SPECK-Hackathon

Hackathon Documentation

by

Philip Heller, Hichem Ben Aoun

Degree Course: Information Systems M.Sc.
Matriculation Number: 2111444, 2480704

Institute of Applied Informatics and Formal Description Methods (AIFB)

KIT Department of Economics and Management

Words: 7500

Advisor:	M. Sc. Niclas Kannengießer
Second Advisor:	
Submitted:	May 28, 2023

Contents

List of Abbreviations

ABI Application Binary Interface.

CII critical information infrastructure.

CLI Command Line Interface.

DX Developer Experience.

GUI Graphical User Interface.

ISR Incremental Static Regeneration.

RFC Request for Comments.

SEO Search Engine Optimization.

SSG Server Side Generation.

SSR Server Side Rendering.

TS typescript.

1 Introduction

SPECK is an innovative blockchain-based supply chain solution, purpose-built to revolutionize the way meat, specifically pig products, are managed throughout their lifecycle. Combining state-of-the-art technology with the core demands of the meat industry, SPECK provides unprecedented transparency, traceability, and efficiency to all stakeholders in the pig production supply chain.

Leveraging the power of the Ethereum network, SPECK utilizes ERC-721, a free and open standard that describes how to build non-fungible or unique tokens on the Ethereum blockchain. Each unique token represents an individual pig or a specific pork product. This ensures an unmatched level of accountability, as each token can be traced from its origin through the entire supply chain.

The innovative use of ERC-721 allows for the individual identification of each pig or pork product, meaning that every single piece of meat can be tracked from farm to fork. The unique token attached to each product contains vital information such as the animal's birthplace, diet, health records, and processing and transportation details. This offers consumers and stakeholders complete visibility and assurance about the quality, safety, and origin of their food products.

SPECK's functionality is not limited to the blockchain backend. It also features a robust and user-friendly front end, designed to make the system accessible to everyone from farmers and processors to retailers and consumers. This interactive interface allows users to effortlessly manage, track, and save pig products at every stage of the supply chain.

The next section delineates the requirements, further subdivided into functional and non-functional aspects. The third section outlines the assumptions made during the project conception, detailing the trust model, Threat model, Project-specific assumptions, and Conceptualization. The Implementation forms the core of the document, exploring the technical elements of the project. It dives into specifics about the techstack, which includes the front-end and smart contract framework, the project Architecture, the process of dockerization, and an overview of the Contracts involved, with a deeper focus on the two smart contracts *OrganizationAuthenticator.sol* and *Speck.sol*. This section also examines the Front-end, dev-pages, and public pages of the project. Future work, including aspects like view restriction, Gas optimization, Decentralized data storage, Change in Organization Data, Transfer proposal, and additional front-end features, is discussed towards the end.

2 Requirements

2.1 Functional Requirements

In the SPECK system, both consumers and organizations are provided with the ability to trace the origin of specific food products, down to the individual animals used in their production. Further, consumers are empowered with the ability to track which organizations participated in the making of specific food products, lending to increased transparency in the supply chain.

This system also caters to the growing demand for information regarding animal welfare, enabling consumers to view the animal-welfare data of the organizations the animals have been through. Similarly, both consumers and organizations can access data about the emissions produced by companies involved in the food products' production, promoting environmentally conscious choices.

Animal-related data, such as the animal's genetics and medication history, are added progressively by companies along the supply chain. This data, along with product-related information, is easily accessible through a graphical user interface, facilitating comprehension and decision-making for both consumers and organizations.

Companies are required to upload animal-related and organization-related data to the SPECK system, enhancing the depth and breadth of information available. Organizations are permitted to add attributes to animals and food products within the SPECK system, fostering a detailed representation of the products. However, it is important to note that they cannot physically delete these attributes, ensuring the integrity of data.

Organization-related data, such as animal-welfare indices and emissions, are updated annually, ensuring the information's relevancy. Additionally, organizations have the capability to trace individual animals and food products, which aids in managing supply chains more effectively.

A crucial aspect of the SPECK system is the requirement for organizations to be authenticated by a trusted third party, known as an authenticator. This process adds another layer of credibility to the information presented within the system.

2.2 Non-functional Requirements

For the SPECK system, Availability and Integrity are paramount. In terms of Availability, it is a requirement that all data managed through the SPECK system is highly available. This ensures that necessary information is accessible when needed, thereby facilitating efficient operations and decision-making processes.

Integrity, on the other hand, covers a few critical aspects. Firstly, transaction data about animals must remain immutable. This safeguards the accuracy of the data and prevents unauthorized alterations. Secondly, any modifications made to animal or organization-related data must be readily identifiable by recipients. This promotes transparency and allows users to be fully aware of any changes made. Lastly, transactions stored in the ledger must also be immutable. This principle maintains the veracity of the ledger and ensures that its records remain reliable and uncompromised.

3 Assumptions

3.1 Trust model

- Companies are incentivized to share data via the SPECK system.
- Companies have their own information systems in which they store all data that should be treated as confidential, including billing data and detailed customer data. We assume such information systems as secure.
- Companies independently take care of the key management of private keys or mnemonics for their wallet
- The consortium of companies in the SPECK system agreed on a set of trusted authenticators which are deposited in the organization list.
- All RFIDs used to identify animals are unique.
- Companies can contact each other via communication channels external to the SPECK system.
- Companies only store correct data(e.g., animal welfare score).
- Companies do not create shadow trees from already exhausted product states (i.e. vertices in product tree)

3.2 Threat model

- Nodes may be arbitrarily unavailable (e.g., due to crashes).
- Companies manipulate data stored in the distributed ledger (e.g., to obfuscate bad animal welfare scores).

- Companies steal the identity of other companies in order to publish fictitious production data, thereby damaging the reputation of the organization concerned in the public eye.
- Companies modify entries in their local database that stores the ledger (e.g., to obfuscate bad animal welfare scores).
- Companies transfer the same product to multiple companies subsequent in the supply chain(e.g., to fake transfers).
- Shadow trees may be created

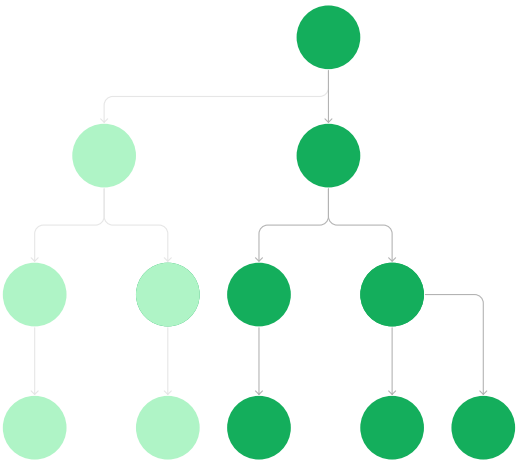

Figure 1: Example of a shadow tree

3.3 Project specific assumptions

The SPECK project was initiated with a few assumptions central to its design and functioning. At the core of our endeavor is the desire to trace and present the entire lifecycle of a pig, from birth to the final product.

