ESTG

TECHNOLOGIES FOR PROCESS TRACEABILITY IN THE SHIPBUILDING INDUSTRY

2023

INSTITUTO POLITÉCNICO DE VIANA DO CASTELO

TECHNOLOGIES FOR PROCESS TRACEABILITY IN THE SHIPBUILDING INDUSTRY

The West Sea Viana Shipyard case study

Pedro Xavier Mendes Araújo

Escola Superior de Tecnologia e Gestão

INSTITUTO POLITÉCNICO DE VIANA DO CASTELO

Technologies for process traceability in the shipbuilding industry: The West Sea Viana Shipyard Case Study

Tecnologias para rastreabilidade de processos na indústria naval: O caso de estudo dos estaleiros West Sea Viana Shipyard

Pedro Xavier Mendes Araújo

Mestrado em Engenharia Informática
Escola Superior de Tecnologia e Gestão

Orientador: Prof. Doutor Sérgio Ivan Lopes
Orientador: Prof. Doutor António Miguel Rosado da Cruz

May 30, 2023

Resumo

A rastreabilidade de materiais e elementos em qualquer tipo de indústria é essencial e de grande valor, embora atualmente esteja a ganhar ainda mais importância devido às fábricas da Indústria 5.0. Isto é importante no contexto do fabrico de qualquer produto, isto porque, permite que o utilizador conheça a origem de um determinado produto, seja uma matéria-prima ou um componente fabricado através outros elementos, posteriormente fabricados. Esta cadeia de informações rastreáveis pode ser mantida através de várias tecnologias, uma das quais é o 'blockchain'. Este conceito de rastreabilidade é aplicado em qualquer tipo de indústria de alguma forma, mas por vezes existem métodos mais adequados que podem até acrescentar valor ao produto final.

Portanto, este processo permite a rastreabilidade dos componentes de produtos em toda a cadeia de fabrico, desde a matéria-prima até o produto final, passando por cada componente intermediário e etapa do processo, em todos os participantes industriais da cadeia de fabrico. As informações geradas durante o processo de fabrico, incluindo certificados e resultados de inspeção, também são registadas e, portanto, rastreáveis.

Na indústria naval, é essencial garantir altos níveis de qualidade durante o processo de fabrico. Para isso, é importante selecionar os métodos e tecnologias adequados para a rastreabilidade dos processos, considerando as exigências e restrições específicas da indústria naval. Desta forma, o cliente pode ter as informações de qualquer material, componente, peça e bloco do navio de forma simples e confiável. A rastreabilidade do processo pode acrescentar valor ao navio, pois todas as informações rastreáveis podem ser facilmente transferidas para o cliente final, incluindo todos os certificados e documentos necessários.

Nesta tese é proposta uma arquitetura de sistema centrada em tecnologia 'blockchain' que promove a rastreabilidade dos processos de produção aplicados à indústria naval, com base no Estaleiro West Sea Viana.

O sistema será implementado como um caso de estudo no processo de fabrico de um bloco de construção naval e, portanto, prontamente extensível ao processo geral de fabrico de navios, contribuindo assim com uma 'blockchain' contendo ou referindo-se a todos os documentos relevantes (certificados, garantias, etc.) e outros dados de fabrico pertinentes que podem ser entregues ao cliente junto com o navio, aumentando assim a confiabilidade. Além disto, é também feita a implementação de duas provas de conceito de interfaces visuais, tudo isto é testado com informação proveniente do Estaleiro West Sea Viana e validado pelos mesmos.

Abstract

Traceability of materials and elements in any type of industry is essential and of great value, although it is currently gaining even more importance due to the factories of Industry 5.0. This is important in the context of the manufacture of any product. This allows the user to know the origin of a particular element, whether it is a raw material or a component made of other previously manufactured elements. This chain of traceable information can be persisted by several technologies, one of which is the blockchain. This concept of traceability is applied in any type of industry in some way, but sometimes there are more appropriate methods that can even add value to the final product.

Therefore, this process enables the tracking of products' components throughout the chain of production, from raw materials to the final product, passing through each intermediate component and process stage, on every industrial player in the chain. The information generated through the manufacturing process, including certificates and inspection results, is registered and thus trackable.

In the shipbuilding industry, the process of building a ship is complex, has several stages, and it is important to guarantee high-quality levels during the manufacturing process. For this, the methods and technologies adopted for process traceability must be selected, having in mind the specific requirements and constraints of the shipbuilding industry in the most efficient way. In addition, process traceability may add value to the ship, as all the traceable information may be easily transferred to the end customer. This way, the customer can have the information of any material, component, part, and block of the ship easily and reliably, as well as all the certificates and documents necessary for any process.

This thesis proposes a blockchain-centered system architecture that fosters the traceability of the production processes applied to the shipbuilding industry, based on the West Sea Viana Shipyard.

The system will be implemented as a case study in the manufacturing process of a shipbuilding block, and therefore readily extendable to the overall ship manufacturing process, thus contributing with a blockchain containing or referring to all relevant documents (certificates, warranties, etc.) and other pertinent manufacturing data that can be delivered to the customer together with the ship, and thus increasing trustability.

Agradecimentos

É com grande satisfação e alívio que dou como terminada esta jornada que culmina na conclusão desta tese. Gostaria de aproveitar este momento para agradecer a todos os que contribuíram de alguma forma para a realização deste trabalho. Foram vários momentos de muito esforço, dedica ção e aprendizagem, e nada disso teria sido possível sem o apoio de determinadas pessoas. Assim, neste espaço, quero expressar a minha gratidão a todos que fizeram parte desta trajetória.

Agradeço, em primeiro lugar, à minha família, que sempre esteve ao meu lado e me apoiou em todas as etapas. Sem o incentivo e o suporte incondicional dos meus pais, avó e tia, não teria chegado até aqui.

Agradeço também aos meus amigos, que foram verdadeiros parceiros durante essa jornada. As suas palavras de encorajamento, conselhos e amizade foram fundamentais para que não desistisse em momentos difíceis.

Não posso deixar de mencionar os meus orientadores, António Cruz e Sergio Lopes, que me guiaram durante todo o processo de pesquisa e escrita desta tese. Os seus concelhos, sugestões e críticas foram fundamentais para o desenvolvimento do meu trabalho.

Por fim, agradeço à minha namorada, Cláudia Ribeiro, que sempre esteve presente, apoiou-me e incentivou-me a não desistir em nenhum momento. A sua compreensão, amor e paciência foram fundamentais para que eu me pudesse dedicar ao máximo a este projeto.

A todos vocês, muito obrigado!

Pedro Xavier Mendes Araújo

“Technology is best when it brings people together.“

Matt Mullenweg

Contents

List of Figures

List of Tables

Acronyms

Amazon S3	Amazon Simple Storage Service
API	Application Programming Interface
APP	Application
AWS	Amazon Web Services
CLI	Command-Line Interface
CORS	Cross-Origin Resource Sharing
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
DApp	Decentralized Application
DSR	Design Science Research
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JAR	Java ARchive
JRE	Java Runtime Environment
JS	JavaScript
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
JWT	JSON Web Tokens
OOP	Object-Oriented Programming
ORM	Object Relational Mapping
P2P	Peer To Peer
POC	Proof Of Concept
PWA	Progressive Web App
REST	Representational State Transfer
RxJS	Reactive Extensions Library for JavaScript
SaaS	Software as a Service
UI	User Interface
UX	User Experience
VM	Virtual Machine
WORA	Write Once Run Anywhere
WWW	*World Wide Web*

Chapter 1

Introduction

This work aims to analyze, design, and develop a system that can effectively trace all the components of a ship during its manufacturing process (including related documents), using blockchain technology. By creating a secure and reliable history of the ship's data, the system can add value to the ship. The West Sea Viana Shipyard will serve as a case study for the implementation of this system.

The first chapter of this document presents the motivation for carrying out this project. The problem statement is then discussed, followed by the project's goals and contributions. Finally, the chapter concludes by outlining the structure of the entire document.

1.1 Motivation

Traceability in manufacturing, for any type of industry, enables the tracking of products' components throughout the production chain, from raw materials to the final product, passing through each intermediate component and process stage, on every industrial player in the chain [2]. The information generated throughout the manufacturing process, including certificates and inspection results, is registered and therefore traceable. In [8], Beyhl et al., present an overhaul idea of what the traceability enables for any kind of industry is given:

> "...traceability enables a semi-automatically documentation of the innovators' journey, enables the generation of documentation, including cross-references, and therefore makes the final handover more complete..."

