resistant and usually without a central authority. This technology allows a community of users to record transactions that cannot be changed once published [55].

These properties can reduce the importance of trust among single parties, as the consensus of the whole network is necessary for a transaction to be published. This is why blockchain has been able to digitize processes that previously required trust in a central authority. The most prominent are cryptocurrencies like Bitcoin and Ethereum [33, 7].

2.2.1 Smart Contracts

Some blockchains can be extended and leveraged by smart contracts, essentially collections of code and data deployed on the blockchain network using cryptographically signed transactions. Nodes within the blockchain network execute the smart contract, and the results of execution are recorded on the blockchain. Users can create transactions that send data to public functions offered by a smart contract. The smart contract then executes the appropriate method to perform a service. Since the code is on the blockchain, it is tamper-evident and resistant, making it a trusted third party. Smart contracts can perform various functions such as calculations, storing information, exposing properties, and automatically sending funds to other accounts [55].

Non-Fungible Token

An example of a smart contract standard would be the ERC-721 for NFTs on the Ethereum network [12]. NFTs are digital assets stored on a blockchain and represent a unique item or asset, such as a piece of artwork.

2.3 Tracking and Tracing

Tracking and tracing of logistic networks is considered an important issue [40]. In this context, tracking refers to collecting and managing information about the current location and status of a product or delivery item [40]. Tracing on the other hand looks back in time and refers to the storing and retaining of a product or item's history [45].

Chapter 3

Related Work

This chapter provides an overview of the existing literature on art tracking with IoT and blockchains, highlighting previous research, methods, and findings related to the research question. Lastly, we discuss the reviewed literature and conclude the chapter by identifying research gaps.

3.1 Artwork Conservation

When it comes to preserving artworks, several factors play a role. This includes their exposition of human factors and environmental variations [24]. More specifically, factors that can hamper artwork integrity include humidity, temperature, light, pollutants, and microbiological organisms [39]. Usually, temperature and humidity are the most sensitive parameters [39]. Monitoring such factors can be of significant importance when it comes to artwork preservation and health [8]. In addition to the conservation of artworks in museums or galleries, there is also the aspect of ensuring the safety of artwork during transportation.

3.2 Artwork Transportation

Collections of artworks have been exhibited for generations. More often than not, artworks are showcased at different venues worldwide. The dangers imposed on artwork during transportation have been thoroughly researched and led to the development of innovative packaging and other safety measures [30]. Today, there exist a number of companies that specialize in the transportation of artwork [23, 21, 54]. The advertised solution often involves shock-absorbing as well as climate-controlled packaging. Even though these packaging solutions have been adequately tested, many reviewed companies rely on customer trust regarding their effectiveness in real-world applications. This presents an opportunity for monitoring systems leveraging the potential of emerging technologies to improve the artwork transportation process further.

3.3 Artwork Monitoring Systems

Technological advancements have made it possible to study the impacts of transportation on the integrity of artwork in more detail. Numerous studies have been conducted regarding this matter. One study utilized a small logging device to gather information on shock and vibrations generated during transportation [38]. The study found that during the several dozen transportations monitored, the artwork suffered significant shock and vibration, even though the latest packaging techniques and appropriate means of transportation were used. This demonstrates that continuous monitoring of these parameters can be useful when verifying the integrity of the artwork after transportation.

In this context, [24] proposes a real-time system that collects information about the artwork's environment and its safety conditions. This information can then be used to determine the quality of the conditions of artworks. The system uses a low-cost IoT node that can operate with very low power consumption, thus allowing for the realization of pervasive monitoring systems. This proposed system demonstrates how violations can be detected as they happen or have already happened.

The study by [32] goes one step further and proposes a proactive approach to potential violations. The so-called PACT-ART architecture employs advanced computing techniques like data mining and business process intelligence to predict the future state of the process. PACT-ART can then point out any possible violations and recommend actions to mitigate the misbehavior. [9] expands on this and presents a system allowing for continuous risk assessment during storage, handling, transport, and exhibition.

3.4 Artwork Management and Documentation

Correct and secure information about artworks is one of the main concerns in the art world [25]. New technologies, such as blockchain, have been proposed as promising solutions to increase artworks' transparency, traceability, validity, and provenance [41]. Especially smart contracts and NFTs have demonstrated the potential of blockchain technology to revolutionize the art world.

Wang et al. [51] has shown the benefits of NFTs protecting digital assets by proving their existence and ownership. Malik et al. [28] has suggested the application of NFTs beyond digital art, proposing to physically tag an IoT device to an artwork or sculpture to transfer and track ownership. The IoT device is designed such that an attempt to tamper with the device will result in a blockchain record. This would potentially reduce intermediaries by providing a verifiable certificate that proves ownership, custodial history, and authenticity of a physical asset.

Platforms such as artory.com [3], 4art-technologies.com [1], and verisart.com [50] have already started to deploy blockchain and are offering a way to digitally register an artwork, introducing transparency and authenticity to the artwork, its history, and provenance [25].

Wang et al. [52], for example, developed a blockchain-based trading system for artworks that uses NFTs to represent physical artworks introducing traceability, irreversibility, and

transparency into the art market. In this regard, Vairagade et al. [49] has proposed an advanced NFT Minter for a blockchain-based artwork trading system. Besides trading, the work by Yeh et al. [56] developed an artwork rental system based on blockchain technology. The work is a proposal for a complete application of blockchain in the field of art leasing, which allows renters to securely and transparently browse and rent the available artworks.

3.5 Applications of Blockchain and IoT

Supply Chain Management (SCM) is another area where IoT is being used to improve processes. Similar to monitoring the environment around an artwork, IoT is being used in SCM to monitor the product state to ensure the right quality [5]. Another technology that is innovating SCM is using blockchain. The combination of these two technologies in industrial systems and supply chains has been a "hot trend" in recent years [26]. Blockchain presents an opportunity to build trust with its unique immutability characteristic. This can be used to improve documentation and traceability of physical assets and ensure the authenticity of the asset and collected data. The work that follows has already been established in this context.

Bocek et al. [6] present a solution to monitor relevant environmental data while transporting medical products. Upon delivery, the collected data is checked for compliance by a smart contract and then stored on the blockchain. There, the data is immutable and verifiable by any party. The proposed system comprises backend, frontend, and IoT sensor devices. The architecture and components of the system are shown in Figure 3.1. In addition to the Ethereum blockchain network, modum.io uses a relational database to store raw temperature data and user credentials. The mobile clients download the temperature data via Bluetooth from the sensors and submit it to the server, which sends it to the SC to evaluate the regulatory compliance and store the result on-chain.

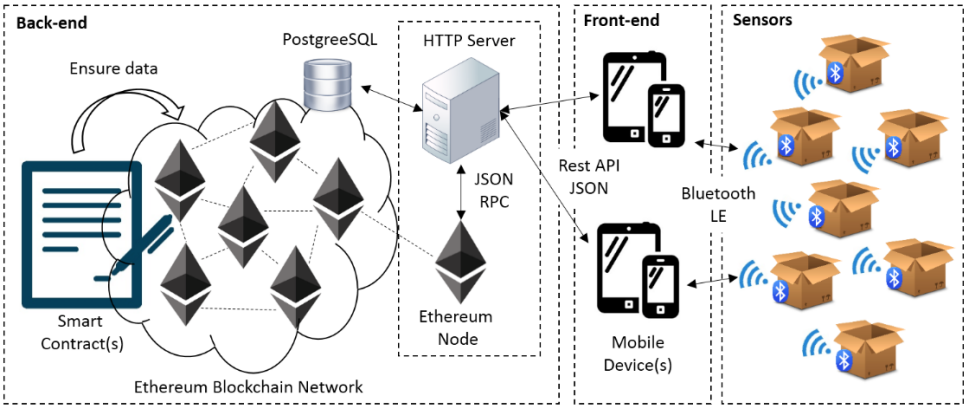


Figure 3.1: Modum.io AG Blockchain Architecture [6]

authena.io [4], is a Swiss-based company that provides a platform for tracking and verifying the authenticity of physical assets using blockchain technology and IoT. Their platform offers three main products:

- AUTHENA SHIELD: A tamper-proof end-to-end authenticity solution.
- AUTHENA L1VE: Real-time tracking of location and environmental conditions across countries and distribution channels.
- AUTHENA M3TA: Creating a secure link between physical product and its twin in the Metaverse.

According to [19], the system developed by authena can track the location as well as environmental data of an asset. This data is then stored securely and traceably on the blockchain. Unfortunately, the system is not open source, and the company provides no technical insight regarding the system's architecture.

everledger.io [17], is a digital transparency company providing technology solutions to increase transparency in global supply chains. They use blockchain to track and verify the authenticity of high-value assets such as diamonds, wine, and art. Their platform allows users to track the entire lifecycle of an asset, from production to ownership, using a secure and transparent blockchain-based ledger. Everledger mainly focuses on fraud detection, verification, and provenance records by issuing digital certificates stored on a private blockchain.

3.6 Discussion

This literature review has shown that the art world's conservation of artwork remains a concern. This concern is increased when it comes to the transportation of artworks. The integrity and health of an artwork depend on a number of factors, including the environment surrounding the artwork. Besides shock and vibration, temperature and humidity are suggested to be the most sensitive parameters.

With new technologies like IoT, it becomes possible to monitor environmental parameters continuously. Multiple papers have already demonstrated the potential benefits of artwork monitoring [32, 9, 24, 38]. These benefits include the verification of packaging techniques and registration of any potential disturbances to the environment that could damage an artwork. We have also seen that the gathered data from artwork monitoring can be used to predict future violations and suggest actions to prevent damage.

On the other hand, we have reviewed the existing literature on blockchain opportunities in the art world. The main benefits include increased transparency and traceability by generating a secure chain of ownership and simplified verification of authenticity.

When combining the two technologies, we have seen substantial research in the area of SCM. Within SCM, subprocesses often involve the transportation of assets while ensuring their integrity and health. This can be very similar to the requirements of artwork transportation. However, little to no literature was found on the combination of IoT and blockchain in artwork logistics. The reviewed literature suggests that the benefits of both technologies can be combined to improve the current process of artwork transportation.

Chapter 4

Architecture and Design

This chapter describes the application scenario in section 4.1 and provides an overview of the system architecture in section 4.2.

4.1 Scenario Definition and Goal

This project is designed for a scenario involving the transportation of sensitive pieces of artwork from one place to another. More specifically, a *sender* is handing custody of the artwork to a *carrier* who is responsible for transporting the artwork safely to a *recipient*. This means the *carrier* controls the transportation environment so that any damages can be prevented. Upon arrival, the artwork's custody is transferred to the *recipient*, and the process is finished.

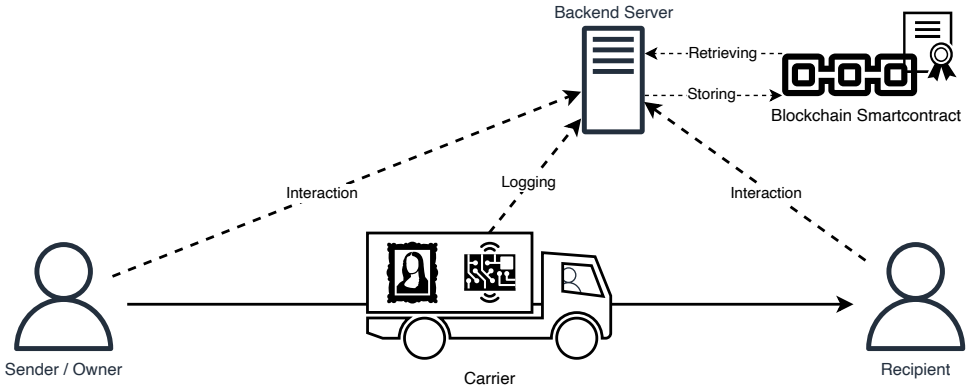


Figure 4.1: Defined scenario for the system to consider

The *sender* of the artwork registers the artwork on the system and inputs relevant information to the application. The system returns an artwork identifier which can be used to retrieve details about the artwork. Before the item is handed over for transportation, the *sender* registers the *carrier* and the *recipient* by adding their corresponding wallet addresses to the system. This allows the other actors to retrieve details about the artwork and interact with the system. A logging device is also registered in the same manner,

which records and stores environmental data, precisely temperature and humidity, during transportation to be audited by the actors. The logger is set up with a defined threshold specific to the artwork. It will report any violation of this threshold to the system automatically. Once the actors are registered, and the logger is set up, both the *carrier* and the *sender* must verify the integrity of the artwork and initiate the request for a status change to "in transit". After which the *carrier* will transport the artwork to its destination where the integrity of the artwork is verified by both the *carrier* and *recipient*, and the status is changed to "delivered." If any violation occurred during the transportation the timestamp of the most recent one will be visible for all associated actors in the system. Any past violations can be checked in the device's database or in the system's transaction history.