However, to make the system more accessible and less complex, we've simplified certain aspects. For instance, we've streamlined the process of transferring a pig from one entity to another (peers), so it's more user-friendly and doesn't become a bottleneck in the system's operation.

Despite our comprehensive approach, we consciously decided to exclude a few elements from the proof of concept (PoC) stage. It's important to note that this exclusion doesn't reflect oversight but a deliberate decision based on various factors such as feasibility,

priority, and project timeline. These elements may be revisited and incorporated in future versions (see 5).

Another key assumption made during the conception of the SPECK project is the level of technical competence of the users, both organizations and individual consumers. The system assumes that users have the necessary technological knowledge to interact with the platform, input data where necessary, and interpret the information provided.

The operation of SPECK also relies on the assumption of stable and robust internet connectivity, as it's a blockchain-based system requiring constant data transfer. Additionally, it's assumed that all participating entities comply with relevant local and international regulations regarding animal welfare, food safety, and data privacy.

Furthermore, SPECK assumes that all participating organizations are comfortable with blockchain technology and recognize its value in ensuring traceability and transparency in the supply chain. And finally, we assume that consumers and organizations hold a deep concern for sustainability and animal welfare, influencing their purchasing or operational decisions based on the data provided by SPECK.

3.4 Conceptualization

During the conceptualization of the project, the team identified four key modules that form the foundation of the system.

The first module is the baseline, which involves managing the pig meat supply chain. This module is implemented in the *Speck.sol* contract, which serves as the core component responsible for creating and tracking unique product tokens, representing individual pigs or specific pork products. It ensures transparency, traceability, and efficient management throughout the lifecycle of the products.

The second module is the organization validation, implemented in the *OrganizationAuthenticator.sol* contract, as mentioned earlier. This module focuses on authenticating and registering organizations participating in the supply chain. It verifies the identity and authorization of organizations to ensure that only validated entities can interact with the system.

The third module is cumulative aggregation, which involves clustering products or animals into groups or clusters. This allows for the registration of products or pigs in bulks, streamlining the process and improving efficiency in managing large quantities of data within the supply chain.

Lastly, the fourth module addresses the confidentiality of certain attributes or product properties. It recognizes the need to protect sensitive information, such as medication details, animal welfare scores, or environmental data. This module ensures that access

to specific attributes or properties is restricted only to authorized entities, maintaining confidentiality while providing transparency across the supply chain.

By conceptualizing the project into these modules, the team establishes a comprehensive framework that addresses various aspects of the pig meat supply chain, including authentication, traceability, efficiency, and data confidentiality.

4 Implementation

This section will describe the process for the creation of a PoC for the SPECK project and discuss choices for the techstack, the architecture and the different containerized components during the implementation of the project.

At the beginning of the hackathon, a demonstrative repository using the Truffle suite (Ganache, Truffle and Drizzle) was provided as entry point and proof of concept to get started [Mikael Beyene et al., 2023]. However, during the early stages of conceptualizing and exploring options (creation of a SPECK-PoC) a few problems and limitations of the exemplary repository were identified:

1. [Mikael Beyene et al., 2023] uses create-react-app [Facebook, 2022] (slower as it still uses webpack instead of modern solutions such as vite or turbopack, fewer features for quality of life and Developer Experience (DX))
2. [Mikael Beyene et al., 2023] uses drizzle which has not been changed in 3-4 years, poorly documented, missing support for typescript (TS)
3. [Mikael Beyene et al., 2023] does not work out of the box and has configuration overhead (has .env file that is not needed but needs to be configured)
4. [Mikael Beyene et al., 2023] does not encapsulate the individual components into own containers (Ganache and Truffle are run in same container which is not best practice resulting in more complexity to manage containers, slower build times)
5. [Mikael Beyene et al., 2023] installs dependencies during runtime instead of during build time (slow cold and hot starts)

Therefore, the project dependencies and virtualization through Docker (ultimately also the architecture) were reworked to find fitting solutions addressing the problems described above. The individual issues live in different conceptual layers described in the following sections (see 4.1, 4.2 and 4.3). An early version of an exemplary application using the updated environment is tagged in [Philip Heller and Hichem Aoun, 2023a][1]. The final

[1]Note, that a few things were changed in the final implementation of SPECK (see tag message).

submission of the SPECK project is located in the repository in the main branch [Philip Heller and Hichem Aoun, 2023b].

4.1 Techstack

In the exploration phase for a DLT, more specifically a Solidity project, different technologies have been looked at to find a suitable solution. These can be separated into the frontend and the blockchain.

4.1.1 Frontend

For the frontend, Nextjs [Vercel, 2023] was chosen. It supports TS out of the box and leverages technologies used in large scale applications for Search Engine Optimization (SEO) such as Server Side Generation (SSG), Server Side Rendering (SSR) and Incremental Static Regeneration (ISR). It also features a built in router, automatic code splitting and a rich ecosystem of plugins and popular tools. Though many of these tools are not strictly necessary for SPECK, these attributes made Nextjs a popular framework with a large community, a stable environment and integration of newest innovations in React. A great example is the use of component level client and server rendering that was introduced in React 18. Nextjs is the first large meta framework that implemented it as beta in a Request for Comments (RFC).
To simplify the styling process, Tailwindcss [Tailwind, 2023] was used. Tailwindcss is a utility class system that shortens the amount of css and helps creating consistent styling throughout the application.
For the interaction with web3-components, an alternative for drizzle was chosen. Unfortunately, drizzle does not support TS and is poorly documented. It has also not seen any development activity lately. As alternative, Wagmi [Wagmi, 2023] was chosen. It supports TS and delivers a large set of hooks that provide several updates in state kept track of by Wagmi. It reduces the complexity of interaction with smart contracts and the blockchain significantly resulting in greater DX.

4.1.2 Smart Contract Framework

There are currently two large and popular frameworks commonly used and referenced for the development of web3 applications: Hardhat and Truffle. Both frameworks deliver tools and interfaces for the development of smart contracts and complex applications. One advantage of hardhat is it's modularity and it's out of the box support for TS. Truffle on the other hand comes with it's own suite of tools such as Ganache [ConsenSys Software Inc., 2023] that are tightly coupled.

Due to a high degree of feature parity and in the name maintainability the Truffle suite was chosen for smart contract development. Thereby, the example project [Mikael Beyene et al., 2023] resembles the structure a bit more reducing complexity for developers who already work at the critical information infrastructure (CII).