In the shipbuilding industry, the process of building a ship is complex, as it is described in [9]. This process has several stages, which are described in [10], being important to guarantee high-quality levels during the manufacturing process. For this, the methods and technologies adopted for process traceability must be selected, having in mind the specific requirements and constraints of the shipbuilding industry in the most efficient way. In addition, process traceability may add value to the ship, as all the traceable information may be easily transferred to the final client. This way, the customer can have the information of any material, component, part, and block of the ship, easily and reliably, as well as all the certificates and documents necessary for any process, following the standards of the classifier society [22].

In order to fulfill the need of a traceability system that targets the shipbuilding industry, this work presents a case study with a company that is an international reference in the naval sector, the West Sea Viana Shipyard. The system is implemented along with two interfaces that work as proofs of concept to show how the system can be used.

The final goal is to implement a system that will help the users to easily see the full traceability of any element in a safe and structured way with the usage of blockchain to ensure the trustability of the data.

1.2 Problem Statement

The traceability of the parts and components used in the manufacturing process of a given product is a relevant topic that has been fostering value increase in the value chain, as a result of the increasing availability of technology enablers (e.g., Internet of Things (IoT) or Distributed Ledger Technologies (DLTs)). Traceability technologies and methods enable the user to know the origin of a particular element, whether it is a raw material or a component made of other previously manufactured elements. This chain of traceable information can be persisted by several technologies, one of which is the blockchain. One of the most obvious advantages of traceability is the ability to quickly know where a part is, where it came from and where it went during its transport. Another advantage is, in the case of the automotive or heavy metalworking industries, the ability to track the parts of a batch when a defect is detected, thus enabling its quick replacement to achieve high-quality standards and meet the needs of buyers.

The traceability of the elements or components used for the manufacturing of a given product is something important that is gaining more value today, mainly due to technological advances that the paradigms Industry 4.0 and Industry 5.0 have been continuously pushing.

By applying traceability techniques and methods to the shipbuilding industry, in addition to the ability to track parts of the ship during its manufacturing, we can also know other relevant types of information, such as the percentage of recycled material on the entire ship, the provenance of all its parts, their location if repairs are required, the quantity used of a given component, and also the possibility of associating to each component its invoice/receipt and quality certificates, following the standards of the classifier society [22], both of the supplier and of each process of the production chain. The use of a blockchain-based system for traceability, ensures the registering of information in a decentralized, transparent and trustable way, avoiding counterfeits and untracked changes. This transparency of the data registry also ensures confidence in the data by the customer, adding value to the components/parts/raw materials that are used in building a ship. It should also be considered that a historic with manufacturing data will be created, leading to faster detection and correction of possible failures. Besides that, for the customer to easily consult the information, a Web App/Mobile App can be created for accessing this blockchain-based traceability information and reveal it to the user in a friendly and interactive way. Taking this into account, the sale of a ship would have added value, because the customer is not limited to buying a ship, but also a chain of information of all its components.

1.3 Goals / Contributions

Taking into account the importance of traceability to any type of industry, and in the particular case of the shipbuilding industry, the objectives for this project are:

- The study of the traceability in general and in the particular case of the shipbuilding industry and how the blockchain can add value to this sector;
- The design of a model that can support all relevant data and properties to maintain on a blockchain (and what to maintain off chain) to deliver to the client alongside the final product, in this case, the ship itself. This information will refer to all raw materials, materials, components, parts, blocks, and all documents and certificates, following the standards of the classifier society [22], thus creating a chain of reliable and immutable information available to the company and to the customer;
- The development of a blockchain-based backend registry platform for components and provision of traceability information applied to the manufacturing process of a block of a ship, knowing that this process can be replicated for any other block.

Each stage of the production chain of a block of a ship will be traced from its beginning to its end. On the blockchain, there will be associations with references to documents and certificates of each component from any origin (supplier countries), whether these are certificates of quality, warranty, origin, or even navigability. This way, all certificates can be accessed easily and from anywhere, recursively from an item to be tracked, to all its components.

Several things are aimed to be achieved and developed during this project in order to achieve the minimum valuable product:

- Scientific Paper, in order to build a product that makes sense and that adds value to the shipbuilding businesses, a lot of research was made about how these kinds of products are made and what already exists. All this knowledge allowed us to design an innovative product, and with that it was also possible to write and publish a scientific paper that improves and helps the community to grow. This paper was published on an international conference with the title *IoT and Blockchain Technologies for Process Traceability in the Shipbuilding Industry*, [3].
- Architecture design, one of the first steps to build a viable product is to design a solid architecture that uses the right tools for the job and all the necessary communications. This allowed the development to go a lot more smoothly without the need to wonder about what to do next or how to do it.

- Smart Contract with API, being a blockchain based product, this had to be the first part of the actual development, with a strong base it is possible to build the other parts around it. From this, we get an authenticated API that can only communicate with the smart contract. This smart contract is responsible for the main storage and traceability operations.
- Cloud solution to store files, this kind of storing is used nowadays in order to keep the hosting platform less heavy, and it provides a much safer platform and scalable platform, it is a good solution, it can evolve quite easily, and the files will always be available.
- Backend with basic authentication, the interface for all communication, it is the only possible way to interact with the system, and it is the place where all the business logic happens. It as an autogenerated documentation, testing platform, and it has JWT based authentication to ensure the safety of the system.
- Web application, the main way for the clients/users to interact with the system. It is a POC that show one possible implementation of a web app with only the required features for a user/client to work with.
- Mobile application, the main way for the employees to use the system. It is also a POC that shows how the employees could use the system in a day-to-day basis.
- Experimental Validation, it shows how the application performed with real data from The West Sea Viana Shipyard and how they evaluated the system.

1.4 Research Method - Design Science Research (DSR)

Design Science Research (DSR) is a research approach that focuses on the creation and evaluation of designs, artifacts, and systems that address practical problems and meet specified requirements. The goal of DSR is to generate knowledge and understanding about how to design and create effective and useful artifacts, systems, and solutions that can be applied in real-world settings.

DSR is often used in fields such as computer science, engineering, and management, where the focus is on developing practical solutions to complex problems. DSR projects typically involve the development and testing of prototypes, and the evaluation of their effectiveness in addressing the problem at hand.

DSR differs from other research approaches, such as basic research or applied research, in that it is focused on the design and development of practical solutions, rather than on the discovery of new knowledge or the application of existing knowledge to solve problems. This focus on design and development makes DSR a unique and valuable approach to research, as it allows researchers to create solutions that are tailored to specific needs and requirements.

The steps used to implement DSR are:

1. Identify a practical problem that needs to be addressed: DSR is focused on addressing real-world problems, so it's important to start by identifying a problem that you want to solve.

2. Define the requirements and constraints of the problem: Once you have identified a problem, you should define the requirements and constraints of the problem, including any technical, economic, or social constraints that need to be considered.

3. Develop a design or solution: Based on the requirements and constraints of the problem, develop a design or solution that addresses the problem. This can involve creating a prototype or prototype of the solution and testing it to see how it performs.

4. Evaluate the design or solution: Once you have developed a design or solution, you should evaluate its effectiveness in addressing the problem. This can involve collecting data, running experiments, or conducting surveys to assess the performance of the design or solution.

5. Disseminate the results: Share your findings with others through publication in academic journals or conferences, or through other means such as blogs or online platforms.

By implementing DSR, we need all this steps instantiated for this context, so we have the following contextualized steps:

1. Identify a practical problem that needs to be addressed: The problem is described in Section 1.2.

2. Define the requirements and constraints of the problem: All the requirements and constraints are presented in Section 3.1.

3. Develop a design or solution: The developed solution is presented in total in Chapter 3.

4. Evaluate the design or solution: The validation is made on Chapter 4.

5. Disseminate the results: The results presented in Chapter 5.

It's important to note that DSR is an iterative process, and you may need to go back and revise your design or solution based on the results of your evaluations. Additionally, DSR projects often involve collaboration with other researchers and practitioners, so it may be helpful to work with a team or seek feedback from others as you progress through the research process.

1.5 Document structure

This document is structured in 5 main chapters.

The Chapter 1 is the Introduction, here is presented the motivation, the problem statement, the goals, and contributions that are planned to add to the project and the complete document structure.

The Chapter 2 is the Background and consists on the presentation of the essential concepts for this project, here it is presented the relevant concepts with some theoretical introduction and some related works (Literature Review).

The Chapter 3 describes the Process traceability in the shipbuilding industry. It presents the proposed architecture for the system, system modeling and the actual development of the system with the used tools and complete description of the implementation.