Goal

The goal is to create a minimal working system for artwork tracking, considering the following abstractions and the defined scenario. The system should allow the creation of an NFT associated with a piece of artwork and storing and retrieving data on the NFT. The actors can interact with the SC via a Representational State Transfer (REST) Application Programming Interface (API) that is hosted on the internet. This API provides an abstracted form of interaction that does not need a low-level understanding of the underlying blockchain components. Additionally, the system includes a programmed logging device to record humidity and temperature data and report any deviations of a predefined threshold to the API. Finally, the system should provide a user interface that allows actors to log in with their Ethereum account and interact with the system.

Summary of Abstractions

The real-world scenario of transferring custody of artwork for transportation is naturally much more complex and involves more actors [30]. However, in this work, we are making the following abstractions:

- The sender of a specific artwork is also the owner of the artwork
- When considering the transportation of the artwork back to the location of origin, the *sender* and *recipient* are the same.
- Only temperature and humidity data are recorded by the logger
- The authenticity and integrity of the artwork can be verified by any of the actors

4.1.1 Actors

As already introduced in the section Scenario Definition and Goal, the system is designed for three actors. Each of the actors is represented by an Ethereum account which is used for authentication and authorization. The actors are defined as follows.

- **Sender**: A person or institution responsible for dispatching the artwork from a departure location to a destination. This Actor is also considered the *owner* of the artwork. In this context, they can be used interchangeably. The responsibilities of this actor include the initial registration of the artwork as an NFT, the holding of the NFT in their wallet, and the setup of the IoT device.

- **Carrier**: A transportation company responsible for safely transporting artwork from one location to another. The *carrier* is committed to delivering the artwork without any damage or alteration. They are also responsible for verifying the integrity of the artwork and the logger with the *sender* and *recipient* upon departure and arrival, respectively.

- **Recipient**: A person or institution receiving the artwork. They share the responsibilities of the *carrier* to verify the integrity of the artwork upon arrival.

4.2 Overview

Since the system is considering a particular scenario, This chapter aims to provide a high-level overview of the architecture of the system while later going into detail about its components and actors. From now on, we will call the artwork tracking system `ARTIS`.

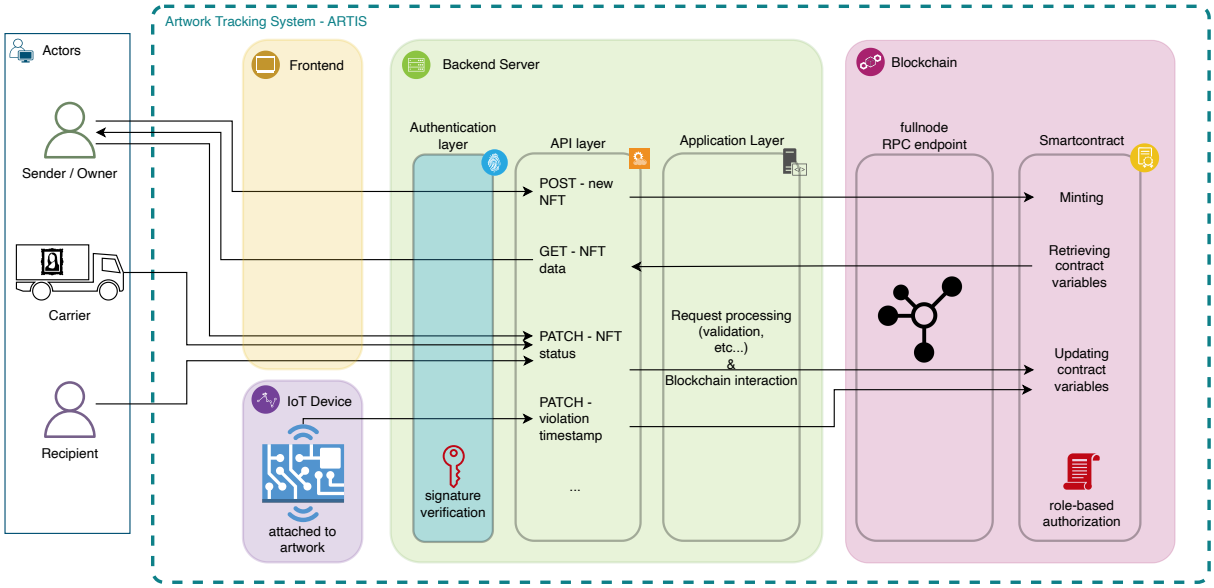


Figure 4.2: System Architecture

This architecture was designed using a top-down approach, with the initial step of building a server application that can interact with the SC. In the implementation phase, we added the specific functionality needed to support the defined scenario. In Figure 4.2 you can see the final architecture of the system.

Function	unregistered	owner	recipient	carrier	logger
Get artwork data	✗	✓	✓	✓	✗
Update roles	✗	✓	✗	✗	✗
Update violation timestamp	✗	✗	✗	✗	✓
Update requested status	✗	✓	✓	✓	✗
Update current Status	✗	✗	✗	✗	✗
Update data fields	✗	✓	✗	✗	✗

Table 4.1: Available permission sets to registered roles

4.2.1 Technical Components

Apart from the specified external actors of the system, there are also technical components to consider. The main components of the system are visually separated by colored areas in Figure 4.2.

Blockchain

The blockchain component of the system consists of two sub-components. A Remote Procedure Call (RPC) endpoint is needed to interact with the blockchain. The backend server can use this endpoint to submit transactions that are used to interact with the SC. For this project, we will consume a managed service to access the Ethereum network and not run our own Ethereum node.

The second component is the SC itself. It will be in charge of defining the NFT of the artwork. The functionality provided by the SC is defined as follows:

- **Minting**: A new user can create a NFT of their artwork by entering relevant information about it and the outstanding transportation process.
- **Retrieving data**: Users are able to retrieve stored data from the SC.
- **Updating data**: Users can update the stored data.

The SC enforces a role-based authorization that controls the available permission sets of each user. Specifically, this means read access will be granted to *carriers* and *recipients*, while write access is reserved for the *owner*. It will also handle the multi-approval process necessary for updating the status and changing custody upon transportation of the artwork. An overview of the available permission sets of each actor can be found in Table 4.1

Backend Server

The backend server is designed to provide an abstracted form of interaction to the SC by exposing a REST API. Like the actors and the logger, the backend server is also represented by an Ethereum account. This account is used to deploy the SC and is the sole point of access to it. This account issues every transaction. The backend server is also responsible for handling authentication designed to prove that incoming requests are made by the actors who are owners of the Ethereum accounts associated with specific roles.

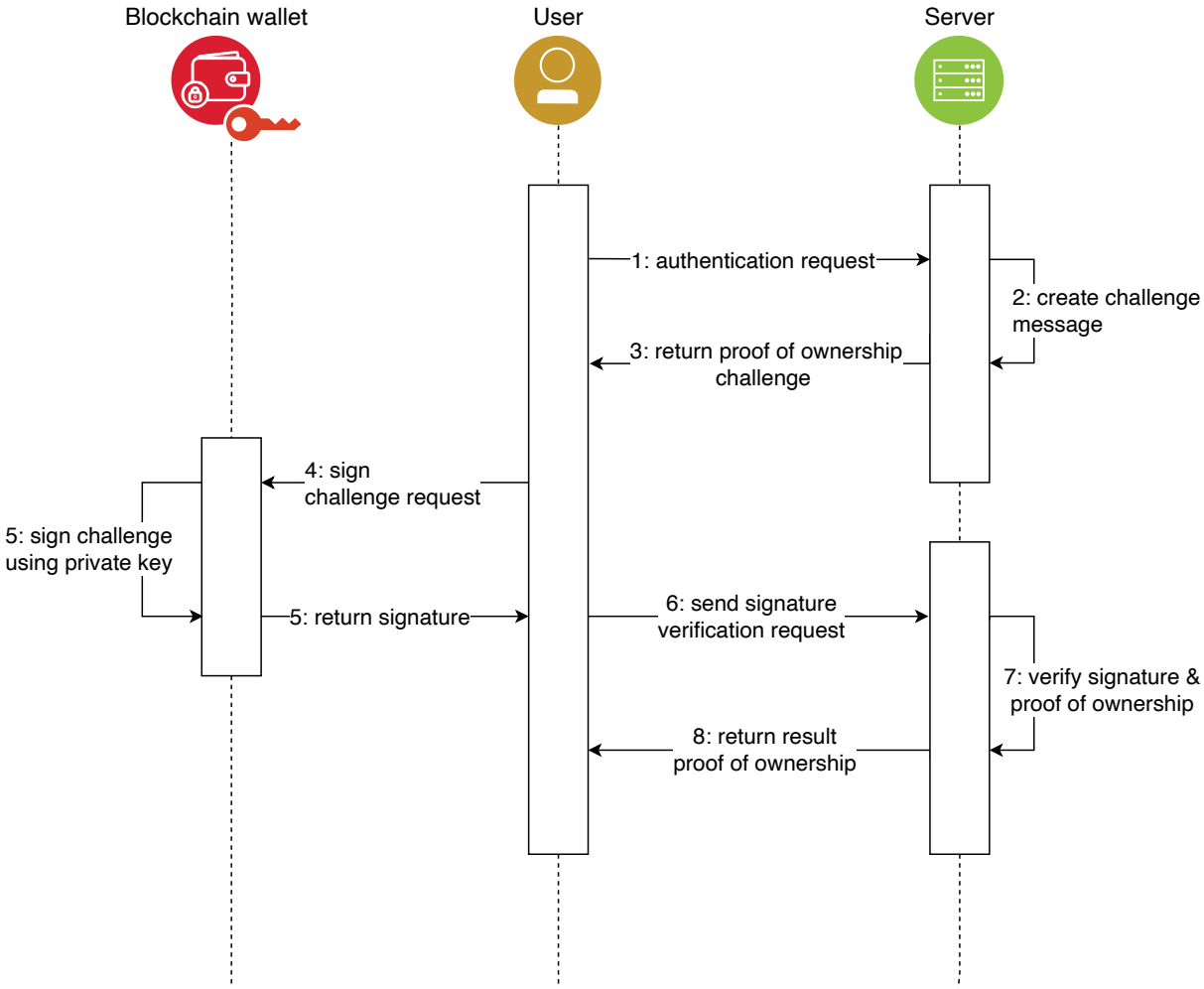


Figure 4.3: Authentication flow

Authentication The authentication flow visualized in Figure 4.3 uses signature verification of cryptographically signed messages to prove ownership over an Ethereum account. To do this, a challenge message is generated upon authentication request. This message contains relevant information to the session, such as the start of the session, the duration of the session, and for which account the session is created. The message is then signed by the private key of the requester's wallet and sent back. The signature can then be verified against the recovered address. If the requested address and recovered address match, this

proves that the request was made by an actor who owns the private key corresponding to the account, assuming that the private key was not compromised.

Features In addition to enabling the features of the SC, and managing authentication, the server will validate incoming requests and provide meaningful error messages to the user.

IoT Device

The logging device records temperature and humidity data while the artwork is in transit. The recorded data is stored in a local database and evaluated the data against a predefined threshold. The timestamps of any violations are reported to the system. If necessary the recorded data can be accessed after by connecting to the logger. The device will interact with the system via the API provided by the backend server. It will not interact with the SC directly. However, the logger is also represented by an Ethereum account and equipped with a key pair. This will be used for authentication by signing a message.

Frontend

We included a user interface in the system to simplify the interaction with the system. The frontend presentation layer is a web application designed to facilitate the authentication process by integrating popular blockchain wallets to sign messages and allow login with wallet functionality. Therefore it should afford to easily guide the user through the authentication flow and provide a graphical interface to the backend server API.

For this purpose, the frontend will consist of three main views: login view, home view, and detail view. The login view mainly consists of a welcome screen and a login button. The authentication process described in Section 4.2.1 will be initiated upon pressing the login button. After successful login, the user is redirected to the home view. The home view allows the user to quickly see which artworks are associated with this account. Additionally, this view will afford to register a new artwork with the system. Lastly, the user can navigate to the detail view by clicking on a specific artwork. The detail view displays all the information about the artwork and lets the user update information. The status of the artwork as well as any violations are also displayed in the detail view.

Chapter 5

Implementation

This chapter highlights the implementational aspects of the system. Introducing the technologies used to build the components. Each section title refers to its respective GitHub repository of the `ARTIS-project`[1] GitHub organization.