Choosing other dependencies solved the issues 1-2. Issues 3-5 are concerned with the dockerization and devops which are discussed in the following sections 4.2 and 4.3.

4.2 Architecture

The technologies listed above result in a cohesive mono repository. Figure 2 depicts the individual components and it's dependencies. Ganache serves as the local blockchain emulator, providing a simulated Ethereum network for testing and development purposes. It creates a sandbox environment where developers can create and manage test accounts, deploy smart contracts, and execute transactions. It comes as a Graphical User Interface (GUI) for users or as a Command Line Interface (CLI). This is more relevant for the dockerization in which Ganache is used through the CLI while developers can also opt to use a version on their host machine. Truffle t simplifies the creation, compilation, testing, and deployment of smart contracts on the Ethereum network. Truffle's integration with Ganache enables developers to deploy and interact with their smart contracts in the local blockchain environment without manually passing the Application Binary Interfaces (ABIs) and binaries. It also provides the ABIs for the frontend. Last but not least, Nextjs with Wagmi is used to create an extensive and accessible web application. The GUI is accessible through a number of browsers. Wagmi uses the ABIs and creates easy to use, typesafe hooks for the interaction with the smart contracts on the blockchain.

4.3 Dockerization

Dockerizing the entire project helps the project in a couple of ways:

1. **Easy reproducibility and consistentcy**
   Docker permits to easily create containerized environments and encapsulates all the necessary dependencies and configuations required in the project. Builds will be reproducible across different environments which eliminates "works on this machine" issues.

2. **Simplified setup and configuration**
   Setting up the entire project environment with different frameworks and tools is often time consuming and error-prone. The dockerized project will eliminate manual installations and configurations and get developers started quickly (reducing

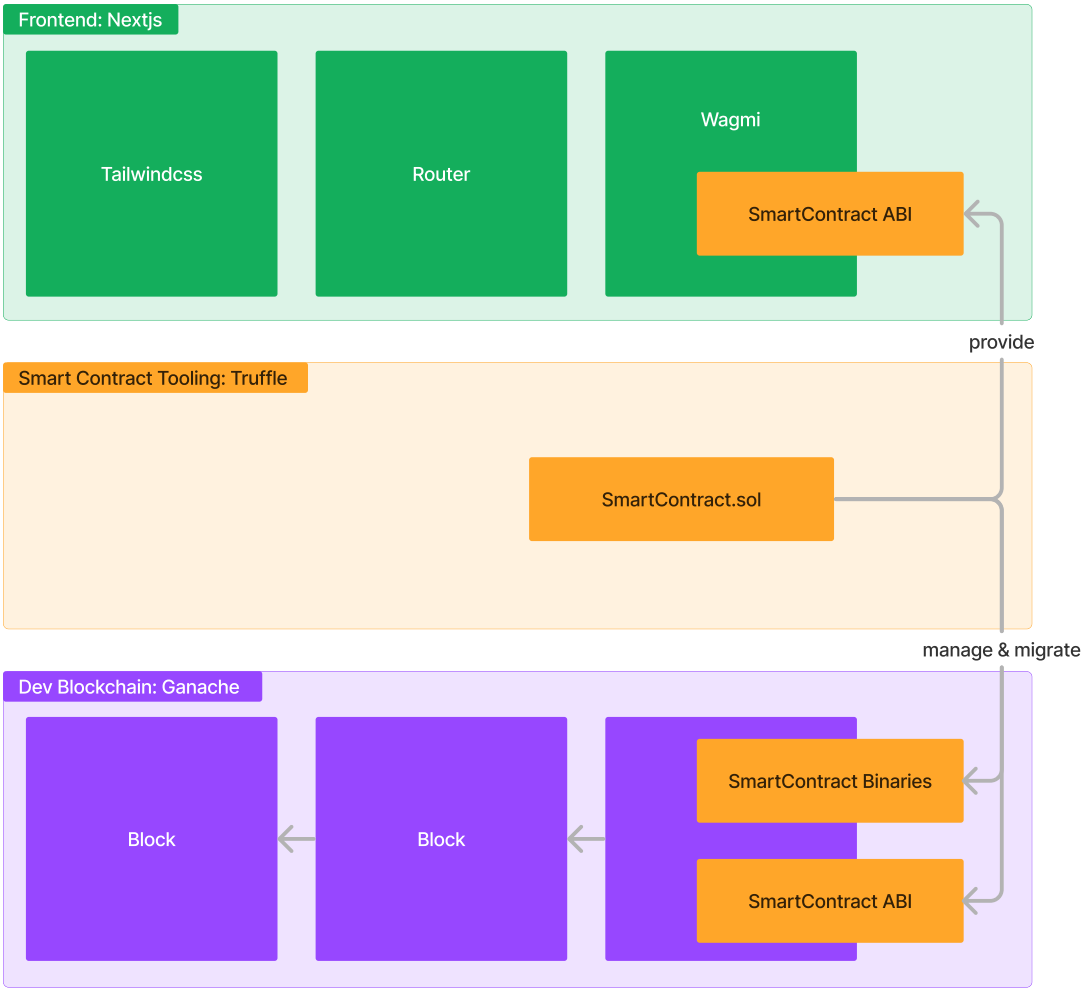

Figure 2: Architecture of SPECK

onboarding time) through a standardized setup process.

3. **Dependency management** Using different dependencies of a piece of software on the same host machine can be challenging. Docker simplifies the versioning and prevents issues with dependency conflicts of same dependencies. Each container has it's own environment that is independent.

4. **Orchestration and spinning up the project**
   Spinning up a development environment with multiple interacting components is time consuming. Here, tools such as Docker Compose help by allowing to spin up the entire environment in a single command (docker-compose up).

Most of the configuration is handled through the docker-compose.yml file in the repository [Philip Heller and Hichem Aoun, 2023b]. It specifies the services, configurations,

networking and volumes for persistent data. With this setup, developers can easily start the development with either a clean blockchain or with data persisted from last development session.

4.4 Contracts

The smart contracts used in this project are written in the programming language *Solidity*. The functionalities are separated into two contracts based on the separation of concerns paradigm. The *OrganizationAuthenticator.sol* contract handles the registration of organizations and their permission participate in the *SPECK* system. This architecture allows other contracts or services to use the information stored on the *OrganizationAuthenticator.sol* contract to allow or restrict access to certain functionalities or information. The *Speck.sol* is used for the saving and retrieving of product data and its history. Permission for the creation of products is derived from the registration data of the *OrganizationAuthenticator.sol*.

4.4.1 OrganizationAuthenticator

The *OrganizationAuthenticator.sol* contract is used to authenticate and register organizations. It allows organizations to request registration, and the contract owner can approve the registration requests. The contract also provides functions to retrieve organization data, check registration status, and manage organization requests.