The Chapter 4 is the Validation, and it presents the experimental validation, the tests that were used and some analysis to the results;

This thesis closes with the Chapter 5 that is the Conclusions and Future Work, here it is presented the conclusions regarding the obtained results and some possibilities about the future work.

Chapter 2

Background

In this chapter, it is presented the several concepts, terminologies, technologies, programming languages and paradigms that are considered relevant for the context of this work alongside their basic explanation, or with a more complex development in some particular cases. Besides this, it is also presented the literature review that analyses several systems that present solutions for traceability with blockchain centered applications and that target several industries, most of them are a part of Industry 5.0 or Industry 4.0.

This concepts, technologies, and processes are presented in different sections. There is a section for relevant concepts about traceability, another for environment and deployment concepts and technologies, and a final section about technologies and concepts used for development. In some cases, there are some processes shown in order to contextualize and to allow a better understanding about the concepts themselves and how they interact and work together.

To close this chapter, the Literature Review is presented, this refers to several articles that present solutions for traceability in several industries with blockchain based systems, those are used mainly to understand how these systems are thought, how their architecture is done and, mainly, what are the common issues that this systems face and how to solve them. It also allows us to understand how innovative this project is in this particular industry and how it can improve it.

2.1 Relevant Concepts

In this section, several relevant topics will be addressed related to the investigation and theoretical part of this project, as well as concepts deeply connected with the development and practical part of the project.

It starts with the explanation about the traceability applied to the shipbuilding industry with focus on the manufacturing process of a ship, how the traceability is applied and how they manage documents, these topics are specific for the case study company, The West Sea Viana Shipyard [20].

Then, some programming concepts are addressed, this concepts refer to what is going to be implemented, how it will be done, the conventions that are followed and terms that will be used during the next chapters to explain how some specific problems were solved during the development phase of the project.

After that, the programming languages and technologies used for the development phase are addressed, in here, some concepts are just briefly or theoretical explained because they will be addressed again in the next chapters to describe how the implementation was done. With that in mind, some technologies are introduced and contextualized.

This section finishes with the concepts about environment and deployment, again, in here these technologies are addressed only theoretically because they will be addressed again in the next chapters, so some of them will be only contextualized.

2.1.1 Traceability in the Shipbuilding Industry

Taking into account the importance of traceability in any type of industry, the objectives defined for this work have been, 1) the study of traceability in the shipbuilding industry; 2) the achievement and design of a model that defines all the characteristics and information that are relevant to maintain on a blockchain for delivery to the final customer together with the ship. This information must refer to all the raw materials, materials, components, parts, blocks, and all documents and certificates, following the standards of the classifier company [22] corresponding, thus creating a reliable and immutable chain of information that shall be available to the company and the customer. This system will be developed for the manufacturing process of a ship block, knowing that this process can be replicated to any other block of any ship.

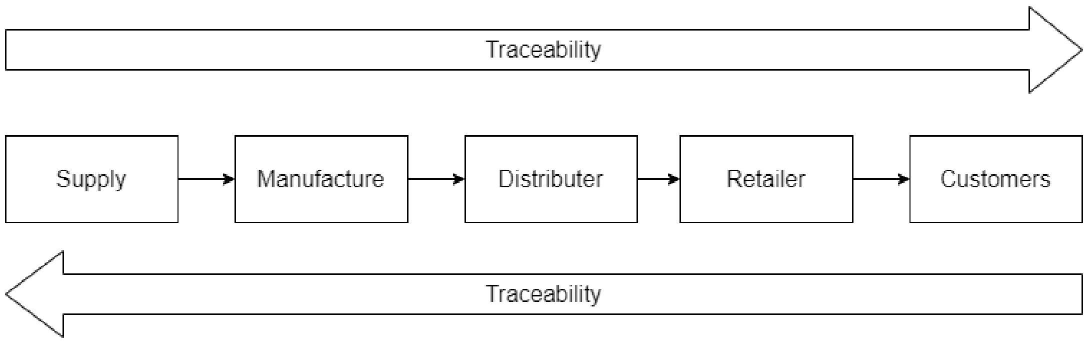


Figure 2.1: Standard blockchain traceability model as defined in [4]

Figure 2.1 represents the standard activities model used to describe traceability in a supply chain, representing that each stage of this production chain will be traced from its beginning to its end. This model can be applied to any type of industry, and in the context of this paper, it will be applied to the shipbuilding industry, more specifically in the production process of a ship block. Figure 2.2 represents an activities flow model of that process.

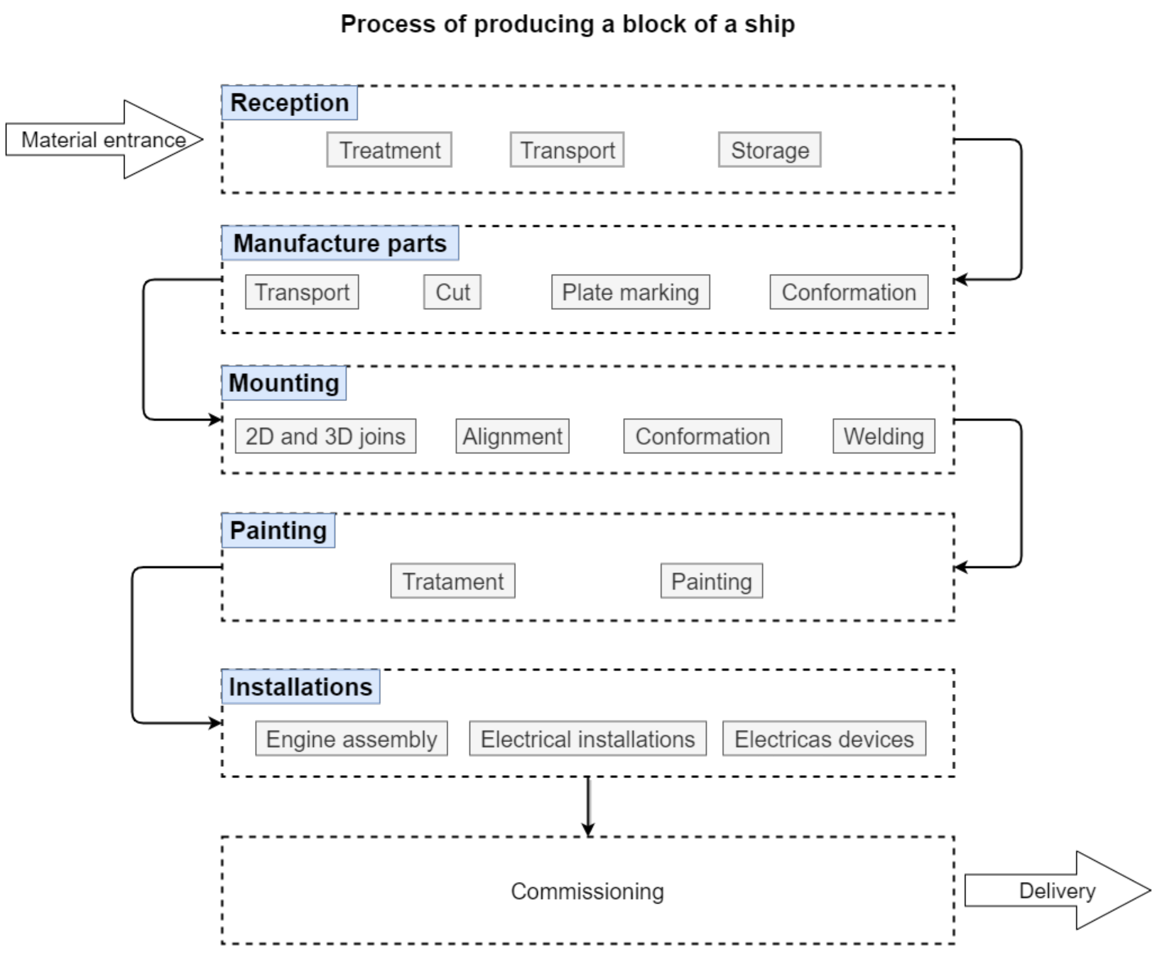


Figure 2.2: Ship block manufacturing process

As stated earlier, references to documents, e.g., certificates, warranties, for any part and any source (supplier countries) will also be associated with the blockchain information about each part's traceability, so that all certificates can be accessed easily and from anywhere.

For the practical case of this paper, a case study has been established with the West Sea Viana Shipyard[1] [20] , which is an international reference shipyard, located in the north of Portugal, specialized in ship construction, repair, and conversion.