5.1 artis-smartcontract

The SC was written in Ethereum's native programming language Solidity [42]. The contract was developed using hardhat [20], a development environment for Ethereum. Hardhat makes developing, compiling, testing, and deploying smart contracts easy. The SC has been deployed to the Ethereum testnet Sepolia [14] during development but is intended to be deployed on the mainnet in production. The Ethereum blockchain was selected due to the research group's and the author's prior experience.

When the SC is first deployed the `smartcontractAdmin` variable is set to the deploying address. This address will be the only point of access to any SC function. It is also the account used by the server to interact with the SC and sign transactions.

```
1 constructor() ERC721("Artwork", "ARTIS") {
2   smartcontractAdmin = msg.sender;
3 }
```

Listing 5.1: SC constructor function

As seen in Listing 5.1 the SC extends the ERC721 [12] contract which provides the NFT interface.

5.1.1 Data Structures and Events

We will explain the data structures used in more detail to understand the rest of the contract.

[1] https://github.com/artis-project/

Structs

To satisfy the artwork tracking use case three additional structs have been defined as seen in Listing 5.2. The `ArtworkData` struct contains all information about a specific artwork. This includes the addresses of each actor as well as the violation timestamp and the status of the artwork. The status field in turn contains two fields as defined in the `Status` struct. They are used to support the multi-approval process when it comes to changing the status of an artwork. To match the physical artwork with the NFT we also included an objectID field. If needed, the `ArtworkData` struct can be extended with more data fields.

The `StatusApprovals` struct stores which of the actors has already approved a status change. The logic behind this multi-approval process will be explained in detail later.

```
struct ArtworkData {
  uint256 id;
  string objectId;
  address carrier;
  address logger;
  address recipient;
  Status status;
  uint256 violationTimestamp;
}

struct StatusApprovals {
  bool carrier;
  bool owner;
  bool recipient;
}

struct Status {
  string currentStatus;
  string requestedStatus;
}
```

Listing 5.2: SC structs

Enums

The SC defines one enum that defines the different status values that exist. Enums help during development as they prevent misspellings or other types of errors.

```
enum StatusValue {
  IN_TRANSIT,
  TO_BE_DELIVERED,
  DELIVERED,
  NONE
}
```

Listing 5.3: SC enums

Mappings

The `artworks` mapping is used to map artwork ids to `ArtworkData` structs. This mapping is used to store the references to each new artwork. Similarly, the `approvals` mapping is used to store and manage the approval data of an artwork. Simply put, this mapping can answer the question of who has approved a certain status of a specific artwork.

```
mapping(uint256 => ArtworkData) internal artworks;

mapping(uint256 => mapping(StatusValue => StatusApprovals))
  internal approvals;
```

Listing 5.4: SC mappings

e.g. `approvals[1][StatusValue.IN_TRANSIT].carrier` is `true` if and only if the carrier has already approved to change the artwork status of artwork 1 to IN_TRANSIT.

Events

The contract defines three events (Listing 5.5). The `Updated` event is emitted whenever the data associated with an artwork has changed. The `StatusApproved` event is emitted whenever the multi-approval process (Figure 5.1) has been successful. If an approval is still missing the `ApprovalMissing` event is emitted. Users can subscribe to these events and be notified whenever an event is emitted.

```
event Updated(
  uint256 indexed tokenId,
  ArtworkData newData,
  address owner,
  StatusApprovals approvals
);

event StatusApproved(
  uint256 indexed tokenId,
  StatusApprovals approvals,
  address approver
);

event ApprovalMissing(
  uint256 indexed tokenId,
  string requestedStatus,
  address missingApproval
);
```

Listing 5.5: SC events

5.1.2 Functions and Modifiers

To support the functionality designed in Section 4.2 the contract implements three main functions, modifiers, and helper functions. However, we will not go into detail about the

helper functions.

Function Modifiers

In Solidity, we can define function modifiers, that can be used to prepend functionality to a function. If a function modifier is used in a function the code in the modifier is executed beforehand. For instance, this can be used to perform a check before a function is run. Function modifiers can be reused on multiple functions and thus introduce cleaner and less redundant code.

```
modifier onlyAdmin() {
  require(
    msg.sender == smartcontractAdmin,
    "only accessible by smartcontractAdmin wallet 403"
  );
  _;
}

modifier read(address sender, uint256 tokenId) {
  require(
    ownerOf(tokenId) == sender ||
    carrierOf(tokenId) == sender ||
    recipientOf(tokenId) == sender,
    "sender is not authorized 403"
  );
  _;
}

modifier write(address sender, ArtworkData memory data) {
  if (data.violationTimestamp != 0) {
    require(
      sender == loggerOf(data.id),
      "only logger is allowed to add a violationTimestamp 403"
    );
  }
  require(
    bytes(data.status.currentStatus).length == 0,
    "currentStatus is updated automatically 403"
  );
  if (sender != ownerOf(data.id)) {
    require(
      bytes(data.objectId).length == 0 &&
        // address(1) is submitted if the field did not change
        data.carrier == address(1) &&
        data.recipient == address(1) &&
        data.logger == address(1),
      "only owner has write permissions 403"
    );
  }
  _;
}
```

Listing 5.6: SC modifiers

The modifiers in Listing 5.6 are used to check if a caller is authorized to call a specific function. If the call is forbidden the transaction is reverted with an appropriate error message. The last three characters of each error message represent the corresponding Hypertext Transfer Protocol (HTTP) status code. This status code is directly forwarded by the server API in case of an error.

To ensure that all SC interaction is done through the server API we modified every public function with the `onlyAdmin` modifier. This modifier reverts any transaction which is sent from a different account than the specified admin account.

The `read` modifier is additionally added to each function that reads data from the SC. We check that the wallet that authenticated a request to the server API is registered as an actor with read permissions.

Similarly, we defined the `write` modifier to check that certain `ArtworkData` fields are only modified by the owner or logger respectively. The `currentStatus` field cannot be modified at all, it is updated automatically once the actors approved a status change. The detailed permission sets of each actor can be found in Table 4.1

The contract makes use of one additional modifier called `exists` which simply checks if a given tokenId exists.

safeMint

This function is called to create a new NFT. This process is called ”minting”. The safeMint function in an ERC721 contract creates and assigns ownership of a new NFT to a designated recipient.

```
1  function safeMint(address to, ArtworkData memory data)
2    public
3    onlyAdmin
4  {
5    totalSupply.increment(); // start ids at 1
6    uint256 tokenId = totalSupply.current();
7    _safeMint(to, tokenId);
8    ArtworkData memory newArtwork = ArtworkData({ ... });
9    artworks[tokenId] = newArtwork;
10 }
```

Listing 5.7: SC safeMint function

Our implementation is rather straightforward, first, we increment the counter variable, which keeps track of the total supply. We also use this number as a token identifier which is equal to the artwork Identifier (ID). With this unique ID, we call the `_safeMint(to, tokenId)` function of the ERC721 contract provided by openzeppelin [34]. In Line 8 of Listing 5.7, we initialize a new ArtworkData instance with the initial data, which is partially provided by the user via a function parameter. We assign the values of the objectID property, the actors, and the logger properties. The violation timestamp, ID, and status properties are ignored and set to a default value. Finally, the artwork data is associated with the newly minted artwork in the `artworks` mapping.

updateArtworkData

The data associated with an artwork can be updated by calling this function. Upon successful update, the function does not return anything but emits an event with the updated values. This is based on a limitation of Solidity where it is impossible to return an object from a state-changing function. Additionally, this function allows you to update multiple fields at once. This has some advantages:

simplified interface: Using only one update function instead of exposing an update function for each field reduces the number of public functions. This results in a more understandable and usable contract.

atomic updates: If an error occurs during the execution of this function the transaction as a whole is reverted, resulting in the initial state of the contract. This maintains data consistency and prevents partial updates.

reduced costs: The possibility to update multiple fields at once reduces the number of transactions that need to be submitted and in turn reduces execution costs.

```
function updateArtworkData(ArtworkData memory data, address sender)
  public
  onlyAdmin
  exists(data.id)
  write(sender, data)
{
  if (data.violationTimestamp != 0) { ... }
  if (bytes(data.status.requestedStatus).length != 0) { ... }
  if ((bytes(data.objectId).length != 0)) { ... }
  if (data.carrier != address(1)) { ... }
  if (data.recipient != address(1)) { ... }
  if (data.logger != address(1)) { ... }

  emit Updated( ... );
}
```

Listing 5.8: SC updateArtworkData function

The `updateArtworkData` expects the updated data as a parameter of the type ArtworkData. Solidity expects every field of the `data` parameter to be populated. This is why we check for each field if the value should be updated and execute the according update. For string fields, this check is done by checking the length of the incoming string. An empty string with length 0 indicates that the user did not update this field. The same check is not possible with addresses. This is why we used the address 0x00...001 to mark an empty address update field. The zero address (0x0) is reserved to remove an address from a field.

The code to update artwork properties differs in complexity. For the `violationTimestamp` field it is as simple as assigning the new timestamp to the corresponding artwork:

```
    artworks[data.id].violationTimestamp = data.violationTimestamp
```

This works identically for updating the roles and the objectID. When updating the logger address we implemented an additional check to prevent updates when the artwork is IN_TRANSIT.

```
    require(
        !(artworks[data.id].status.currentStatus.equal("IN_TRANSIT")),
        "logger cannot be updated in transit 403"
    );
```

Updating the transportation status The logic behind updating the status (Figure 5.1) is more complex. The reason for this is the multi-approval feature of this contract. To satisfy our use case, the actors must agree on the transportation status of the artwork. To ensure this agreement, each affected actor must submit an update request with the same value. Only if this requirement is satisfied, is the binding transportation status updated.

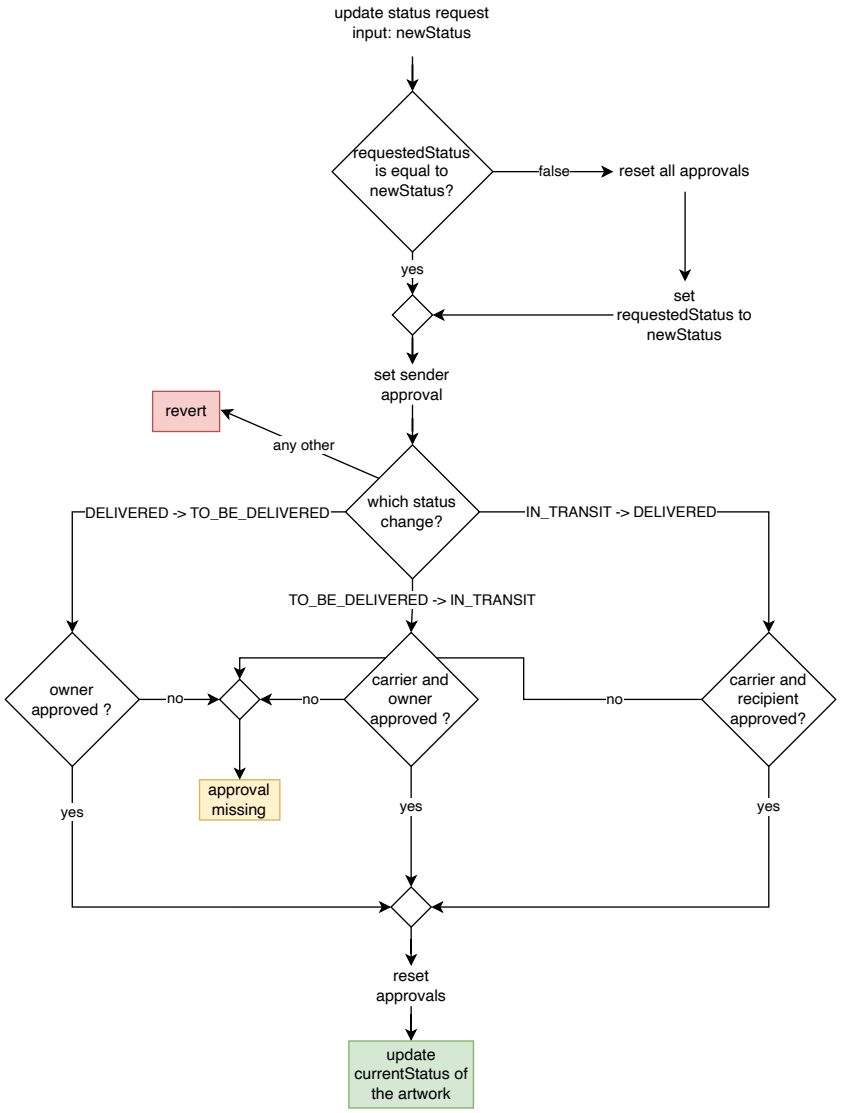


Figure 5.1: Multi-approval transportation status change

To differentiate between the status that actors want to change to and the status which is currently in place we introduced two variables: `requestedStatus` and `currentStatus`. Users can request a status change by submitting a new `requestedStatus`. If this value does not match the previous value, all approvals are reset, and the `requestedStatus` field is set to the new value. The next step is to set the approval for the actor who sent the initial update request. Because different actors must approve different status changes, we must check for the three distinct cases.