The *OrganizationAuthenticator.sol* smart contract serves a crucial role in the SPECK system as the gateway for organizations to participate in the network. This contract is employed to authenticate and register organizations, ensuring that only authorized entities have access and can contribute to the system after requesting registration (see figure 8).

Key elements of this contract include:

1. **Data Structs**
   The contract uses the data structure `Organization` struct to encapsulate and manage key information about an organization, including its name, type, email, address, and scores pertaining to animal welfare and environmental impact, among other details.

2. **Organization Registration**
   A crucial function of this contract is to enable organizations to request registration. Upon request, the contract generates a unique ID for the new organization, stores their data, and marks the registration request as active. It then emits an event notifying that registration has been requested.

3. **Organization Authentication**
   The contract contains methods to authenticate organizations based on their ID or Ethereum address. This function is important in maintaining the security and integrity of the *SPECK* ecosystem.

4. **Registration Approval**
   The contract also allows the contract owner to register an organization, given that the organization has an active registration request. Once registered, the organization's registration request is deactivated, and it is officially marked as registered.

5. **Data Retrieval**
   Functions are provided to retrieve data about an organization, either based on its Ethereum address or the sender's address. Additionally, a list of all organizations that have requested registration can also be retrieved.

6. **Total Request and Registration**
   The contract keeps track of the total number of organizations that have requested registration and those that have been registered.

All of these functionalities contribute to the contract's primary goal of maintaining a trusted network of authenticated organizations that participate in the SPECK system, reinforcing the system's core values of transparency and accountability.

A detailed documentation of *OrganizationAuthenticator.sol* is included in the appendix A.1 or in the GitLab repository ([Philip Heller and Hichem Aoun, 2023b]).

4.4.2 Speck

The *Speck.sol* is a smart contract written in Solidity that represents a product as an ERC721 token, an Ethereum standard for non-fungible tokens. It allows authenticated organizations, as verified by the contract *OrganizationAuthenticator.sol*, to create and transfer these tokens, which also contain detailed metadata attributes of a product such as RFID, genetics, gender, slaughter method, findings, pH value, previous product, product type, animal weight, fat percentage, feed, medication, and timestamp. The contract utilizes the utility library "Counters" for handling token ID increments and has a "NewProductCreated" event that gets emitted every time a new product is created. Furthermore, the contract provides various functions for interacting with the products including creating new products, transferring products, fetching product data, fetching multiple products data at once, fetching product history, and calculating the depth of a product's history.

Key elements of this contract include:

1. **ERC721 Token**
   The contract utilizes the ERC721 standard to create non-fungible tokens, representing unique products.

2. **OrganizationAuthenticator**
   This is an authentication system implemented to ensure only registered organizations can create and transfer products.

3. **Modifiers**
   The contract uses modifiers such as `onlyRegistered` and `previousProductCheck` to control access and validate the existence and ownership of previous products in a chain.

4. **Product Struct**
   This data structure is used to store detailed data about a product.

5. **Mappings**
   The contract uses mappings to keep track of product data (`_products`) and their owners (`_tokenOwner`).

6. **Events** The contract emits a `NewProductCreated` event every time a new product is created.

7. **Functions**
   There are several functions that enable interaction with the contract and its data, including:

   (a) createNewProduct(): Creates a new product and increments the token ID.
   (b) transferProduct(): Transfers ownership of a product to a new address.
   (c) getProductData(): Returns the details of a specific product.
   (d) getMultipleProductData(): Returns details of multiple products.
   (e) getProductHistory(): Returns the full history of a product.
   (f) getHistoryDepth(): Calculates the depth of a product's history.
   (g) totalProductAmount(): Returns the total number of products created.
   (h) getOwnerOf(): Returns the owner of a specific product.

8. **Constructor**
   The contract constructor sets the name and symbol of the ERC721 token and initializes the OrganizationAuthenticator contract.

A detailed documentation of *Speck.sol* is included in the appendix A.2 or in the GitLab repository ([Philip Heller and Hichem Aoun, 2023b]).

4.5 Contract interaction

The Speck contract and the OrganizationAuthenticator contract interact with each other to enable secure and authenticated product management. The OrganizationAuthenticator contract serves as an external authentication system, responsible for verifying the registration and authorization of organizations. It provides the necessary logic to authenticate organizations on the blockchain. The Speck contract utilizes the OrganizationAuthenticator contract by integrating it as an instance within its own contract. Through the onlyRegistered modifier, the Speck contract leverages the authentication functionality of the OrganizationAuthenticator contract to validate the permissions of organizations attempting to create or transfer products. This interaction ensures that only authenticated organizations are allowed to perform specific operations within the Speck contract, establishing a secure and trusted environment for managing unique product tokens.

4.6 Front-end

Though the SPECK project did not necessarily include a frontend for the creation of new products and the management of such products for the organizations feeding the system with data (any entity involved in the product chain), the developers opted to create a hidden directory for these kinds of

As described in the sections 3.3 and 3.4 the main goal was to be able to create a fully working PoC that enables the representation of the entire lifecycle of a product (pig). Though the creation of pages to manage these products is specifically not a requirement for the PoC, a very simple and rudimentary version of these kind of pages was created along the public pages in the mono repository. This way, users can easily create and preview an example of their own choosing. Hereafter, the pages that are linked from the root page and meant to be used by a consumer are called "public" or "publicly available" pages. The other pages, that are meant for development and demonstration purposes will be referred to as "development" or in short as "dev" pages. As described in section 5.5 the actual interaction with the blockchain by entities involved in the value creation will either use their own solution or a separate repository for their specific features will have to be created (this could also be a future project that starts with what is made available through the dev pages). Since the public pages are only relevant as soon as data is available on the blockchain, this documentation will firstly introduce the dev pages and afterwards describe the public pages. An overview of the pages and their flow is depicted in the figures 3 and 4.