2.1.1.1 Shipbuilding Manufacturing Process

The process of building a ship has many complex steps. A ship is made of several blocks, as shown in Fig. 2.3. These blocks are previously assembled separately and, when all the blocks are ready, the ship assembly process begins. This process, which is described in detail in [9], consists of the assembly of all the blocks that give rise to the complete ship.

[1]West Sea Viana Shipyard [20], Portugal, `https://west-sea.pt`

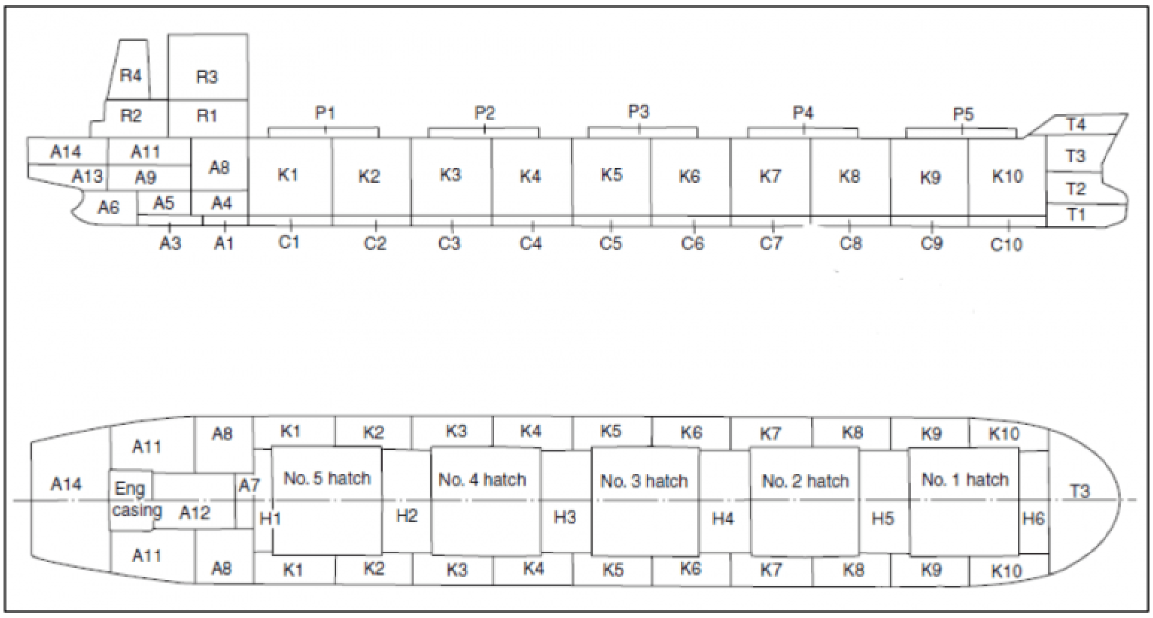


Figure 2.3: Division of a ship into blocks. Image taken from [9]

The purpose of the case study is to put into practice the model and system presented, this will be applied to the production process of a single type of block of a ship, but it can be applied to any type of block. For this, it was drawn a sketch (cf. Fig. 2.2) of what is the process of producing a block of a ship. This is just a sketch based on the process demonstrated in [10], and it is not necessarily the process used by West Sea Viana.

With this model in mind, applied to the specific context of West Sea, the proposed solution will be able to track any raw material, material, element, part, and block of a ship from its reception, manufacture, or production to its assembly or delivery through any stage of the production process. For this, all relevant traceability information will be stored in real-time on a blockchain that can be consulted by the company and the customer. And, they also have access to documents and certificates obtained or generated during all stages of the production process, which are saved on an off-chain database.

2.1.1.2 Manufacturing Process Traceability

To understand more about the West Sea Viana shipbuilding process and the problems they need to solve, a quiz with 20 questions that relate to functional traceability needs and requirements of the manufacturing process has been made and presented to the company. This subsection addresses the Traceability of the production process.

Regarding the traceability of The West Sea's production process, it is currently done with their internal database using two identifiers on each component. The components can be tracked when needed, and all the smaller parts of each component can be traced back to their original component (parent component). Also, the company follows the ship recycling process, which means they keep track of all the information and inventory (assets and equipment) of every material, including materials harmful to the environment. The company also wants to keep all the information about their production process, in case of any reformulation, and the information and details about the inspection process are also properly referenced and kept. All materials have to be provided with an identification label and their identification is carried out in that way. The identification of the origin and supplier of each material and the association of each sub-part with its parent component is assured by the suppliers and the company only needs the final certificates of conformity, quality, and approval. As the last thing, the company does not need to know which operator worked on a specific component, they only need to know the company that they belong to.

2.1.1.3 Document Management during Manufacturing

Regarding the issuance of certificates, they can be requested at any time before the design phase is over. The material certificates are issued through tests carried out by certified inspectors, and origin certificates are issued by the suppliers and delivered in the final phase. There is no need for these certificates to be in paper format, so they are sent by email in a PDF format, and they are stored on a centralized database within the company network. Also, these documents are not associated with the component's invoice, but they are with the purchase order. These certificates are made available to the customer from an internal system, as a small part of the final document delivery process, which includes operation and maintenance manuals and project schematics. We also understood that the navigability certificate is issued by an authorized entity and that only the Technical and Test Coordination Department has access to that certification information.

2.1.2 Programming Concepts

In this section, some concepts about programming are addressed, this includes basic concepts about different parts of the application (Frontend and Backend 2.1.2.4), different types of applications (Web 2.1.2.6 and mobile 2.1.2.7), patterns and ways of developing an application (API 2.1.2.5), security in the applications (Authentication and Authorization 2.1.2.1 and JWT 2.1.2.2) and other concepts used during the development (Recursion 2.1.2.3).

2.1.2.1 Authentication and Authorization

In the tech world, more specially in the programming world there are two very important concepts when we try to make any type of application secure, those two concepts need to be distinguished, and they are Authentication and Authorization. The differences and different ways of how to use them and implement them are well specified by the authors on [16] and we can also see a visual representation on the Fig. 2.4.

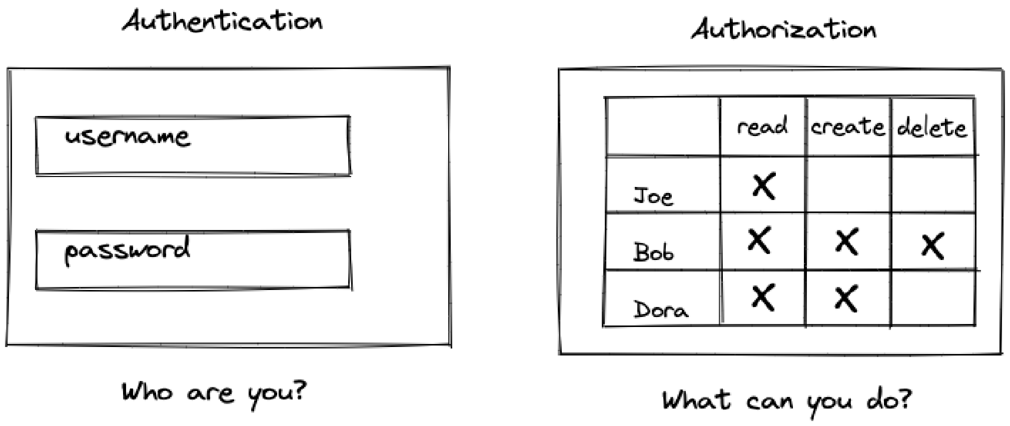


Figure 2.4: Representation of Authentication vs Authorization

Authorization, is the process performed by an application that determines if the user/client has permission or access to the particular request that is making. There can be several types of authorization, or even none if a particular request is open to the public.

Authentication, can be used by the server and by the client. The server uses authentication in order to know exactly who is trying to access or making a request, but the client uses it to know that the server is real (confirm the veracity of whom the server claims to be or do).

2.1.2.2 JWT

JWT is an industry standard RCT 7519 method for authentication between two parties, it uses a signed token in base64 format that authenticates any web request, it contains JSON objects, that commonly represent identifiers and claims, that allow the authentication to be made. This specific type of token is composed by three essential parts, the Header, that contains the type of algorithm used for the encryption and that also identifies the token as a JWT; the Payload that contains specific fields of information or claims that will be used for the authentication; and the Signature ensures that the token hasn't been altered in any way, it is unique for each JWT. More specific information about JWT and how to use it can be found on [1].