If the status is requested to change from TO_BE_DELIVERED to IN_TRANSIT, the carrier, and the owner must approve that change. This check is performed analogously for the change from IN_TRANSIT to DELIVERED. Lastly, the owner decides to change from DELIVERED to TO_BE_DELIVERED. Any other combination of status changes is not allowed, and the transaction will be reverted.

If the right approvals have already been made, the function will automatically update the `currentStatus` and terminate. If any approval is missing, it will terminate without updating the `currentStatus`. In that case, the actor in question must submit their request for a status update.

getArtworkData

The function to retrieve data about an artwork is the simplest one. It merely returns the `ArtworkData` object mapped to the requested artwork ID. To easily convert the returned tuple to an object using the contract Application Binary Interface (ABI), each property is mapped to the tuple entry specified in the function signature, visible in Listing 5.9.

```
function getArtworkData(uint256 tokenId, address sender)
  public
  view
  onlyAdmin
  exists(tokenId)
  read(sender, tokenId)
  returns (
    uint256 id,
    string memory objectId,
    address owner,
    address carrier,
    address logger,
    address recipient,
    string memory currentStatus,
    string memory requestedStatus,
    bool carrierApproval,
    bool ownerApproval,
    bool recipientApproval,
    uint256 violationTimestamp
  ) { ... }
```

Listing 5.9: SC getArtworkData function signature

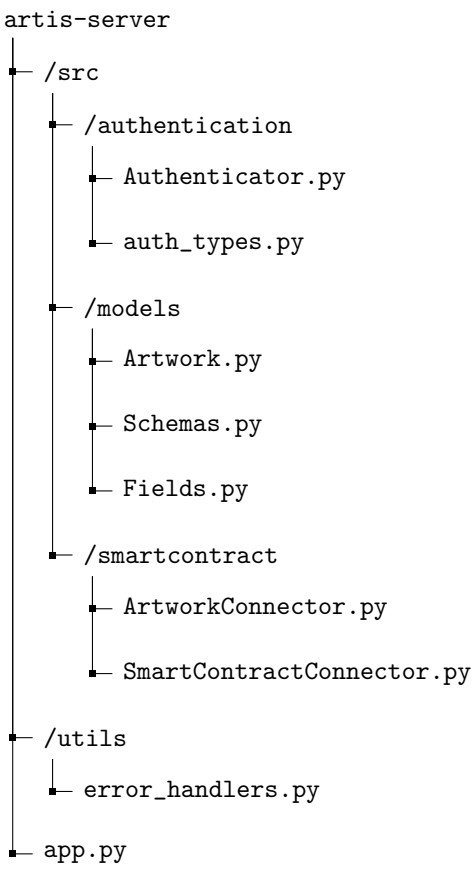


Figure 5.2: Server directory structure

5.2 artis-server

The server is the central part of the system; it connects the end users to the SC. The responsibilities of the server can be separated into three layers. The files of the server's directory structure depicted in Figure 5.2 and the corresponding code will be explained in the next sections. The server code is implemented in Python using a variety of libraries that will be addressed later on.

5.2.1 Authentication Layer

To implement the authentication flow described in section 4.2.1, we defined an `Authenticator` class, which handles the server-side logic. This class implements functions provided by the third web library [47] but has been adapted to fit our specific use case. The functionality described in this section is exposed by the API via the endpoints listed in Table 5.1b

The method described in Listing 5.10 constructs a dictionary with the information used to build the challenge message. The method is exposed by the API layer via the `/auth/payload` endpoint as listed in Table 5.1b. This dictionary is then returned to the client who constructs a message from it and signs it.

```
def generate_client_auth_payload(self, address: str, chain_id: str):
  return {
    "payload": {
      "version": "1",
      "type": "evm",
      "domain": "artis-project",
      "address": address,
      "chain_id": chain_id,
      "nonce": str(uuid4()),
      "issued_at": datetime.now(self.timezone).strftime(self.timeformat)
    ,
      "expiration_time": (
          datetime.now(self.timezone) + timedelta(hours=1)
      ).strftime(self.timeformat),
    }
}
```

Listing 5.10: Generating the challenge payload

```
def verify(
  self,
  domain: str,
  payload: LoginPayload,
  options: VerifyOptions = VerifyOptions(),
) -> str:

  ... # perform checks (e.g. expiration date)

  message = self._generate_message(payload.payload)
  user_address = self._recover_address(message, payload.signature)
  if user_address.lower() != payload.payload.address.lower():
      raise Unauthorized(
          f"The intended payload address '{payload.payload.address.lower
    ()}' is not the payload signer"
      )
  return user_address
```

Listing 5.11: Verifying the signature

The signed message is then sent back and verified by the `verify` method listed in Listing 5.11. After reconstructing the challenge message, the method recovers the account address that was used for the signature. If it matches the address specified in Line 7 of Listing 5.10 the authentication request is valid. This proves that the user who sent the request has the specified account. To recover the address, the python web3 library was used [53].

If the authentication request succeeds the client will issue a JSON Web Token (JWT) signed by the server wallet. This token is then used to authenticate further requests until it expires. The JWT's signature and expiration date are verified upon each request by the `Authenticator` class. The `Authenticator` exposes this functionality with the `/auth/login` endpoint.

Two additional authentication endpoints are exposed by the API. `/auth/logout` closes

the session and `/auth/user` returns information about the subject of a JWT. It can be queried to check if the session is still active and react accordingly on the client side.

5.2.2 API Layer

To expose the functionality defined in the use case and implemented in the SC, it is enough to implement four different API endpoints. The authentication mechanism described in Section 5.2.1 provides another set of four endpoints.

Endpoint	Method
`/artworks`	POST
`/artworks`	GET
`/artworks/<int:artwork_id>`	GET
`/artworks/<int:artwork_id>`	PATCH

(a) Resource endpoints

Endpoint	Method
`/auth/payload`	POST
`/auth/login`	POST
`/auth/logout`	POST
`/auth/user`	GET

(b) Authentication endpoints

Table 5.1: API endpoints

The JavaScript Object Notation (JSON) schema of the artwork resource is shown in Listing 5.12. Such a JSON object is returned by both of the `/artworks/<int:artwork_id>` endpoints. The properties marked with an arrow (→) can be included in the body of a PATCH request to `/artworks/<int:artwork_id>`. The POST request to `/artworks` expects the same body except for the `requestedStatus` property which cannot be set and the `owner` property which can be set to mint an artwork NFT for someone else. The request body is validated using the models defined in the `src/models/` folder. This is done by leveraging the functionality to define custom schemas, and fields of the Marshmallow [29] package.

```
{
  "id": integer,
  "objectId": string, ←
  "owner": string,
  "recipient": string | null, ←
  "carrier": string | null, ←
  "violationTimestamp": integer, ←
  "status": {
→ "requestedStatus": IN_TRANSIT | DELIVERED | TO_BE_DELIVERED | NONE,
    "currentStatus": IN_TRANSIT | DELIVERED | TO_BE_DELIVERED,
    "approvals" {
      "owner": boolean,
      "recipient": boolean,
      "carrier": boolean
    }
  }
}
```

Listing 5.12: Artwork JSON schema

```
{
  "owner": [integer],
  "recipient": [integer],
  "carrier": [integer],
  "logger": [integer],
}
```

Listing 5.13: GET `/artworks` response

A GET request to `/artworks` will return the artwork IDs where the requester is registered as an actor. This response is shown in Listing 5.13. A POST request to the same endpoint will return `{ "tokenId": integer }`.

To build the API application we used Flask [18]. The entry point of the Flask application is the `app.py` file in the base folder. The endpoint definitions of the `/artworks` route are shown in Listing 5.14. The logic in the entry methods is reduced to a minimum. The body is validated and loaded to the internal `Artwork` class. Then the request is processed by the application layer and returned after being converted back to a JSON object.

```
@app.get("/artworks")
@auth_required(authenticator)
def get_all() -> dict:
  return {"artworks": sc.getArtworkIdsByAddress(g.sender)}

@app.post("/artworks")
@auth_required(authenticator)
def mint() -> dict:
  artworkData = Artwork.load_from_mint(request.get_json())
  return {"tokenId": sc.safeMint(to=g.sender, data=artworkData)}

@app.get("/artworks/<int:artwork_id>")
@auth_required(authenticator)
def get(artwork_id: int) -> dict:
  return sc.getArtworkData(artwork_id, g.sender).dump()

@app.patch("/artworks/<int:artwork_id>")
@auth_required(authenticator)
def update(artwork_id: int) -> dict:
  newArtworkData = Artwork.load(request.get_json() | {"id": artwork_id})
  return sc.updateArtworkData(newArtworkData, g.sender).dump()
```

Listing 5.14: Definition of Endpoints

5.2.3 Application Layer

The application layer connects to the SC and submits the transactions needed to fulfill the API request. To do this we are using the python web3 library [53]. This functionality is defined in the `/src/smartcontract/ArtworkConnector.py` file. It defines a class that extends the `SmartContractConnector` abstract class. The `SmartContractConnector` class contains code that is generally necessary to connect to a SC. This includes setting

up the Web3 provider, as well as the account to submit the transactions, and adding some middleware. Additionally, we decided to define two *abstractmethods* that fetch the contract address and contract ABI dynamically. This ensures the API is always connected to the newest contract. In our case, we used GitHub variables to store the contract address and the Etherscan [16] API to query the ABI. With this setup, we built an extensible platform that can add endpoints to connect to additional SCs more easily.

The `ArtworkConnector` class has the same interface as the solidity contract. As shown in Listing 5.15, each public method directly connects to a SC method. Web3.py provides a simple-to-use smart contract proxy object which we stored in the class variable `self._contract`. This object makes all the contract functions available with the `functions` property. To call a contract function it suffices to execute `self._contract.functions.<function_name>(<input>).call()`. This will return the value from the SC and can more or less be directly returned to the user. If the function mutates the state of the contract, we need to use `.transact()` and a transaction hash is returned instead. We use this transaction hash to wait for the transaction receipt and extract the data from the emitted events. This is done by the helper function `self._handleEvent`.

```
def safeMint(self, to: bytes, data: Artwork):
  owner, mint_data = data.to_sc_mint()
  tx_hash = self._contract.functions.safeMint(
    to if not owner else owner, mint_data
  ).transact()
  event_args = self._handleEvent(tx_hash, "Transfer")
  return event_args.get("tokenId")

def updateArtworkData(self, newArtworkData: Artwork, sender: bytes):
  tx_hash = self._contract.functions.updateArtworkData(
    newArtworkData.to_sc_update(), sender
  ).transact()
  event_args = self._handleEvent(tx_hash, "Updated")
  new_data = event_args.get("newData")
  new_data = dict(new_data, **{"owner": event_args.get("owner")})
  new_data["status"] = dict(
    new_data["status"], **{"approvals": event_args.get("approvals")}
  )
  return Artwork.load(data=new_data)

def getArtworkIdsByAddress(self, address: str):
  artwork_ids = (
    self._contract.functions.getArtworkIdsByAddress(address).call().
    _asdict()
  )
  # incoming lists are zero padded to the total supply of tokens
  remove_zeros = lambda d: {
    k: list(filter(lambda x: x != 0, v)) for k, v in d.items()
  }
  return remove_zeros(artwork_ids)

def getArtworkData(self, artworkId: int, sender: str):
  data=self._contract.functions.getArtworkData(artworkId, sender).call()
  return Artwork.load(data=dict(data._asdict()))
```

Listing 5.15: ArtworkConnector class methods

5.3 artis-rockpi-logger

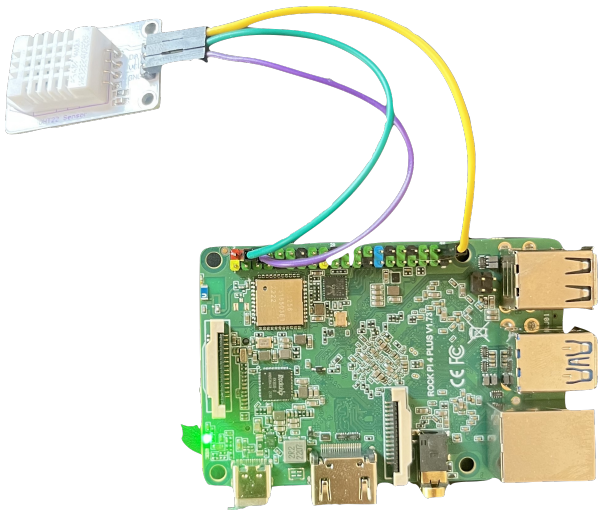


Figure 5.3: Rock Pi with a DHT-22 sensor

Another major component of the system is the logging device used to monitor the environmental data during the transportation of the artwork. For this project, we used a Rock Pi [37] with a DHT-22 [2] sensor attached. (Figure 5.3)

The decision for this device and sensor is based on simplicity, flexibility, and low acquisition costs. The Rock Pi is compatible with the Linux operating system, making writing scripts using high-level programming languages like Python very simple. It also has General Purpose Input/Output (GPIO) support to connect the DHT-22 sensor easily. The sensor is capable of recording temperature and humidity data from the environment.