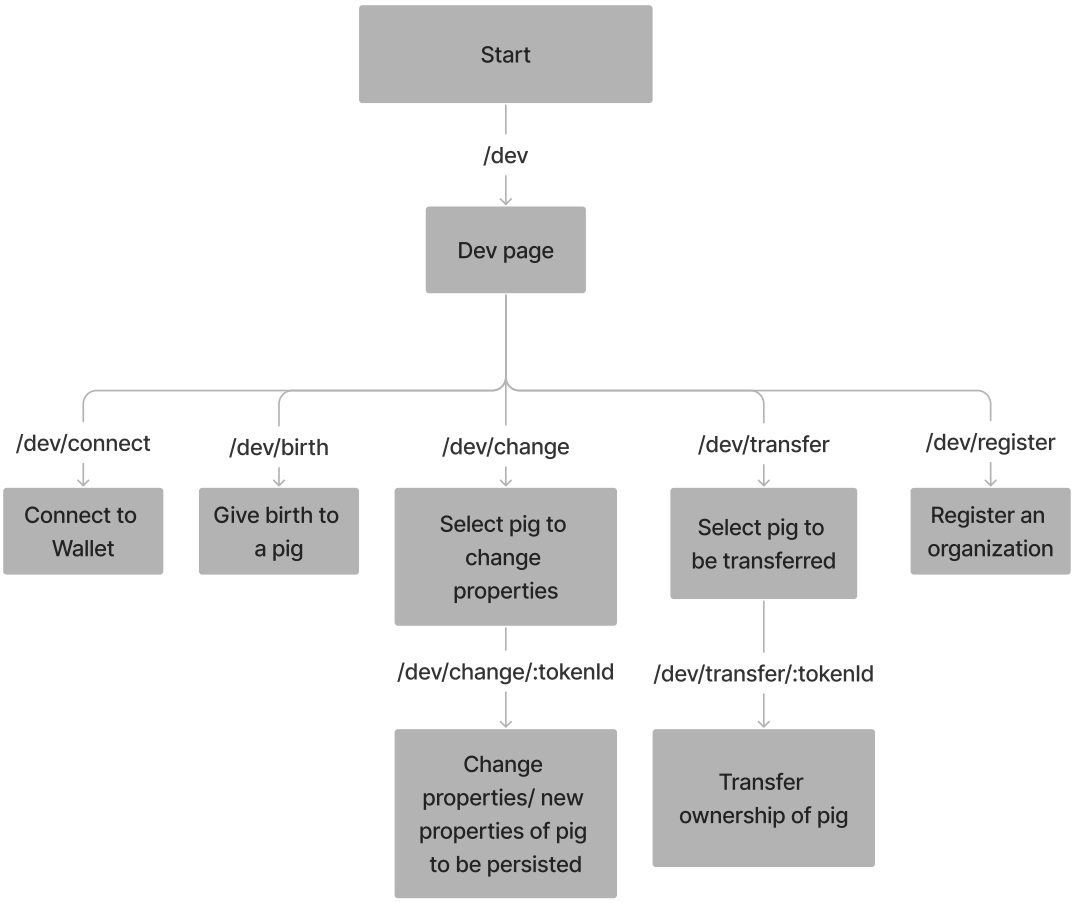

Figure 3: Dev pages

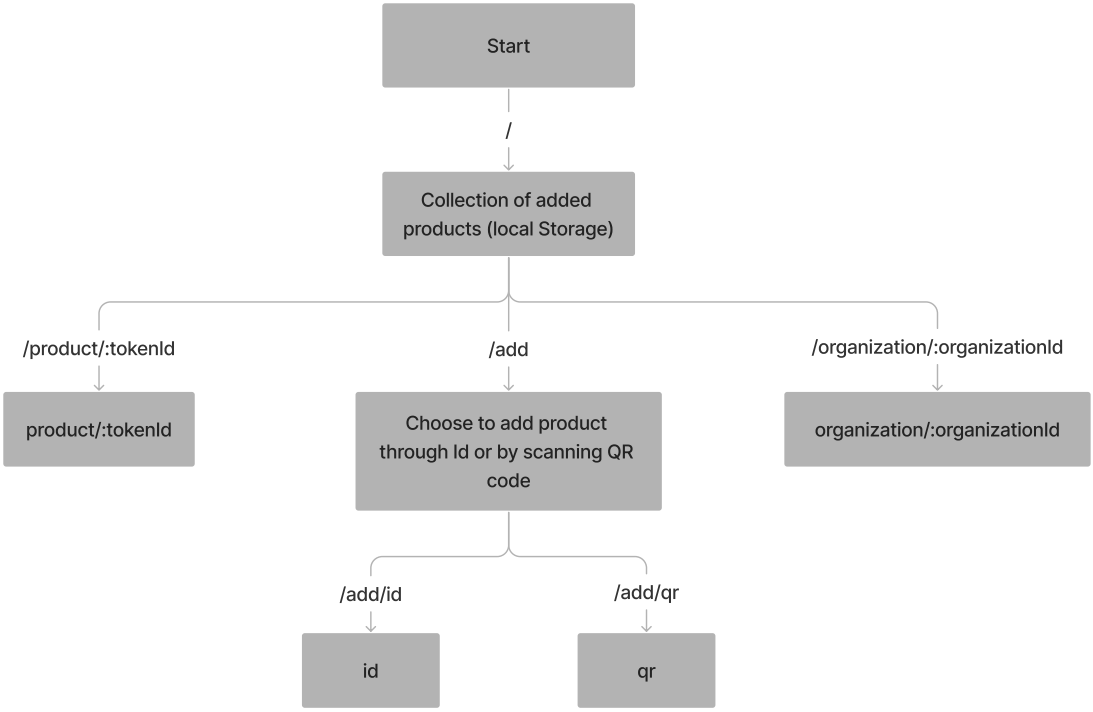

Figure 4: Public pages

4.7 Dev Pages

The dev pages contain a series of pages that are necessary to quickly interact with the smart contract. In theory, these interaction could also have been modeled and executed through functions in Truffle, however, since any transactions and presentation layer tasks are performed through the web app's GUI, it is a quick and easy way to perform all of these tasks in the browser. Additionally, this could serve as an entry point for an implementation that would allow producers to implement a GUI of their own.

The first page listed in the development pages is the "connect" page. It allows producers to connect their wallet and use it for any interactions with the smart contract. Note, that theoretically the consumer would not need a wallet for the public pages as the only actions performed in these pages are "read" actions. However, to keep it simple, the current implementation always requires a connection. This could be adapted through Wagmi's providers.

As soon as a user connected their wallet, they are able to:

1. Give birth to a random pig [*/dev/birth*; leverages the *createNewProduct* method]
2. Select a pig to change properties [*/dev/change*; leverages the *createNewProduct*

method]

3. Select a pig to be transferred to another wallet [*/dev/transfer*; leverages the *transferProduct* method]

4. Register themselves as an organization (which will allow consumers to see the organization's details) [*/dev/register*; leverages the *requestRegistration* method]

If a developer were to clone a fresh version of the repo and spins up all the containers, a minimum set of actions that would need to be performed to see something relevant would be to first connect their wallet (1.), optionally register themselves as organization (4.[2]) and finally give birth to a (random) product/pig. Changing properties and transferring ownership to another entity will show lead to a more realistic supply chain that will also be visualized in the timeline for the consumer.