2.1.2.3 Recursion

In the context of programming, Recursion is a really important concept/technique that simply represents an already active subprogram that is being invoked again, either directly by itself or by any other subprogram. If we consider, a recursive function is a function that simply calls itself. It is helpful in several situations, specially to make the code more readable and simple, but it can make usage of too many errors and cause stack overflow errors. This concept can get a lot more complex with some specific patterns [5].

2.1.2.4 Frontend and Backend

Frontend and Backend are important concepts in programming (Figure 2.5), specially when developing for the web. The frontend is also referred to as the 'client side' of the application, it includes everything that the user can directly interact with, we can also, arguably, say that the UI/UX are a part of the client side including responsiveness (in case of the web). The backend represents the 'server side' of the application, and it is the part that the users cannot directly interact with, their responsibilities include the handling and storing of data as well as all the business logic.

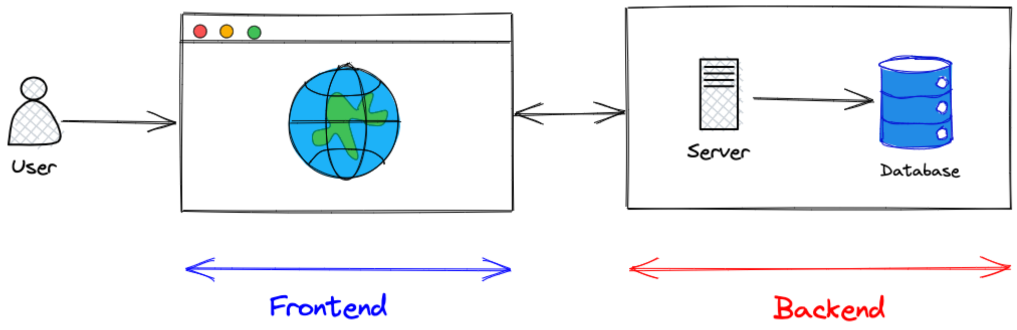


Figure 2.5: Frontend vs Backend representation

Both the client side and the server side need to be able to communicate with each other in order to build a fully operational application, but it does not mean one can work without the other, this is absolutely true for the backend because it does not depend on the frontend, in the other side, most times the frontend depends on the backend, so it cannot work without it.

2.1.2.5 API

APIs are mechanisms that allow several software components/services/applications to communicate with each other without the need to know how they are implemented, we can even say that is works as a middleware/intermediary between applications. They are useful for every part involved in the business of the software because it simplifies how applications integrate in existing (or new) architectures.

A REST API defines a set of methods like GET, POST, PUT, DELETE, between others, that the client can use to access the server. It doesn't necessarily have a graphical interface, but it can. More specific information about REST can be found on [24] and a graphical representation can be found on Fig. 2.6.

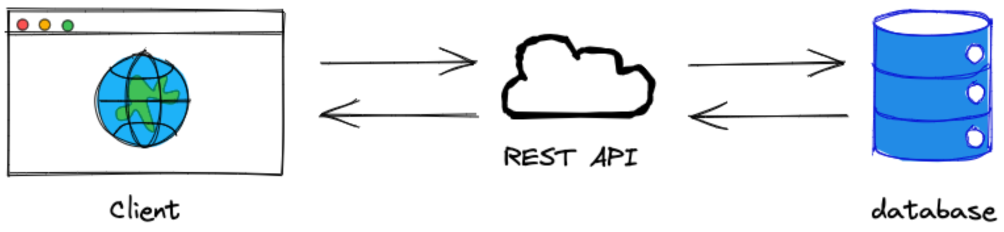


Figure 2.6: REST API

2.1.2.6 Web APP

A web application is a piece of software accessible only by using a web browser, and it is composed by a frontend and a backend 2.1.2.4. Different from a website, a web application is interactive, and it lets the user manipulate content, a website is usually just static data and there are no interactions.

2.1.2.7 Mobile APP

A mobile application is a piece of software developed specifically to be used on a mobile device, such as a smartphone or tablet. There are three main types of mobile applications: native apps, hybrid apps and progressive web apps. In a very basic way, a native app is made to run on a specific platform or device (Android or IoS); a hybrid app has different aspects of web apps and mobile apps, but they can be thought as web apps in a native app container, they work as both and they are usually built for the web, IoS and Android at the same time; a PWA is a mobile app that only needs the browser to run, they usually do not need an app store to be downloaded, a browser will work.

2.1.3 Languages and Technologies

In this section, some technologies used in the project are addressed, most of them are only introduced or contextualized because they will be address again later during the development explanation chapters, this includes programming languages (Java 2.1.3.5 and GO 2.1.3.7), blockchain and web3 technologies (Blockchain 2.1.3.1, Smart Contract 2.1.3.2, Decentralized Applications 2.1.3.3 and Hyperledger Fabric 2.1.3.4), backend technologies (Spring and Spring Boot 2.1.3.6) and frontend technologies (HTML5, CSS3, JavaScript 2.1.3.8, Angular and RxJs 2.1.3.9).

2.1.3.1 Blockchain

In a simple way, the blockchain is a system that stores data in a way that makes it almost impossible to change, making it also impossible to hack the system. It is an immutable ledger (or a distributed database) that is shared among the nodes of a computer network, and so it brings the assurance of fidelity and security of a record and generates trust without the need for a trusted third party

(Fig. 2.7). The blockchain systems are better known for their current role on cryptocurrency, that is to maintain a secure and decentralized record of transactions. A key difference from a normal database is that the blockchain stores data in blocks that are linked in the form of a chain, each block stores a set of transactions which their hashes are stored as the header, this allows a quicker and easier verification of individual transactions in a block (Fig. 2.8).

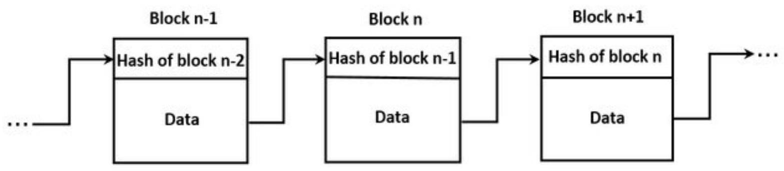


Figure 2.7: Blockchain structure (Image taken from [19])

Applying the blockchain to concepts like traceability, supply chains can be improved by enabling faster and more monetarily efficient delivery of products by also improving the coordination between involved parties, and helping access to financing. Besides this, it provides transparency related to the product's information.

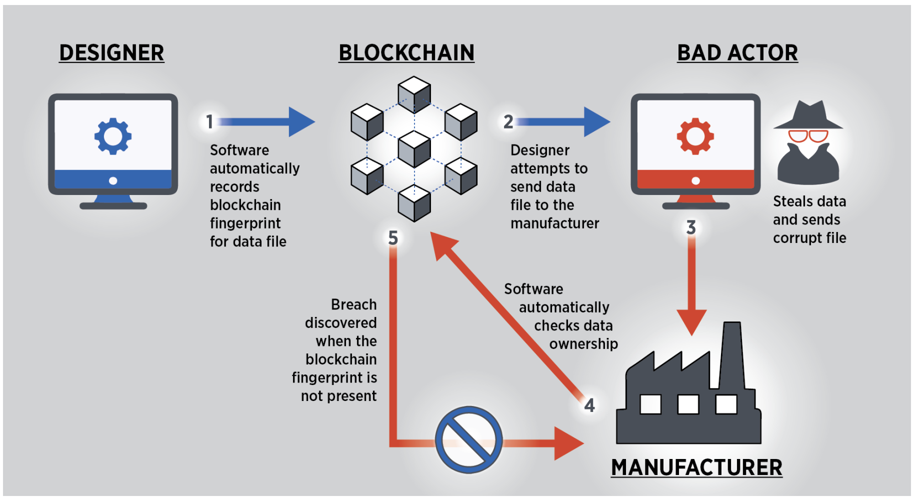

DESIGNER
BLOCKCHAIN
BAD ACTOR
1
Software automatically records blockchain fingerprint for data file
2
Designer attempts to send data file to the manufacturer
Steals data and sends corrupt file
3
5
Breach discovered when the blockchain fingerprint is not present
Software automatically checks data ownership
4
MANUFACTURER

Figure 2.8: Blockchain security (Image taken from [11])

2.1.3.2 Smart Contract

Smart contracts are programs stored on a blockchain network that run when their conditions are met, they are used to automate the execution of agreements, concede to all parties the certainty of the agreed outcome (less time lost and without intermediary parties) and they are also used to automate a workflow, a graphical example of this interaction is shown on Fig. 2.9. When the subject is cryptocurrency, it allows a user to lock up their funds and release them when the smart contract conditions are met, so when implementing banking products, a central authority is not needed anymore. With the usage of a smart contract, there is no need for a central server or agent to trust, it defines the conditions that all parties agree. All involved parties run the smart contract and always get the same result, this way all parties can be sure of the authenticity of the outcome (outcome is always correct). It also opens a lot of possibilities for decentralized apps and removes the need for expensive third parties (intermediaries). So when using a smart contract, we can assume that a transaction made through it is traceable, transparent, and irreversible.