To read data from the DHT-22 sensor, we used an existing library called "Rockfruit Python DHT" [48] that we adapted slightly to work with our board. The library provides a function with the signature `read_retry(sensor, pin, retries=15, delay_seconds=2)`. This method reads data from the DHT sensor of the specified type (in our case 22) on the specified GPIO pin. It returns a tuple of humidity (as a floating point value in percent) and temperature (as a floating point value in Celsius).

The function will attempt to read multiple times (up to the specified retries) until a reading can be registered. If no reading can be registered after the number of retries, the function will return `(None, None)`. The default delay between retries is two seconds but can be overridden.

5.3.1 Code

In Listing 5.16, you can see that the script is executing the `read_retry` function in an endless loop, and whenever the reading has been successful, it stores the data along with a timestamp of the reading in seconds and a readable format in a local database.

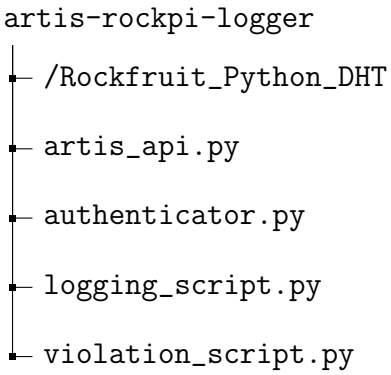


Figure 5.4: Logger directory structure

```
while True:
  humidity,temperature = Rockfruit_DHT.read_retry(sensor,pin,retries=15)
  timestamp = datetime.datetime.now()
  if humidity is not None and temperature is not None:
    c.execute(
      "INSERT INTO readings VALUES (?, ?, ?, ?)",
      (
        timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        int(timestamp.timestamp()),
        temperature,
        humidity,
      ),
    )
    conn.commit()
```

Listing 5.16: Temperature and humidity logging loop in `logging_script.py`

The logger contains a second important script which is run simultaneously. The `violation_script.py` is executed with the predefined temperature and humidity thresholds and the corresponding artwork ID.

```
python3 violation_script.py \
  --artwork-id 1 \
  --temperature-threshold 30 \
  --humidity-threshold 80
```

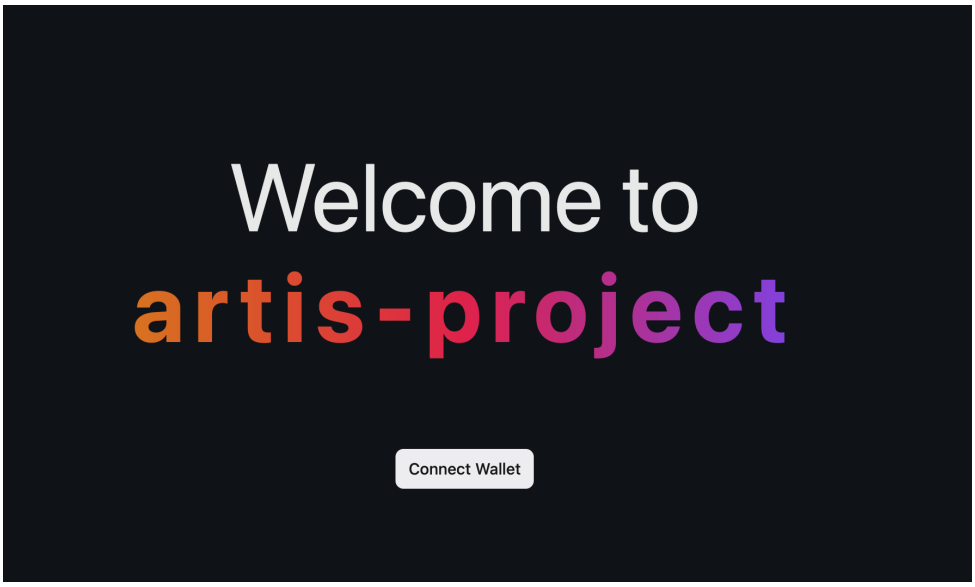
Listing 5.17: Executing the `violation_script.py`

This script contains another while loop that continuously checks for new entries in the database and evaluates whether the threshold has been exceeded. If there has been either a temperature or humidity violation, the script calls the corresponding method to call the `ARTIS` server API. These methods are defined in the `artis_api.py` file.

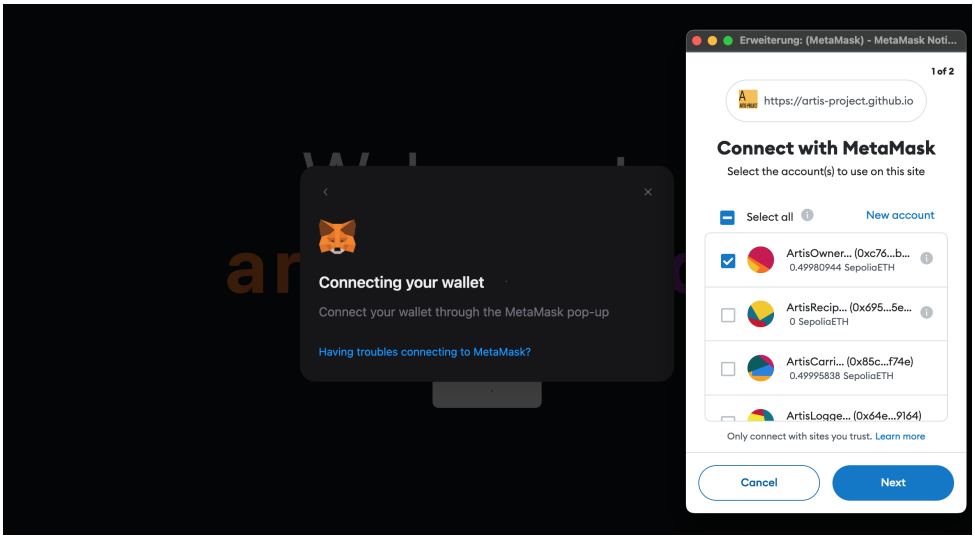
To authenticate to the API, the logger follows the same authentication flow illustrated in Figure 4.3. The methods used to send the authentication requests and sign the challenge message are defined in the `authenticator.py` file.

5.4 artis-frontend

The last component of the system is the user interface. It is a single-page application built with React [36] and TypeScript. The styling is done with Tailwind CSS [46]. These technologies were selected due to the author's prior experience. Theoretically, the user interface is not required to interact with the system. However, simple authentication is a large benefit of the user interface. Its integration with Metamask [31] enables a seamless login and authentication flow. From a code perspective, thirdweb [47] is used to access the API of Metamask. A short demo of the user interface can be viewed here GitHub[2]



(a) Welcome Screen



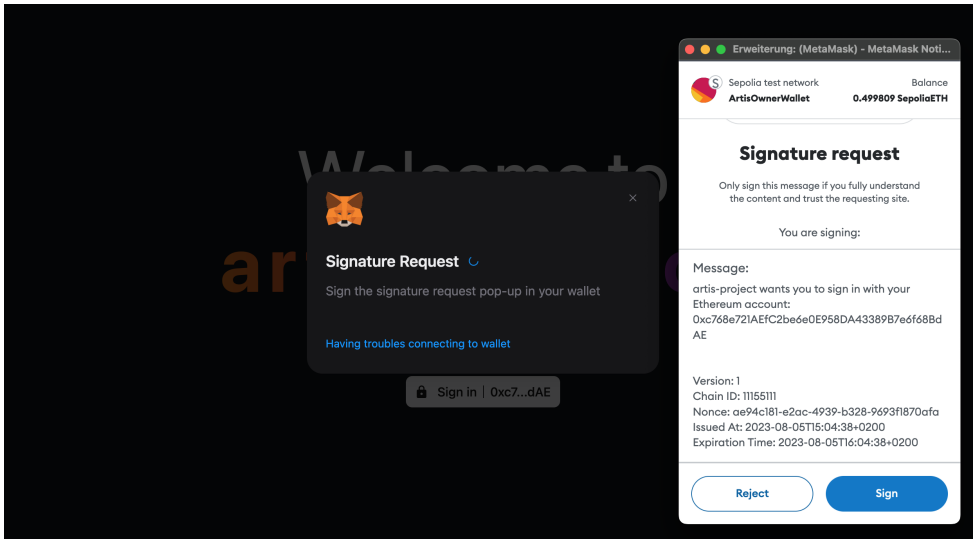
(b) Connecting to Metamask

An unauthenticated user first lands on this welcome page with a button to connect the website to Metamask as shown in Figure 5.5a. After pressing the button, Metamask opens

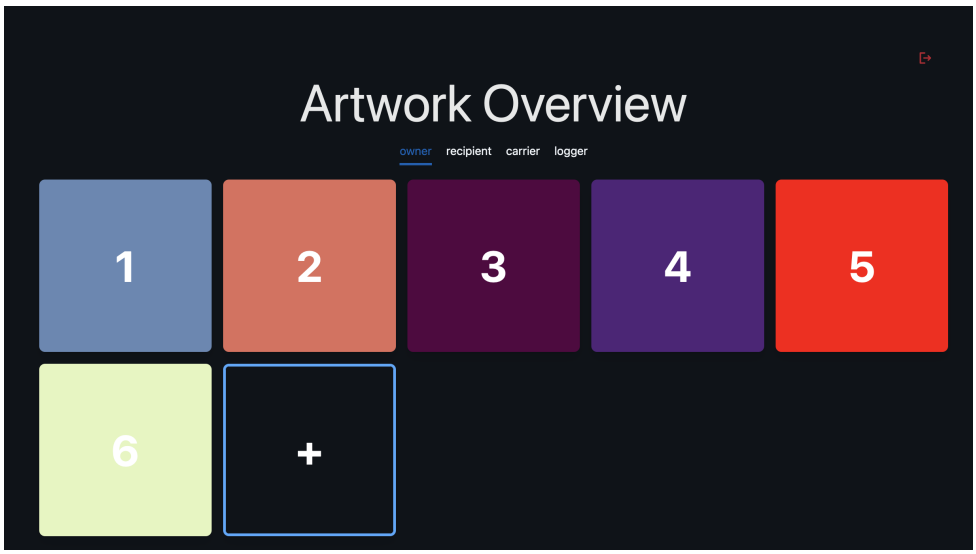
[2]https://artis-project.github.io/artis-thesis/frontend-demo.mov

a pop-up to select an account to connect to the website (Figure 5.5b). Finally, the button label changes to "Sign in" and as soon as the user presses the button again, a signature request is sent to the server. The message visible on the pop-up in Figure 5.5c is a message constructed from the response of the POST `/auth/payload` request.

Once the user signs this message, the signature is sent back to the server, validated, and an authentication token is issued. The user is now logged in and can see all the artworks by their IDs that are associated with the logged-in account (Figure 5.5d).



(c) Sign in request



(d) Artworks page

Figure 5.5: Sign in process

The artwork page shows a colorful tile representing an artwork. Each tile is labeled with the corresponding artwork ID. The tabs on the top can be selected to view all the artworks where the account is registered *e.g.* the user might be the artwork owner 1 but the recipient 2.