4.8 Public Pages

The public pages are consumer facing pages that are meant to be used to keep track of any products acquired in the physical world. As soon as the product is physically available, a consumer has the option to either scan a QR code (not yet implemented, see 5.5) or to input the product ID (tokenId of the product on the blockchain) to add the product to the watch list. The products that are kept in the watchlist are currently stored in local storage. Persisting the products in such a way can be replaced by any arbitrary solution to persist data such as traditional databases or chrome's sync storage. Any product saved in the product overview can be viewed in a timeline. The timeline shows all the individual entries of the product in time. From there, a consumer may also see the organizational data that was registered as described in the last section 4.7. A few public pages are appended, see figures 5, 6 and 7.

5 Future Work

In the following section potential enhancements and adaptations for the future iterations of the SPECK system will be explored. We will delve into the options available for the system's expansion, refinement, and further integration. These future possibilities aren't fixed paths but are intended to open up a dialogue about the ongoing evolution of SPECK, aligning it with industry advancements, user expectations, and emerging trends in the food supply chain management. Each option explores the scope of possible improvements,

[2]If not registered, consumers will see an error message when inspecting the organization associated with the product

anticipating future demands and preparing SPECK to meet them head on. This iterative approach aims to ensure that SPECK continues to deliver on its promise of transparency, efficiency, and accountability in the meat industry's supply chain. The presented list of solutions is not extensive an only presents a selected amount improvements.

5.1 View Restriction

In future versions of the SPECK system, confidentiality requirements would need to satisfy the following non-functional requirements. While every entity in the SPECK system can access data stored in the distributed ledger, certain exceptions are in place to protect sensitive information.

Specifically, details about the medication of animals during both breeding and mast stages are confidential and solely accessible to companies within the coalition. This is likely due to the sensitive nature of such data and the need to ensure proper use of this information.

Similarly, the weight of animals during the mast stage is also regarded as sensitive information and is only available to coalition companies. Such measures likely reflect the competitive nature of the industry and the importance of maintaining certain data points as confidential.

Post-slaughter data points including findings, pH-value, and fat percentage after slaughter, are also treated with strict confidentiality. These specific data points are vital indicators of the quality and health of the slaughtered animals, and are therefore accessible only to companies within the coalition. This ensures that the information is responsibly managed and used for the appropriate purposes within the industry.

5.2 Gas Optimization & Decentralized Data Storage

One area of future work that can greatly optimize the SPECK system involves the strategic management of data storage in the context of the smart contract. Currently, data is saved directly on the smart contract, which while secure, can result in high gas consumption due to the inherent costs associated with storing and manipulating data on the Ethereum network. A potential solution to this issue is the integration of a decentralized data storage solution such as the InterPlanetary File System (IPFS). IPFS can offer a scalable and efficient method for storing data, allowing the system to maintain its decentralized nature while significantly reducing gas consumption. Instead of storing all data directly on the smart contract, data could be stored on IPFS, and the smart contract would only need to store the IPFS hash pointer to that data. This approach would retain the immutability and transparency of the data, while also improving the overall efficiency and cost-effectiveness of the SPECK system.

5.3 Change Organization Data

As part of our continuous improvement efforts, one aspect that requires attention pertains to the flexibility of modifying organization data stored in the OrganizationAuthenticator.sol contract. In the current state of the system, organization data can only be entered once during the registration request process, with no provision for subsequent changes. This rigid model doesn't account for dynamic variables like animal welfare scores or environmental scores, which are subject to fluctuation over time. Therefore, it is crucial to consider a system that allows for the updating of such attributes.

The proposed solution is not about allowing direct alterations to the existing data but introducing an appending system. With this system in place, new data would not overwrite the old, but would instead be added as a new entry, effectively creating a historical record of the organization's data. This would maintain the integrity of the historical data while allowing for real-time updates, thus keeping the system's data reflective of the current status of the organization. Consequently, this system would ensure that the SPECK system remains an accurate and reliable tool for all stakeholders, upholding its mission for transparency and accountability in the meat industry supply chain.

5.4 Transfer Proposal

Future improvements to the SPECK system should consider implementing a transfer proposal system within the Speck.sol contract, to ensure transparency and control during the transfer of product ownership. Under the current system, transfers can occur without explicit agreement from both sides, potentially leading to disputes and inconsistencies. A transfer proposal system would allow the current owner to propose a transfer, including all relevant product data and potentially the sale price. The receiving party would then have the opportunity to review and accept this proposal before the transfer is executed.

Incorporating this system into the Speck.sol contract would necessitate a new data structure to store and manage these proposals. This enhancement would not only streamline the process of ownership transfers but also bolster the transparency and accountability of transactions within the system. Ensuring that both parties have a clear understanding and acceptance of the terms of transfer is crucial to maintaining trust and reliability within the SPECK ecosystem.

5.5 Frontend features

Looking towards the future, several key enhancements have been identified to further improve the SPECK system's functionality, usability, and user experience. Firstly, inte-

grating an automatic clearing function for the local storage after a set time period would be beneficial. This feature would ensure efficient use of storage resources and maintain optimal system performance.

Secondly, implementing a QR code scanner into the system could significantly streamline data entry and retrieval processes, making it more user-friendly, especially for consumers wanting to quickly access information about a product.

Thirdly, creating a separate repository specifically for producers could provide them with a tailored platform, meeting their unique needs and requirements. This dedicated space would offer producers enhanced usability and an improved, industry-specific experience.

Lastly, a few other important additions have been suggested to further personalize and improve user experience. These include the option for users to use an account to sync data across devices, providing continuity and flexibility. Additionally, introducing settings to customize individual user experiences, such as controlling how long a product is kept in storage, or user interface preferences like dark mode, would give users greater control over their interaction with the SPECK system.