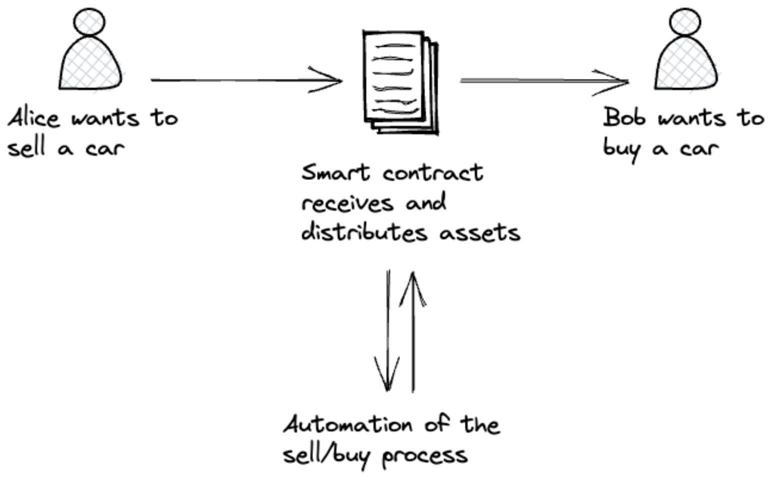

Alice wants to sell a car
Smart contract receives and distributes assets
Bob wants to buy a car
Automation of the sell/buy process

Figure 2.9: Example of usage of a smart contract

2.1.3.3 Decentralized Applications

Decentralized Applications are applications/programs that run on a blockchain or P2P (peer-to-peer) network of computers, instead of relying on a single computer, and they are free of the control, interference, and competence of a single authority, a graphical comparison between normal Apps and dApps can be found on Fig. 2.10. These applications provide safeguarding of privacy and flexibility, but sometimes it is impossible to scale and, because they are the future, normal FE frameworks are not adapted to dApp user interfaces (problems with identity and usage of the blockchain world state, most cases need a BE/API to take care of all interactions). Some common platforms used for the creation of dApps are: Ethereum or Hyperledger Fabric.

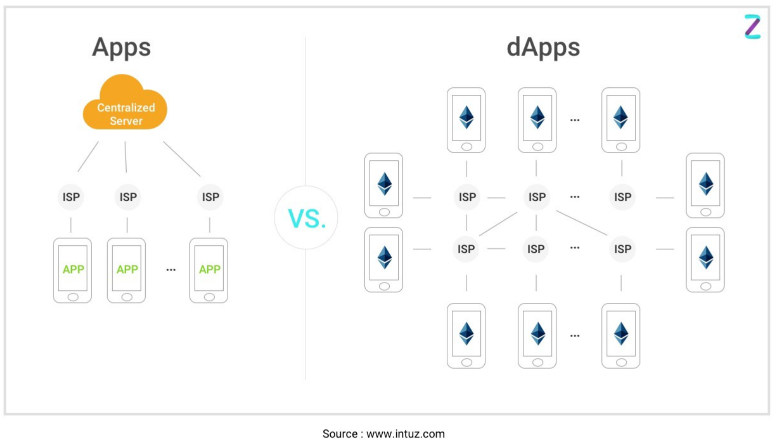


Figure 2.10: Structure of decentralized applications (Image taken from [18])

2.1.3.4 Hyperledger Fabric

Hyperledger Fabric is an open source modular and distributed blockchain framework that acts as a foundation for developing blockchain-based solutions/products, and applications. It allows the plug-and-play of components and modules with a unique way to solve consensus within a network without the need to sacrifice performance or privacy, this versatility eases the development and puts this framework in a range of multiple use cases for multiple industries.

2.1.3.5 Java

Java is an object-oriented programming language that compiles to platform independent byte code, this means it compiles directly to byte code that can run in any machine without recompiling by executing the code through the JVM, for this, a machine just needs to have the JRE installed, like most machines nowadays. This language follows the slogan of WORA, “Write Once, Run Anywhere”.

2.1.3.6 Spring & Spring Boot

The Spring framework provides a complex but comprehensive infrastructure to develop applications in Java, and it has a lot of moderns to ease the development, decrease boilerplate code and coding time like support for databases, security, tests, the following of patterns like MVC and dependency injection.

Spring Boot is a framework built on the top of the Spring Framework that provides a faster and easier way of setting up, configuring and run Spring applications. It helps to eliminate even more boilerplate configurations/code, it provides more initial dependencies to simplify the development of an application, it avoids a lot of the complexity of the application deployment by providing an embedded server, and it simplifies the integration of other Java dependencies/frameworks like ORM’s or structure converters.

2.1.3.7 Go

Go is a compiled and statically typed programming language that focus on simplicity and efficiency, it is a modern choice for high performance server side applications because it compiles extremely fast.

2.1.3.8 HTML5, CSS3, and JS

HTML, CSS, and JS are the languages of the web, they are used together to build attractive and interactive websites, and they are the base for most frontend frameworks.

HTML is a markup language that provides the basic structure of websites. CSS is a styling language that is used to create layouts, format the structure and create attractive presentations. JS is a programming language used to control the behavior of different elements on the website and change its structure in order to make the website interactive.

2.1.3.9 Angular & RxJS

Angular is a frontend framework based on Typescript, and it is used to build user interfaces for the web. It as a powerful CLI that generates most boilerplate code, and it comes preconfigured with a lot of the most common modules and configurations like routing, testing frameworks and style preprocessors. It is well known by having a long learning curve but as a result, it creates very robust frontend applications.

RxJS is described in their documentation [21] as a library for reactive programming that uses observable, which makes it easier to compose asynchronous or callback-based code, so it is used for composing asynchronous and event-based programs by using observable sequences.

2.1.4 Environment and Deployment

In this section, concepts and technologies about the cloud are addressed alongside some general concepts about the cloud environments, again, most of these concepts will be only introduced or contextualized with a basic introduction because they will be addressed again during the next chapters about the implementation. Here can be found concepts about containerization and virtualization (Docker 2.1.4.1 and Virtual Machines 2.1.4.2), cloud technologies 2.1.4.3 (AWS 2.1.4.4) and cloud storage (S3 Buckets 2.1.4.5)

2.1.4.1 Docker & Docker Compose

Docker is a technology for building, deploying, and managing containerized applications that can run on any environment. A docker container virtualizes a single kernel OS, that allows the apps that are run by it much faster, besides that, the main advantage is that an app that is virtualized through a docker container can run in any machine in the same way because the environment is always the same.

Docker compose is a tool used to manage several docker containers, it allows developers to compact all docker configurations on a single file in order to run all needed containers at the same time, the same way every time. The difference between this and Docker is shown on the Fig. 2.11.

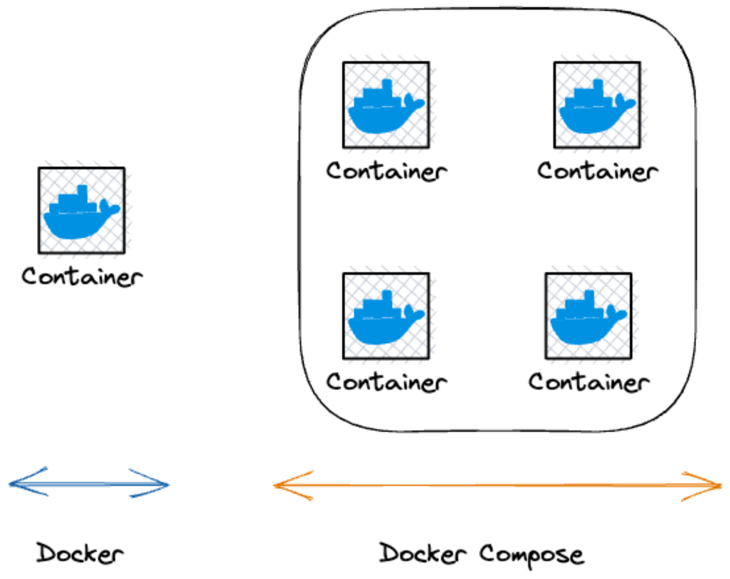


Figure 2.11: Docker and docker composer

2.1.4.2 Virtual Machine

A VM is a virtual environment that functions as a virtual computer system that works just like any other physical computer, it as a CPU, memory, storage, and it can connect to the internet even though it does not have the hardware of a physical computer. A VM is a virtual computer that runs on a physical computer, but the operating systems work separately, and it is used to run programs and deploy apps.