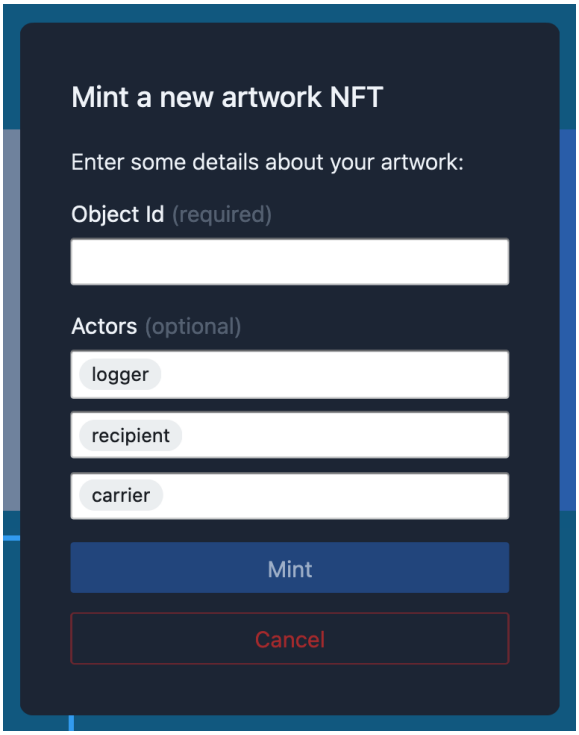


Figure 5.6: Artwork mint modal

Figure 5.7: Artwork detail view

To create a new artwork NFT, the user can click on the transparent tile at the bottom. This will open up a modal where the user can enter details about the artwork (Figure 5.6). To view more details about an existing artwork the user can click on a tile to open up the detail view shown in Figure 5.7. This view displays important information like the status of the artwork, the role the user is registered as, and the timestamp of the most recent violation if any occurred. Further, the user can edit values by switching on the edit mode with the switch on the top right.

Chapter 6

Evaluation

In this chapter, we evaluate different aspects of the developed system. Section 6.1 shows a cost and a performance analysis of the developed SC. Section 6.3 concludes the analyses with a field test of a simulated artwork transportation scenario. Finally, this chapter is concluded with a discussion in Section 6.4 To evaluate the performance of the system we also recorded the response time of each request. This can give an indicator

6.1 Cost and Performance Analysis

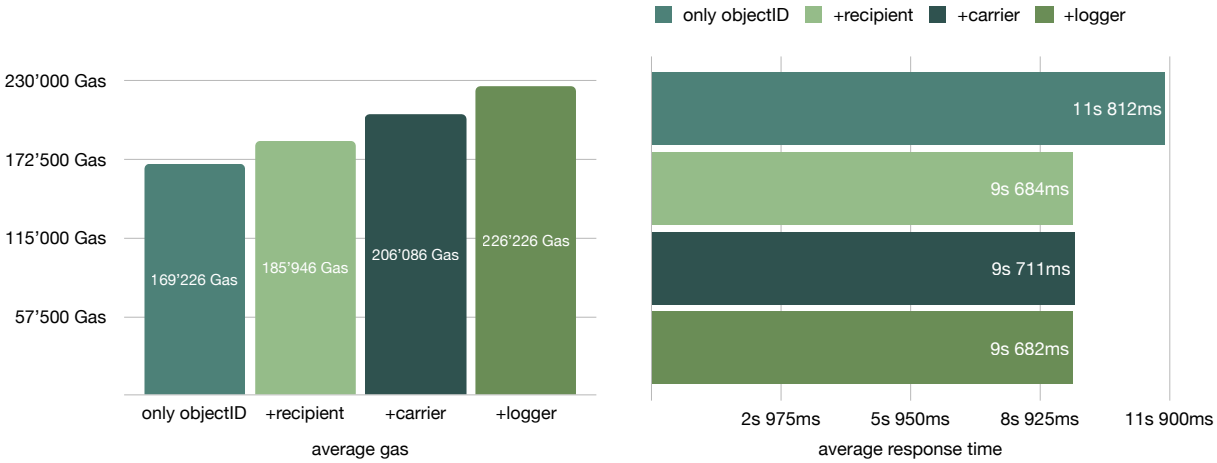
The primary cost factor of the artis-system is the SC. Each execution of a function that mutates the state of the SC costs a certain amount of gas which in turn has a price in Ether. The cost for a transaction thus is calculated as follows:

$$transaction\ fee = gas \times gas\ price$$

To analyze the execution costs of the *safeMint* and *updateArtworkData* contract functions, we executed the corresponding requests of the developed API multiple times ($n = 10$) for each input variety. This analysis was conducted on the sepolia testnet and the resulting transactions were inspected on the Etherscan block explorer. The transactions were submitted with a relatively high gas price (30 Gwei) to make the transactions attractive for validators [13]. During this analysis, we observed that the gas used for a specific function depends on the input parameters but does not vary if the input parameters are the same. To analyze the performance, we also recorded the processing time of each request. This time can be an indicator of the performance but depends on the state of the blockchain and likely differs on the mainnet.

To get a sense of how gas translates into fees, we used the average gas price of the past month for the Ethereum mainnet (28 Gwei, 07.08.23 - 08.08.23) [15] and the current value of Ether in Swiss Francs (CHF) (1625.20 CHF = 1 ETH, 09.08.23) [11]. Because the contract can also be deployed on other blockchains that use the Ethereum Virtual Machine (EVM), we decided to include the cost on the polygon network. Except the contract has not been deployed to the Polygon network, and the same amount of gas

was used for these calculations. The average gas price of the past month (173 Gwei [35], 07.08.23 - 08.08.23) was multiplied by the gas and converted into CHF by using the current price for MATIC (0.60 CHF = 1 MATIC, 09.08.23) [11]. The results were rounded to four decimal points for the tokens and two decimal points for CHF.



(a) Transaction gas usage

(b) Request speed in seconds

Figure 6.1: safeMint analysis

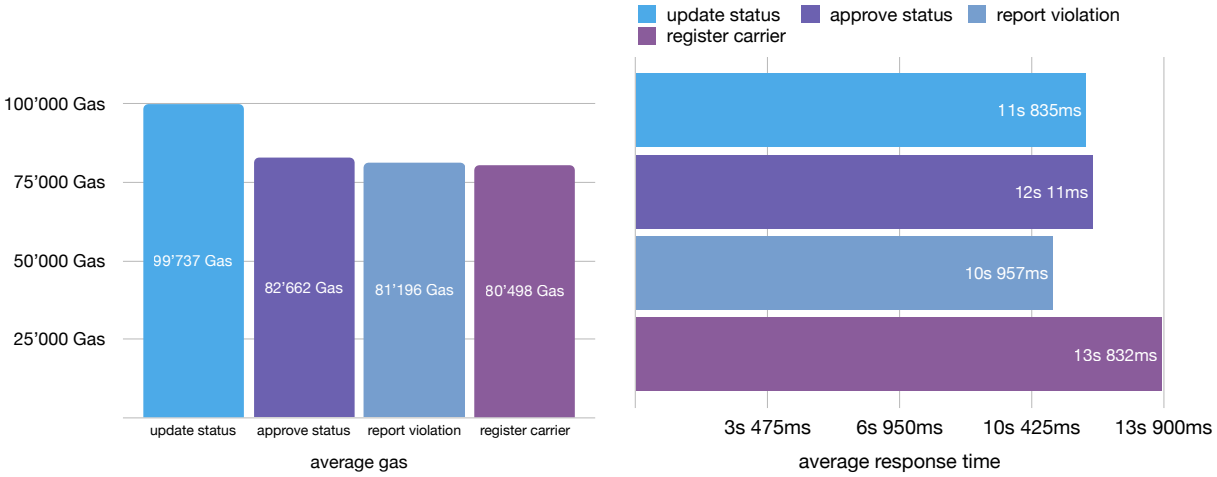
safeMint

Figure 6.1a shows the average gas the safeMint function consumes for various input parameters. Each bar shows the average of 10 function executions. The very left bar shows the average gas used if only the objectID of the artwork is added during minting. The second bar shows the gas consumed if the objectID and the recipient address are provided during minting. This pattern remains the same for the last two bars. The analysis shows that the more parameters are provided, the more gas it consumes. Table 6.1 shows the transaction fees in the corresponding native token and converted into CHF as outlined above.

	only objectID	**+recipient**	**+carrier**	**+logger**
transaction fee ETH (MATIC)	0.0048 (0.0293)	0.0053 (0.0322)	0.0059 (0.0357)	0.0064 (0.0392)
in CHF Ethereum (Polygon)	7.83 (0.02)	8.61 (0.02)	9.54 (0.02)	10.47 (0.02)

Table 6.1: Estimated transaction fees safeMint

The response time of the API calls are mostly below 10s. We could not observe a large difference in the different input parameters. The chart in Figure 6.1b shows the average response time in seconds. The maximum response time recorded was 33s on the fifth call with only the objectID submitted, and the minimum was 3s with the objectID and recipient address submitted. This shows that this metric can fluctuate heavily depending on the state of the network.



(a) Transaction gas usage

(b) Request speed in seconds

Figure 6.2: updateArtworkData analysis

updateArtworkData

Similarly, we also analyzed the updateArtworkData function. The bar on the very left of Figure 6.2a shows the average gas consumed by the function when updating the requested status. Approving the status as a different actor shows an average gas consumption of 82'662. The last two bars show the gas amount for reporting a violation from the logger and registering a carrier.

The cost of these transactions was estimated similarly to with the safeMint function (Table 6.2). The results show that updating an artwork NFT is much less costly than minting it.

	update status	**approve status**	**report violation**	**register carrier**
transaction fee ETH (MATIC)	0.0028 (0.0173)	0.0024 (0.0143)	0.0023 (0.0141)	0.0023 (0.014)
in CHF Ethereum (Polygon)	4.62 (0.01)	3.83 (0.01)	3.76 (0.01)	3.73 (0.01)

Table 6.2: Estimated transaction fees updateArtworkData

The average response time of updating an NFT varies depending on the input parameters. The request that was fulfilled the fasted approved a status change in around three Seconds. Interestingly, the longest response time of over 35 seconds was recorded on the same type of request.

Non-Mutating Functions

The GET endpoints exposed by the API target several functions that do not mutate the state of the SC. These functions do not cost gas, and their execution time is much less (usually $<$ 1 Second). Additionally, the artis-server adds a caching layer to these function calls, which generally reduces the response time of repeated calls to a few milliseconds.

6.2 Security Analysis

Since the system is intended to be used by different actors in a trust-less environment a security analysis is vital for evaluating the prototype. The following Section discusses the different threat scenarios and evaluates the security risk.

6.2.1 Compromised Logger

The logging device is important to the system, and a compromised device would lead to a compromised system altogether. We consider three different attack scenarios when it comes to the logger.

Software Exploitation

Currently, the device itself is not secured in any way. Any malicious actor could connect to the logger device and alter its software. This could include changing the violation thresholds, stopping the logging script, manipulating the local database of the readings, and more. Such a threat scenario compromises the integrity, confidentiality, and availability of the system. Because the current system is not protected against this threat, the security risk is high.

Hardware Exploitation

The recordings can also be influenced externally by exploiting the hardware components. By gaining access to the logger device, a malicious actor could influence the environmental parameters in a minimal perimeter around the sensor and falsify the readings. This could result in violations even if the artwork itself was not affected or could prevent violations even if the artwork was affected. The integrity and availability of the system can be compromised, which poses a high-security risk.

Compromised Credentials

The system only allows registered actors to change the NFT. This is also true for the logger device, which authenticates the system by signing a challenge message with a private key. If the private key is leaked, a malicious actor could report violations even if none occurred. Currently, the private key is stored in an environment variable on the software. An attacker could easily retrieve the value of the private key by connecting to the logger. This indicates a high-security risk for this threat by compromising the confidentiality and integrity of the system.

6.2.2 Compromised Smart Contract

A malicious actor could influence the state of the SC by accessing it directly. The attacker would have to identify flaws in the contract code and find an exploit to manipulate artwork data. We estimate the security risk of finding a flaw in the code to be medium, as the contract was tested during development, and basic code-checking tools could not identify a vulnerability. The SC only accepts transactions issued by the system's admin. If the admin credentials are leaked, the attacker would have unlimited access to the SC. In the current system, the admin's private key is only stored on the artis-server. The server is deployed as a Google Cloud-run service, and the Google secret manager supplies the private key. The security risk for this threat scenario is estimated to be low because the secret manager is an industry-standard solution for managing sensitive data.

6.2.3 Disclosure of Data

The data stored on the SC could be sensitive depending on the use case. The SC prevents unauthorized users from reading data. However, the transactions submitted to update data can contain sensitive information. This information can be read by anyone who has access to the SC address, admin address, or any of the actors' addresses. Because these addresses are not considered secrets, the system must disclose all data to the public. This could introduce a high-security risk for certain use cases as confidentiality would be compromised.

6.3 Field Test

Because the isolated analyses performed above do not indicate how the system would perform during an artwork transportation scenario, we used the system on a simulated transportation scenario.

The steps of the simulated scenario are visualized in Figure 6.3 and described in more detail below.