A Appendix

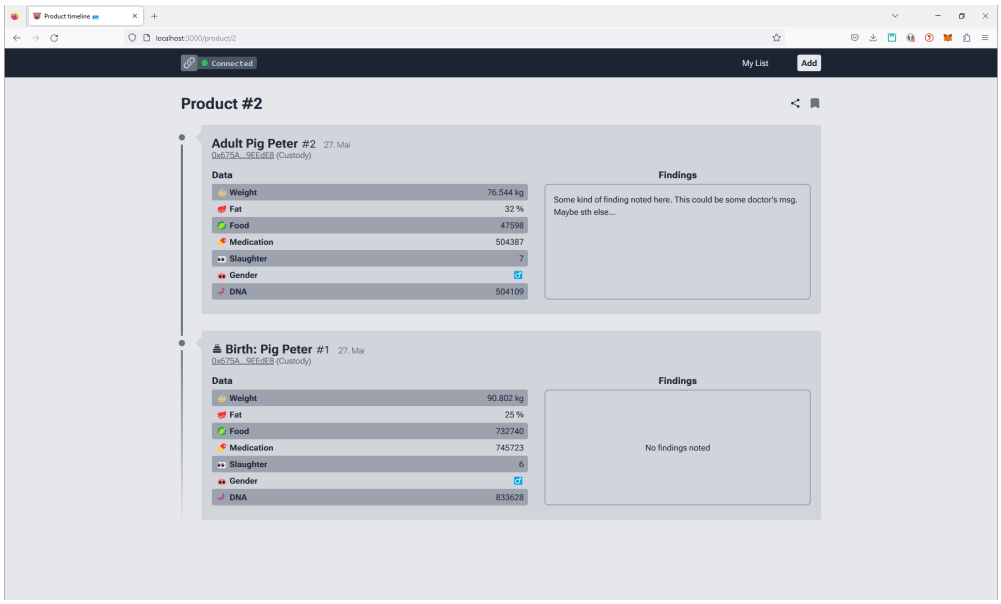

Figure 5: Timeline of a product's chain

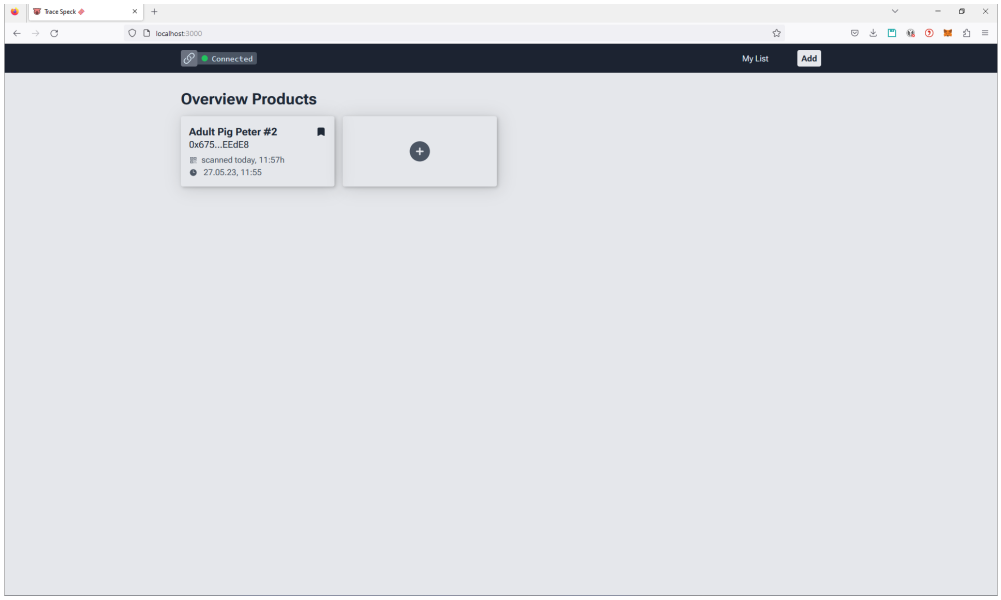

Figure 6: Collection of pigs that added to watchlist by consumer

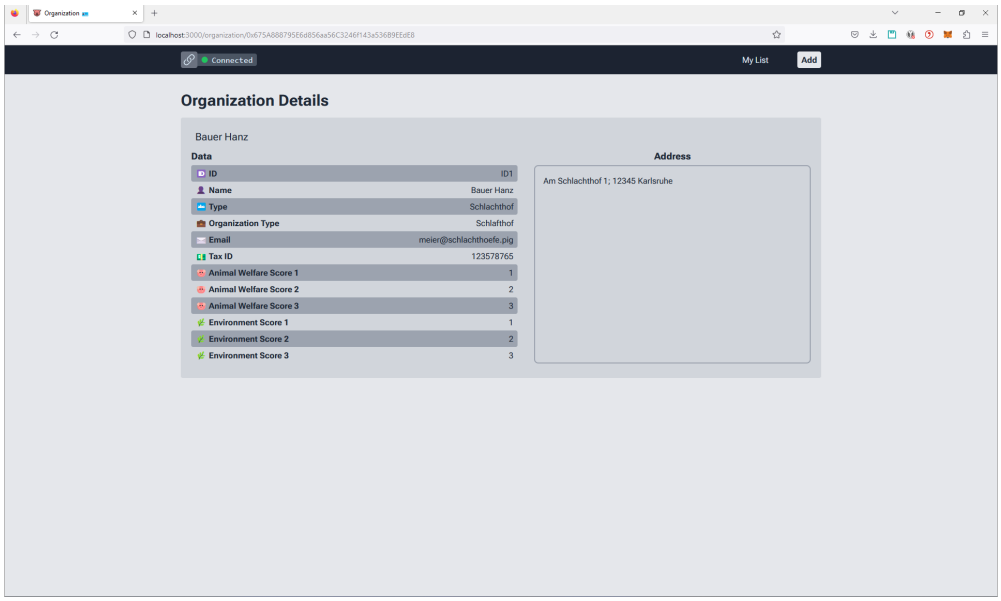

Figure 7: Registered organization displayed to consumer

A.1 OrganizationAutheticator.sol

Imports

- "@openzeppelin/contracts/token/ERC721/ERC721.sol"
- "@openzeppelin/contracts/utils/Counters.sol"
- "@openzeppelin/contracts/access/Ownable.sol"

State Variables

Name	Description
`_orgIds`	A counter to track the next open token ID
`_organizationAuthenticator`	A counter to track the number of registration requests.
`_registeredAmount`	A counter to track the number of registered organizations.
`_registered`	A mapping to store the registration status of organizations based on their ID.
`_organizationData`	A mapping to store organization data based on their ID.
`_registrationRequested`	A mapping to track the registration request status of organizations based on their ID.
`_addressToId`	A mapping to associate an organization's Ethereum address with its ID.
`_registrationRequestedArray`	An array to store the IDs of organizations that have requested registration.

Events

- `_Authenticate(string _msg)`: Event emitted when an authentication is performed.
- `_RegistrationRequested(address indexed requestAddress)`: Event emitted when a registration request is made.