2.1.4.3 Cloud Computing

Cloud computing is the delivery of hosted services that are different computing services (like servers, software, databases, storage) delivered over the Internet rather than keeping them running on a personal hard drive, they can be saved in a remote server, the cloud computing architecture is represented on Fig. 2.12. There are a lot of benefits to cloud computing, being the main ones:

- No need to buy powerful and expensive hardware, cloud computing can provide powerful and faster servers;
- The services provided are extremely safe and highly configurable;
- Backups are automatically done, regular executed and distributed between several datacenters, this way the data on the cloud computing servers are not going to be lost easily;

- These services are highly and easily scalable;

An overview of cloud computing is well conducted on [17], and a good definition of cloud computing that's made there is:

"...cloud computing is a kind of computing technique where IT services are provided by massive low-cost computing units connected by IP networks."

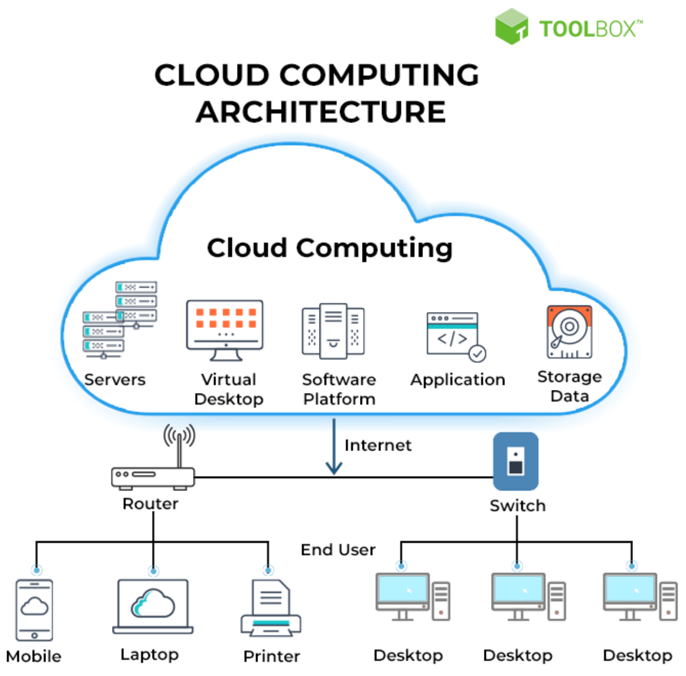


Figure 2.12: Cloud Computing architecture (Image taken from [6])

2.1.4.4 Amazon Web Services

AWS is a well known comprehensive cloud computing platform provided by Amazon, distributed on data centers from all over the world. It provides more than 200 highly configurable, scalable and cost-efficient services and solutions. It is the most used cloud platform as it praises all the cloud computing benefits in an easy and cost-efficient way has the costumer only pays for what it is used by him.

2.1.4.5 Amazon S3 and S3 Buckets

Amazon S3 is an object storage service highly scalable, available, secure and performative. It is also configurable and organizable, so it can be adapted to any use case and any user. On Amazon S3, a bucket is a container where any number of objects can be stored.

2.2 Related Works

The topic traceability in Industry 5.0 is introduced by Fraga-Lamas et al. in [14]. According to the authors, it is not possible to achieve the expected objectives in this type of systems without "a human-centered tracking" approach. So the technologies of self-identification of products for automatic recognition, positioning, and tracking, without human intervention, are essential. Although these types of technologies already exist, the authors make proposals for present and future solutions to problems in this area. Analyses of several scenarios are made, with several self-identification technologies, to find solutions to complex industrial problems. In this context, a methodology is presented to select self-identification technologies for factories in Industry 5.0, and to apply this methodology a specific use case of the shipbuilding industry was chosen. This requires the identification of the main components of a ship during its construction and repair, where passive and active UHF RFID tags were used on a patrol ship under construction, proving that it is also possible to identify, track and trace metal objects with very high-density areas. In addition, the authors conclude that passive RFID (UHF – Ultrahigh-Frequency) technology is ideal for traceability applications because they are smaller, cheaper, flexible, and can last a long time without a battery, while active RFID can be used to manage stocks because they integrate very well with other technologies, like GPS or sensors, and have a long read range. Thus concluding with the ideal technologies to solve the proposed problems and with the implementation and validation of the self-identification system in a patrol ship under construction. All research and results obtained from the article [14] can be used in the proposed system for the traceability of raw materials/components of a ship also including all associated documents/certificates (origin, quality, guarantee), following the rules of the classifier society [22]. Based on the article [14] we already know how to solve one of the possible major problems of this system, self-identification, this way, with this knowledge it will be possible to implement a traceability system that, in addition to what was documented in the article [14], it also includes the traceability of components during the construction of the blocks, and it can include the document management of certificates associated with each component, subcomponent and raw material.

In [15], Froiz-Míguez et al., focus on state monitoring tracking. This aims to monitor and track the health status of employees and events that may affect them through a decentralized real-time tracking system. The system allows the provision of information relevant to the context of use and ensures its availability. In addition, the information is stored on a blockchain, which ensures the traceability and immutability of the data so that it can be shared with other stakeholders. In this case, the information has been collected by sensors, storing it on a blockchain that acts as a reliable data source, due to its immutability characteristics, where it can be accessed from any location and be shared with any interested party, such as insurance companies or medical services. The data collection is performed by sensors and transmitted in real-time to a decentralized system. This can be an interesting way of acting, in the context of this project, if the blockchain traceability data is sold to the buyer of a ship. This would enable them to monitor the ship's construction and assembly in real-time, through the blockchain, always with the certainty that the information is transparent and reliable. This way, the customer can be always aware of the state of the ship's construction even without direct contact with the company, allowing better communication and trust between the parties involved. In addition, a "real-time" or "near real-time" system can be used for the receipt and issuance of certificates/documents that are immediately associated with raw material or part of the ship for an immediate consultation.

Wang et al. [23], address quality and safety problems in the management of materials in a construction project. They propose a traceability model for the quality of the material used, that is based on the mapping of the materials' batches. Based on the batch quality feature, this model associates the supply chain of materials using symbolic traceable objects, together three mapping types are proposed for the material batches. With all this, the authors' proposal addresses the development of a traceable system of the quality of the supply of materials until the completion of a construction project. This system is also able to track the quality of the material in the supply chain. The proposed model can select critical control points for obtaining and transmitting information, storing it in a traceability system, thus resolving problems of traceability of materials and the entire supply chain. In [23], Wang et al. address problems in a construction project, in the context of this project, much of the logic applied can also be reused in the context of the construction of a ship because the system proposed in this project will cover traceability in the process of building elements of a ship. In addition, the feasibility and validity of the proposed system were also proven through its application in the management of materials for a real construction project. The quality control obtained with this system will be something to take into consideration during the development of this system so that this control can be accessed by the company itself and by the customer, thus ensuring greater confidence.

In [7], Beliatis et al. address, in a very specific way, the choice and implementation of digital traceability technologies, aiming to improve the traceability of products in the process of production of electronic products. Taking into account the specific case of the company with which the case study has been made, this aims to technologically advance the requirements of industry 4.0, such as digital traceability and IoT technologies. The authors analyze several traceability technologies, together with the needs of the company in question and its problems with the current lack of traceability during the general production process, which includes a lack of optimization. The main objective of this work was the evaluation and comparison of the best traceability and IoT technologies to find an ideal solution for their case study, but also to fit into other types of companies with similar problems. The authors present a methodology so that their case study can implement an IoT and traceability solution that can solve their specific problems and perhaps also similar problems in other companies.

2.2.1 Discussion

In the process of research and literature review, proposals for models and systems were found that fit, in some parts, in the context of this project, in the article [14] traceability problems in the naval industry associated with self-identification systems and technologies are addressed, in this project, these types of systems will be addressed so that the most appropriate is applied in the case study, in addition to that, these data will be stored immutably and can be transmitted to the customer in real time, to add value to the final product. The system proposed in the article [15] collects data in real time and guarantees its availability to the end user, this system is used in the context of collecting health data from employees of a company, in the context of this project, the information will also be on a blockchain, but the information will only be available to the employer, it will also be available to the end customer, who can access the information of their ship at any time and, even during their construction, depending on the business rules adopted by the case study company. These topics are also differentiators in relation to the system proposed in [23], here quality and safety problems are addressed in a construction project and the proposal for their resolution is through traceability.