1. Creates a new artwork NFT and registers the corresponding roles using the artis-frontend.
2. Request the status of the artwork to be changed to *IN_TRANSIT* from the sender account
3. Approve this request from the carrier account
4. Enable the logger by starting the *logging_script* and the *violation_script* with the thresholds of 25 degrees Celcius and 70% humidity.
5. Take the logging device and transport it from the point of departure to the destination

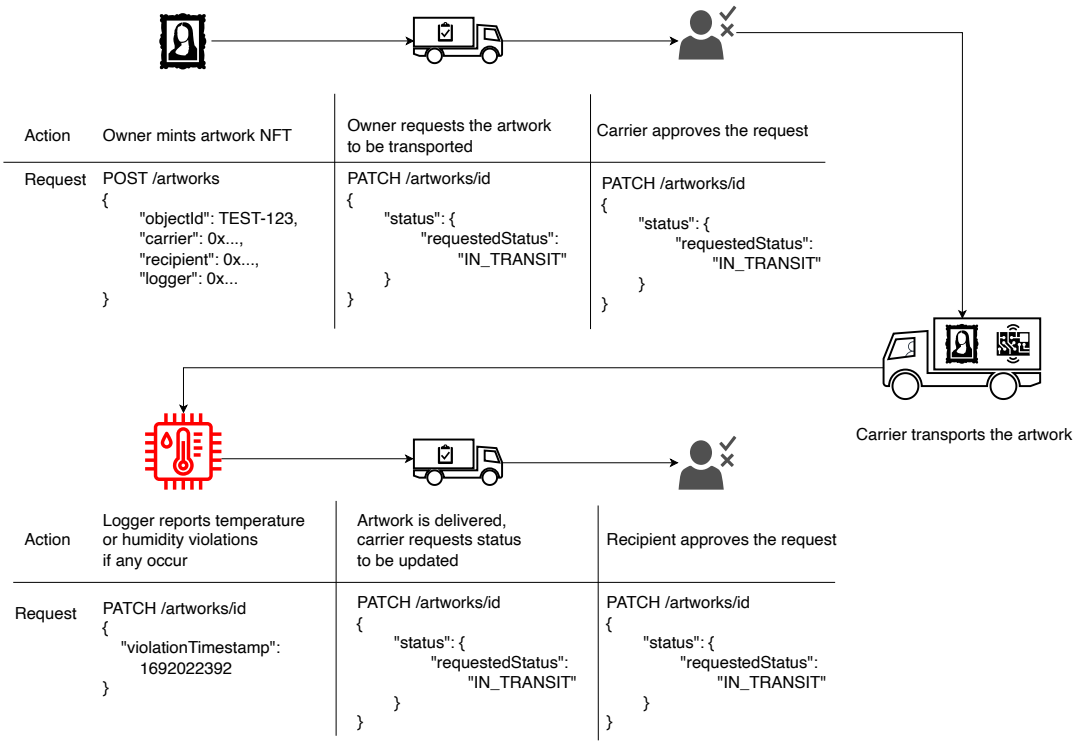


Figure 6.3: Field test scenario

6. During the transportation, simulate a violation by wrapping a hand around the sensor to increase the temperature.

7. Upon arrival, request the status of the artwork to be changed to *DELIVERED* from the carrier account

8. Approve this request from the recipient account

The field test involves a minimum of five requests to the API. If any violations occur, this number is increased accordingly. We also simulated a temperature violation by covering the sensor with our hand (Figure 6.4b) to include a violation in the test. The test was performed on the go, traveling by train and using a power bank to power the device and a hotspot from a cellular phone to connect to the internet. During the test, we recorded different metrics now used for evaluation. The most important metrics include a cost analysis of the field test and an evaluation of the performance of the logger.

6.3.1 Cost

During our field test, nine transactions were issued by the API. The cost analysis also includes deploying the SC and minting the NFT. However, this would be a one-time cost, and its significance will shrink with the number of transportation. Table 6.3 gives an overview of the transactions and estimated costs. The costs are estimated in the same manner as in Section 6.1. However, the transactions were not issued with a fixed gas price, but each transaction was estimated based on the network at the time.

(a) Simulating artwork transportation scenario (b) Simulating temperature violation

Figure 6.4: System Field Test

The total cost of the field test mainly consists of the contract deployment. This transaction accounts for over 78% of the total gas consumption and thus also accounts for the greatest cost factor. The variable part included in another similar transportation scenario of the same artwork amounts to around 25.85 CHF on the Ethereum network or around 0.05 CHF on the Polygon network.

Transaction	Count	Total Cost on Ethereum (Polygon)
Contract deployment	one time	143.79 (0.33)
mint NFT	one time	12.05 (0.03)
update status	2	9.26 (0.02)
approve status	2	7.68 (0.02)
report violation	3	8.91 (0.03)
Total	**9**	**181.69 (0.43)**

Table 6.3: Transaction fees field test

6.3.2 Logger Reliability

The field test lasted around 52 minutes. During the transportation, the logger stored every sensor reading in a local database. Figure 6.5 illustrates the intervals of the sensor readings. Each line indicates a successful reading by the sensor; the red lines represent a violation. The maximum time between two readings was 9 minutes and 36 seconds. The shortest interval was less than a second. We did not test other metrics such as accuracy or range for this prototype.

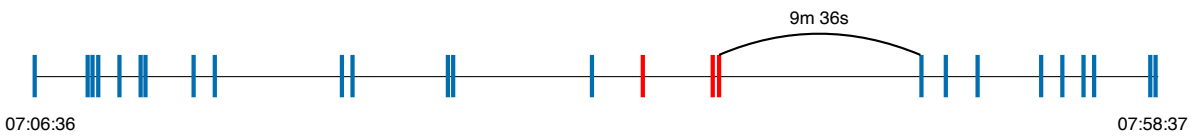


Figure 6.5: Timeline with sensor readings

6.4 Discussion

The performance and cost of the developed system depend mainly on the blockchain used as infrastructure. Such a blockchain is essential of significant importance and must be chosen with great diligence. The prototype deployed on the sepolia testnet demonstrates the influence of different factors, such as network congestion and priority fees on the time needed for a transaction to be completed. On the testnet, the average response time with a fixed gas price of 30 Gwei is reasonable and could support a real-life scenario. However, this is not representative of the mainnet.

However, the gas consumption of the SC functions can be directly adapted to any blockchain running the same EVM version. The analysis shows that executing a simple scenario on the Ethereum network can be very costly. Polygon solves this issue as our cost estimation is reduced by a factor of over 422. On Ethereum, the total cost of a scenario is quite costly, but considering the artwork's potential value, we think it could be acceptable for a real-life scenario. With Polygon, the estimated total cost is minimal and acceptable.

The security risks illustrated in Section 6.2 prevent the system from being used in a real-life artwork transportation scenario. The developed system serves as a proof of concept, and the security vulnerabilities must be mitigated for a production version.

The intervals of successful readings by the sensor have shown to be inconsistent and leave large gaps between readings. Unfortunately, the quality of the sensor would not be enough to support a real-life use case. A suitable sensor should be able to consistently record at shorter intervals, preferably less than a minute long. The reason for the sensor's poor performance is unclear. Possible explanations include a timing issue in the communication between the board and the sensor, voltage fluctuations from the portable power bank, or other environmental factors. We believe this issue could easily be solved using a higher quality board with integrated sensors, such as the B-L462E-CELL1 Discovery kit by ST [44].

Chapter 7

Summary and Conclusions

This chapter aims to conclude the thesis with a summary, conclusions, and potential future work based on the insights gained during development and the limitations of the system.

7.1 Summary

This thesis proposes a novel system to monitor and track artwork during transportation. The system is designed to include both blockchain and IoT to improve transparency and traceability in the art world. The idea is to attach an IoT device to the physical artwork that records and stores environmental data critical to the integrity of the artwork. The logging device evaluates the recorded data against a predefined threshold and alerts the stakeholders if any deviations occur. Any data associated with the artwork is stored on the blockchain to serve as an indisputable record of events. To create a proof of concept for this proposed system, a prototype was realized within the scope of this thesis.

The first step was to establish the theoretical background for this project through a literature review. The thesis introduced the concepts of blockchain, IoT, SCs, and continued with a review of related work. The focus was placed on gathering existing solutions or work that use either blockchain, IoT, or both in tracking and monitoring systems. The results of the literature review were aggregated into a summary and used as inspiration for the design of the proposed system.

The prototype includes a user-friendly web application that serves as an interface to the developed API. This REST API offers an abstracted interaction method with the SC deployed on the blockchain. The system also features a monitoring device that is capable of logging temperature and humidity data and reporting any deviations from predefined thresholds to the server. The developed system presents a novel approach to artwork tracking and monitoring by integrating blockchain and IoT.

The design of the system includes a description of a simplified artwork tracking scenario that should be supported by the final prototype. The architecture of the system was then

derived from this simplified scenario. The final system comprises a frontend user interface, a backend server, a SC, and the IoT device. The frontend is designed to provide a user-friendly interface to the backend server. The backend server exposes a REST API to interact with the SC. The developed SC defines the unique digital counterpart of physical artwork as a NFT. By storing additional details on the SC, it becomes feasible to register stakeholders as key actors. Their designated blockchain identities can then be used to access information about the artwork status or request modifications. Authorization is handled by the SC itself through a role-based policy while authentication is handled on the server by challenging users to provide proof of ownership. The IoT device is designed to record temperature and humidity data and report violations of a predefined threshold via the API to the SC.

The established design and architecture were then implemented in the form of a prototype. The SC was implemented in Solidity and deployed to the sepolia testnet. The backend server is written in Python and uses the web3py library to execute contract functions by initiating transactions. Those functions are then exposed as API endpoints using the Flask framework. The backend server is deployed to the web as a Google Cloud-run service. The frontend is built using react and provides a Graphical User Interface (GUI) for the API endpoints. Using Metamask the user is able to easily authenticate to the backend server by signing a challenge message and providing proof of ownership over the Ethereum account used. Lastly, to monitor humidity and temperature a rockPi was used with a DHT-22 sensor attached. The software for the logger is written in Python.

The system was evaluated in terms of cost, performance, and security as well as a simulated artwork tracking scenario. The cost analysis was performed on the sepolia testnet and used the recorded gas consumption of function executions to estimate the transaction fees on the mainnet. The analysis also includes an estimation of the fees if the system was deployed on the Polygon network. To evaluate the performance of the system, the requests to the API were timed multiple times and the resulting average was discussed. To evaluate the security of the system, a number of potential threat scenarios were analyzed and rated in terms of security risk. The field test involved deploying a SC instance, minting an NFT, registering the actors, requesting the artwork to be delivered, approving a status change, reporting a simulated violation, and finally delivering the artwork and changing the status to delivered. The transportation scenario included the logging device being transported outside for roughly an hour.

7.2 Conclusions

A prototype system to monitor and track artwork using IoT and blockchain technology was successfully designed, implemented, and evaluated. The general objective has been clearly achieved by this prototype. The system is capable of creating an NFT as a digital counterpart of artwork. The owner of the artwork is then able to register other stakeholders and effectively give them read/write permissions to information about the artwork and its current status. By attaching a logging device to the artwork, the system is able to monitor environmental conditions and report any deviations to the system. Those violations are then irrefutably stored on the blockchain. Furthermore, the transaction history

on the blockchain can be used to trace ownership and custody of the artwork. The system also provides a user-friendly frontend interface as well as an abstracted REST API to interact with the system and the underlying SC.

The development of the system presented major challenges. For instance, authenticating to the system involves signing a challenge message with a private key. This process can be cumbersome to achieve without proper wallet integration. To overcome this challenge we decided to integrate the popular wallet provider Metamask in our frontend. With this integration, the authentication process is as simple as clicking a button. Another challenge was finding a suitable IoT device. Initially, the idea was to use a more sophisticated device with integrated sensors. The development on such a board turned out to be more complex and we decided to switch to a simpler device that is able to run on a familiar operating system like Linux. This choice resulted in a decline in the sensor's quality. Initially, the scope of this thesis also included protecting sensitive data by designing a confidentiality scheme using Zero-Knowledge Proof (ZKP). We decided to leave this for future research as this was intended to be achieved by using existing libraries and frameworks that turned out to be harder to integrate than expected. Instead, we focused on a different set of features for the final prototype.

The developed prototype has been used in a simulated artwork transportation scenario and has been shown to work as intended. Nevertheless, the prototype has some limitations that have to be addressed in future work.

7.3 Future Work

This thesis has demonstrated a minimal prototype to demonstrate a novel approach to artwork tracking and monitoring in transportation scenarios. To arrive at a mature state of the system there is still great room for improvement and further development.