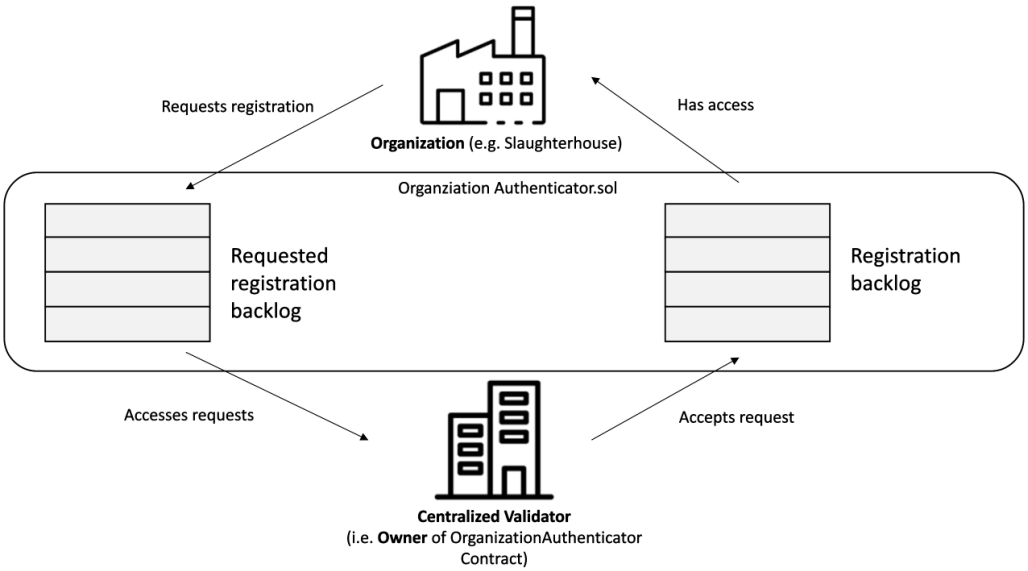

Figure 8: Registration process of *OrganizationAuthenticator.sol*

Structs

`Organization`: A struct to hold organization data

Attribute	Datatype
`id`	string
`name`	string
`type_`	string
`organization_type`	string
`email`	string
`institution_type`	string
`address_`	string
`tax_id`	uint256
`animal_welfare_score_1`	uint256
`animal_welfare_score_2`	uint256
`animal_welfare_score_3`	uint256
`environment_score_1`	uint256
`environment_score_2`	uint256
`environment_score_3`	uint256
`creation_right`	bool

Modifiers

- `_onlyOwner` *(from ERC721)*: Modifier that restricts access to the contract owner.

Methods

Name		Description
`authenticateById`	**@dev**	Check if the organization with the given ID is registered.
	@params	`_orgId`: ID of the organization to check authentication status.
	@returns	A boolean indicating if the organization is registered or not.
`authenticate`	**@dev**	Check if the organization with the given address is registered.
	@params	`_address`: Address of the organization to check authentication status.
	@returns	A boolean indicating if the organization is registered or not.
`requestRegistration`	**@dev**	Request registration for a new organization.
	@params	`_data`: Struct containing organization data to register
`register`	**@dev**	Register an organization.
	@params	`_orgId`: ID of the organization to register
`getRequestedRegistrations`	**@dev**	Get a list of organization requests.
	@returns	An array of organizations
`getMyData`	**@dev**	Get the organization data from *msg.sender*
	@returns	Organization data of the `msg.sender`
`amIRegistered`	**@dev**	Check if the *msg.sender* is registered
	@returns	A boolean indicating if the `msg.sender` is registered or not
`getOrganizationDataByAddress`	**@dev**	Get the organization data of a given address
	@params	`_address`: Address of the organization to retrieve data for
	@returns	Organization data of the given address
`totalRequestedOrganizationAmount`	**@dev**	Get the total number of organization registration requests
	@returns	The total number of organization registration requests
`totalRegisteredOrganizationAmount`	**@dev**	Get the total number of registered organizations
	@returns	The total number of registered organizations

A.2 Speck.sol

Imports

- "@openzeppelin/contracts/token/ERC721/ERC721.sol"
- "@openzeppelin/contracts/utils/Counters.sol"
- "@openzeppelin/contracts/access/Ownable.sol"

State Variables

Name	Description
`_tokenIds`	A counter to track the next open token ID.
`_organizationAuthenticator`	Instance of the OrganizationAuthenticator contract.
`_products`	A mapping for storing product data.
`_tokenOwner`	A mapping for storing the owner of a product.

Events

- `_NewProductCreated(uint256 indexed tokenId, address indexed owner)`: Event emitted when a new product is created

Structs

`Product`: Struct for storing product information

Attribute	Datatype
`id`	string
`rfid`	string
`genetics`	string
`gender`	uint256
`slaughter_method`	uint256
`findings`	string
`ph_value`	uint256
`product_type`	string
`animal_weight_g`	uint256
`fat_percentage`	uint256
`feed`	string
`medication`	string
`timestamp`	string

Modifiers

- `_onlyRegistered`: Modifier that restricts function calls to authenticated organizations.
- `_previousProductCheck`: Modifier that checks if the previous product in the chain exists and belongs to the caller.

Methods

Name	@dev	Description
`createNewProduct`	**@params**	`_product_data`: The Product struct containing the details of the new product to be created.
`transferProduct`	**@params**	`_product_data`: The Product struct containing the details of the product to be transferred.
	@params	*_to*: The address of the new owner of the product.
`getProductData`	**@params**	`_tokenId`: The ID of the product to retrieve data for.
	@returns	A tuple containing the Product struct of the product, its token ID, and its current owner.
`getMultipleProductData`	**@params**	`_token_ids`: An array of product IDs to retrieve data for.
	@returns	A tuple containing arrays of Product structs, token IDs, and current owners for the specified products.
`getProductHistory`	**@params**	`_tokenId`: The ID of the product to retrieve the history for.
	@returns	A tuple containing arrays of Product structs, token IDs, and current owners for each version of the product in reverse chronological order.
`getHistoryDepth`	**@params**	`_tokenId`: The ID of the product to calculate the history depth for.
	@params	`_counter`: A counter variable used to keep track of the depth.
	@returns	The depth of the product history.
`totalProductAmount`	**@returns**	The total number of products.
`getOwnerOf`	**@params**	`_tokenId`: The ID of the product to retrieve the owner for.
	@returns	The address of the current owner of the product.

References

[ConsenSys Software Inc., 2023] ConsenSys Software Inc. (25/05/2023). Ganache - truffle suite.

[Facebook, 2022] Facebook (24/03/2022). Create react app.

[Mikael Beyene et al., 2023] Mikael Beyene, Niclas Kannengiesser, Benjamin Sturm, and David Jin (26/05/2023). mikael.beyene / drizzle-with-events · gitlab.

[Philip Heller and Hichem Aoun, 2023a] Philip Heller and Hichem Aoun (26/05/2023a). Files · exampleproject · uzmzu / speck hackathon · gitlab.

[Philip Heller and Hichem Aoun, 2023b] Philip Heller and Hichem Aoun (26/05/2023b). Speck hackathon · gitlab.

[Tailwind, 2023] Tailwind (27/05/2023). Tailwind css - rapidly build modern websites without ever leaving your html.

[Vercel, 2023] Vercel (27/05/2023). Next.js by vercel - the react framework.

[Wagmi, 2023] Wagmi (27/05/2023). wagmi: React hooks for ethereum.