First, it would be essential to upgrade certain components of the system. This may include upgrading to a better sensor as well as adding more sensors to track location, vibration, and other environmental data. With an upgraded IoT device another field test performed at a larger scale will bring further insight into the feasibility of the system in a real-world scenario.

Secondly, future work should address the security risks outlined in Section 6.2. Primarily the protection of sensitive data should be investigated. This may involve exploring the potential of using ZKPs to protect data exposed in blockchain transactions [43].

Lastly, the current design includes a centralized server to provide an interface to the SC. This introduces a single point of failure to the system and it might be worth exploring a truly distributed approach.

Bibliography

[1] 4art-technologies.com. *4ARTechnologies*. URL: https://www.4art-technologies.com/de/.

[2] adafruit.com. *DHT22*. URL: https://www.adafruit.com/product/385. *Date accessed: 18/08/2023*.

[3] artory.com. *Artory*. URL: https://www.artory.com/. *Date accessed: 22/03/2023*.

[4] authena.io. *Authena*. URL: https://authena.io/. *Date accessed: 22/03/2023*.

[5] Mohamed Ben-Daya, Elkafi Hassini, and Zied Bahroun. "Internet of things and supply chain management: a literature review". *International Journal of Production Research* 57.15-16 (2017), pp. 4719–4742. ISSN: 1366588X. DOI: 10.1080/00207543.2017.1402140.

[6] T Bocek, B B Rodrigues, T Strasser, and B Stiller. "Blockchains everywhere - a use-case of blockchains in the pharma supply-chain". *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2017, pp. 772–777. DOI: 10.23919/INM.2017.7987376.

[7] Vitalik Buterin. "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform." *Ethereum Whitepaper* (2014).

[8] Mauro Cannistraro, Giuseppe Cannistraro, and Roberta Restivo. "Environmental monitoring of sacred artworks – A case study for the search for an index of correlation between particle concentration and mass of fine dust". *Thermal Science and Engineering Progress* 14 (Dec. 2019), p. 100405. ISSN: 2451-9049. DOI: 10.1016/J.TSEP.2019.100405.

[9] Vincenza Carchiolo, Mark Phillip Loria, Marco Toja, and Michele Malgeri. "Real Time Risk Monitoring in Fine-art with IoT Technology." *FedCSIS (Communication Papers)*. 2018, pp. 151–158.

[10] certify-project.eu. *CERTIFY*. URL: https://certify-project.eu/. *Date accessed: 10/04/2023*.

[11] coinmarketcap.com. *CoinMarketCap*. URL: https://coinmarketcap.com/currencies. *Date accessed: 09/08/2023*.

[12] ethereum.org. *ERC-721 Non-Fungible Token Standard*. URL: https://ethereum.org/en/developers/docs/standards/tokens/erc-721/. *Date accessed: 26/03/2023*.

[13] ethereum.org. *Gas and fees*. URL: https://ethereum.org/en/developers/docs/gas/. *Date accessed: 14/08/2023*.

[14] ethereum.org. *Sepolia Testnet*. URL: https://ethereum.org/en/developers/docs/networks/#sepolia. *Date accessed: 26/06/2023*.

[15] etherscan.io. *Ethereum Average Gas Price*. URL: https://etherscan.io/chart/gasprice. *Date accessed: 09/08/2023*.
[16] etherscan.io. *Etherscan*. URL: https://etherscan.io/. *Date accessed: 05/08/2023*.
[17] everledger.io. *Everledger*. URL: https://everledger.io. *Date accessed: 22/03/2023*.
[18] flask.palletsproject.com. *Flask*. URL: https://flask.palletsprojects.com/en/2.3.x/. *Date accessed: 02/08/2023*.
[19] handelszeitung.ch. *Authena: Garantie für fälschungssichere Produkte*. URL: https://www.handelszeitung.ch/podcasts/upbeat/authena-garantie-fur-falschungssichere-produkte-615882. *Date accessed: 23/08/2023*.
[20] hardhat.org. *Hardhat*. URL: https://hardhat.org/. *Date accessed: 26/06/2023*.
[21] hasenkamp.com. *hasenkamp Fine Art*. URL: https://hasenkamp.com/en/fineart/packaging. *Date accessed: 22/08/2023*.
[22] ICOM. *Object ID - International Council of Museums*. URL: https://icom.museum/en/resources/standards-guidelines/objectid/. *Date accessed: 11/03/2023*.
[23] kraft-els.ch. *Kraft E.L.S. AG - Exhibition Logistics Service*. URL: https://kraft-els.ch/en/packing/. *Date accessed: 22/08/2023*.
[24] Elia Landi, Lorenzo Parri, Riccardo Moretti, Ada Fort, Marco Mugnaini, and Valerio Vignoli. "An IoT sensor node for health monitoring of artwork and ancient wooden structures". *2022 IEEE International Workshop on Metrology for Living Environment, MetroLivEn 2022 - Proceedings* (2022), pp. 110–114. DOI: 10.1109/METROLIVENV54405.2022.9826938.
[25] Elisabetta Lazzaro. "Blockchain opportunities and challenges in the art market" (2020).
[26] Z. Li, Ray Y. Zhong, Z. G. Tian, Hong Ning Dai, Ali Vatankhah Barenji, and George Q. Huang. "Industrial Blockchain: A state-of-the-art Survey". *Robotics and Computer-Integrated Manufacturing* 70 (Aug. 2021), p. 102124. ISSN: 0736-5845. DOI: 10.1016/J.RCIM.2021.102124.
[27] Somayya Madakam, R. Ramaswamy, and Siddharthi Tripath. "Internet of Things (IoT): A Literature Review". *Journal of Computer and Communications* 03.05 (2015), pp. 164–173. ISSN: 2327-5219. DOI: 10.4236/JCC.2015.35021.
[28] Nikhil Malik, Yanhao Wei, Gil Appel, and Lan Luo. "Blockchain technology for creative industries: Current state and research opportunities". *International Journal of Research in Marketing* 40.1 (Mar. 2023), pp. 38–48. ISSN: 0167-8116. DOI: 10.1016/J.IJRESMAR.2022.07.004.
[29] marshmallow.readthedocs.io. *marshmallow*. URL: https://marshmallow.readthedocs.io/en/stable/. *Date accessed: 05/08/2023*.
[30] MF Mecklenburg. "Art in transit: studies in the transport of paintings" (1991).
[31] metamask.io. *Metamask*. URL: https://metamask.io/. *Date accessed: 05/08/2023*.
[32] Raef Mousheimish, Yehia Taher, Karine Zeitouni, and Michel Dubus. "PACT-ART: Adaptive and context-aware processes for the transportation of artworks". *2015 Digital Heritage International Congress, Digital Heritage 2015* (2015), pp. 347–350. DOI: 10.1109/DIGITALHERITAGE.2015.7419520.
[33] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System" (2008).
[34] openzeppelin.com. *OpenZeppelin*. URL: https://www.openzeppelin.com/. *Date accessed: 05/08/2023*.
[35] polyscan.com. *Polygon PoS Chain Average Gas Price*. URL: https://polygonscan.com/chart/gasprice. *Date accessed: 09/08/2023*.

[36] react.dev. *React*. URL: https://react.dev/. *Date accessed: 05/08/2023*.

[37] rockpi.org. *Rock Pi 4*. URL: https://rockpi.org/rockpi4. *Date accessed: 18/08/2023*.

[38] DAVID SAUNDERS. "Monitoring Shock and Vibration during the Transportation of Paintings". *National Gallery Technical Bulletin* 19 (1998), pp. 64–73. ISSN: 01407430.

[39] Eva Schito and Daniele Testi. "Integrated maps of risk assessment and minimization of multiple risks for artworks in museum environments based on microclimate control". *Building and Environment* 123 (Oct. 2017), pp. 585–600. ISSN: 0360-1323. DOI: 10.1016/J.BUILDENV.2017.07.039.

[40] A H M Shamsuzzoha and Petri T Helo. "Real-time tracking and tracing system: Potentials for the logistics network". *Proceedings of the 2011 international conference on industrial engineering and operations management*. 2011, pp. 22–24.

[41] Elena Sidorova. "The Cyber Turn of the Contemporary Art Market". *Arts 2019, Vol. 8, Page 84* 8.3 (July 2019), p. 84. ISSN: 2076-0752. DOI: 10.3390/ARTS8030084.

[42] soliditylang.org. *Solidity Programming Language*. URL: https://soliditylang.org/. *Date accessed: 04/12/2022*.

[43] Rui Song, Shang Gao, Yubo Song, and Bin Xiao. "ZKDET : A Traceable and Privacy-Preserving Data Exchange Scheme based on Non-Fungible Token and Zero-Knowledge". *Proceedings - International Conference on Distributed Computing Systems* 2022-July (2022), pp. 224–234. DOI: 10.1109/ICDCS54860.2022.00030.

[44] st.com. *B-L462E-CELL1*. URL: https://www.st.com/en/evaluation-tools/b-l462e-cell1.html. *Date accessed: 18/08/2023*.

[45] Gunnar Stefansson and Bernhard Tilanus. "Tracking and tracing: principles and practice". *International Journal of Services Technology and Management* 2.3-4 (2001), pp. 187–206.

[46] tailwindcss.com. *Tailwind CSS*. URL: https://tailwindcss.com/. *Date accessed: 05/08/2023*.

[47] thirdweb.com. *thirdweb*. URL: https://thirdweb.com/. *Date accessed: 02/08/2023*.

[48] Tim-J-Parbs. *Rockfruit Python DHT*. URL: https://github.com/Tim-J-Parbs/Rockfruit_Python_DHT. *Date accessed: 18/08/2023*.

[49] Rupali Vairagade, Leela Bitla, Harshpal H. Judge, Shubham D. Dharpude, and Sarthak S. Kekatpure. "Proposal on NFT Minter for Blockchain-based Art-Work Trading System". *2022 IEEE 11th International Conference on Communication Systems and Network Technologies (CSNT)*. IEEE, Apr. 2022, pp. 571–576. ISBN: 978-1-6654-8038-3. DOI: 10.1109/CSNT54456.2022.9787667.

[50] verisart.com. *Verisart*. URL: https://verisart.com. *Date accessed: 23/08/2023*.

[51] Qin Wang, Rujia Li, Qi Wang, and Shiping Chen. "Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges" (May 2021).

[52] Ziyuan Wang, Lin Yang, Qin Wang, Donghai Liu, Zhiyu Xu, and Shigang Liu. "ArtChain: Blockchain-enabled platform for art marketplace". *Proceedings - 2019 2nd IEEE International Conference on Blockchain, Blockchain 2019* (July 2019), pp. 447–454. DOI: 10.1109/BLOCKCHAIN.2019.00068.

[53] web3py.readthedocs.io. *web3.py*. URL: https://web3py.readthedocs.io/en/stable/. *Date accessed: 02/08/2023*.

[54] welti-furrer.ch. *Welti-Furrer*. URL: https://www.welti-furrer.ch/en/fine-art-shipping. *Date accessed: 22/08/2023*.

[55] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. "Blockchain Technology Overview" (June 2019). DOI: 10.6028/NIST.IR.8202.

[56] Kuo-Hui Yeh, Tomohiro Inagaki, Der-Chen Huang, Ling-Chun Liu, Yong-Yuan Deng, and Chin-Ling Chen. "An Artwork Rental System Based on Blockchain Technology". *Symmetry 2023, Vol. 15, Page 341* 15.2 (Jan. 2023), p. 341. ISSN: 2073-8994. DOI: 10.3390/SYM15020341.

Abbreviations

ABI Application Binary Interface
API Application Programming Interface

CHF Swiss Francs
CSG Communication Systems Group

EVM Ethereum Virtual Machine

GPIO General Purpose Input/Output
GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

ICOM International Council of Museums
ID Identifier
IfI Department of Informatics
IoT Internet of Things

JSON JavaScript Object Notation
JWT JSON Web Token

NFT Non-Fungible Token

REST Representational State Transfer
RPC Remote Procedure Call

SC Smart Contract
SCM Supply Chain Management

UZH University of Zurich

ZKP Zero-Knowledge Proof

List of Figures

List of Tables

Listings

Universität Zürich[UZH]

Institut für Informatik

Declaration of independence for written work

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich,

Signature of student

Appendix A

Contents of the Repositories

The system is composed of four repositories that are managed in a GitHub organization. The organization as well as the repositories can be found at:

`https://github.com/orgs/artis-project/repositories`

The implementational details of each component and its associated repository are found in Chapter 5, where each Section is named after the corresponding repository. Additionally, each repository contains a `README.md` file which contains important information on the component's installation and configuration.

The repository with the name artis-thesis contains the LaTeX source code for this thesis.