In [7] a specific study is carried out for their case study, aiming to obtain a set of steps to implement the most appropriate traceability system and with the best IoT technologies, in the case of the proposed system, despite being applied in a specific case study, the system proposed in this article can be applied, not only to the production process of any block and stage of a ship, but also to most production processes within the naval industry.

Chapter 3

Process traceability in the shipbuilding industry

In this chapter, it is presented the complete modeling of the system itself, the architecture that was used to develop the system and how it was implemented. It is essential to mention that this chapter is strongly connected to the chapter 2.1.1 and a lot of the information presented here is based on that.

This chapter is separated into three sections: the System Modeling 3.1 that will present the functional requirements, the different types of actors, what those actors can do in the system and how the data was modeled in the system; the Proposed Architecture 3.2 that will present the complete architecture of the system alongside a meticulous description of it; and the Implementation 3.3 that will present the implementation of the system, the tools that were used and how the system was built.

3.1 System Modeling

In this section, it is presented the functional requirements of the system and how it was modeled. Regarding the modeling, the different types of actors are also present and which actions they can perform on the system. With this information, it is possible to show some use case diagrams that will present graphically what a specific actor can do. Besides this, a domain model is also built in order to present how the data was modeled in the system.

3.1.1 Functional Requirements

With the information available on the chapter 2.1.1, several requirements for a traceability solution that also keeps track of associated documentation can be gathered:

- All the smaller parts of each component need to be tracked to their parent component and vice-versa;

- Follow the ship recycling process (keep track of every material, all the information, and all the inventory);
- Keep all the information about the production process;
- Keep all the information about the inspections;
- The documents need to be associated with a purchase order;
- Only the Technical and Test Coordination Department can access the navigability certificate information;
- The traceability should be done using their current identification methods;
- There could be users with different access restrictions;
- All the information about a component should be stored;
- Components can have several documents associated;
- A document is associated with a component through an ID reference;
- All the stored information should always be available;
- The association between a component and a document can be done separately by a different employee.

From these requirements, two main actors have been identified: ***a member of the technical staff*** and ***an admin***.

3.1.2 Use Case Model

The two main actors identified on the section 3.1.1 were:

1. *a member of the technical staff* that represents an employee, a user that uses the mobile application to scan and add components or parts. This is the user that actually create the traceability. This actor can:
   - Authenticate himself using his employee ID;
   - Scan a component;
   - Associate the component with his parent/child component;
   - Add a component manually;
   - Associate a document to a component;
   - Can verify the component he is working on;
   - Can create a request to send all the component information to be stored into the blockchain;

- Edit or delete the component he sent to be stored up to 30 minutes after he sent the request (after that time he is not allowed to change);
- See basic information and associations of a component;
- Access restricted information, depending on their role.

The use case diagram for the technical staff actor is represented in Fig. 3.1.

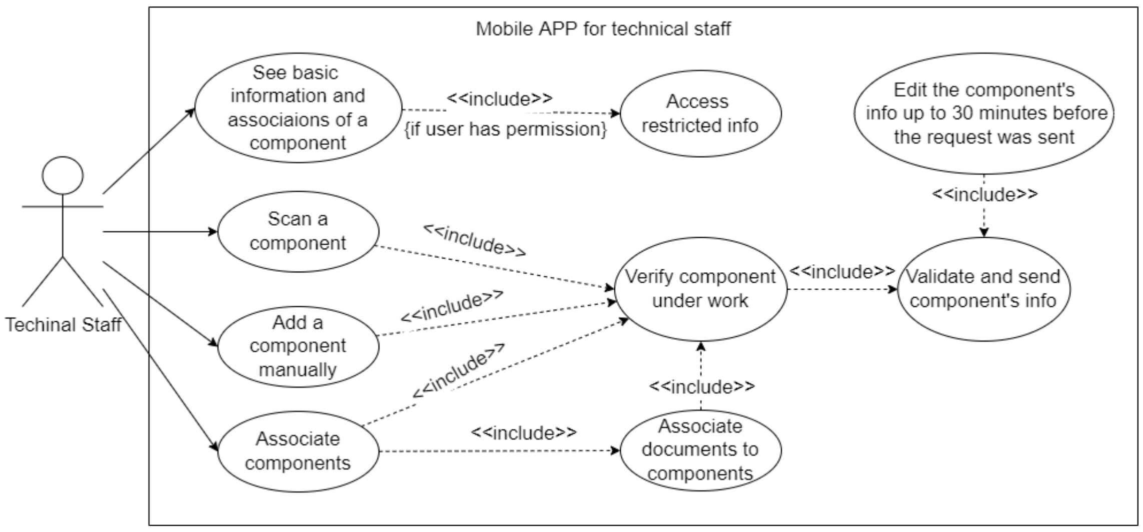

Mobile APP for technical staff
Techinal Staff
See basic information and associaions of a component
<<include>>
{if user has permission}
Access restricted info
Edit the component's info up to 30 minutes before the request was sent
<<include>>
Scan a component
<<include>>
Verify component under work
<<include>>
Validate and send component's info
Add a component manually
<<include>>
<<include>>
<<include>>
Associate components
<<include>>
Associate documents to components

Figure 3.1: Technical staff mobile app access use case diagram.

- *an admin* that is the user that uses the web application, and they can have different permissions. An admin can be allowed only to visualize one ship (in case of a client), he can be allowed to check existing users, and he can be allowed to manage different components/-parts/materials; This actor can:
  - See all blocks of a ship;
  - See all components of a block;
  - See all the associations of a specific component;
  - Access the associated documents of a component;
  - See a page with the complete ship represented in blocks;
  - See a dashboard with relevant statistics about the ship

The use case diagram for the customer authenticated user is represented in Fig. 3.2.

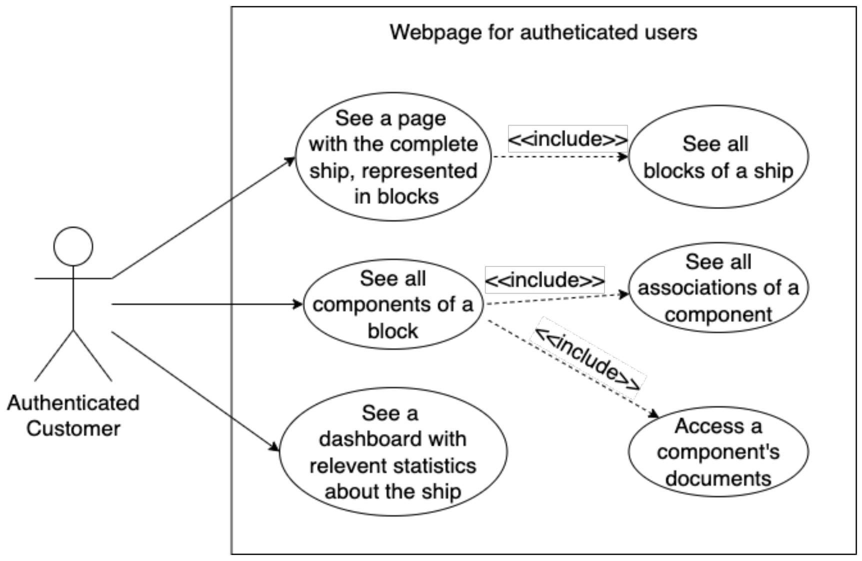

Webpage for autheticated users
See a page with the complete ship, represented in blocks
<<include>>
See all blocks of a ship
See all components of a block
<<include>>
See all associations of a component
<<include>>
Access a component's documents
See a dashboard with relevent statistics about the ship
Authenticated Customer

Figure 3.2: End user webpage access use case diagram.

3.1.3 Domain Model

In order to structure the information and data on the applications, the data model presented in Fig. 3.3 has been put forward. This model represents how the data is structured and where it will be stored in the system architecture. With this, we can distinguish three types of storage for the information:

- On-chain data: Represents the data that is going to be stored on the blockchain;
- Cloud data: Represents the data that is going to be stored on the cloud, using SaaS;
- Off-chain data: Represents the data that is going to be stored on a NoSQL database;

Following an object-oriented approach, we can identify the several models that will represent the information on the system.

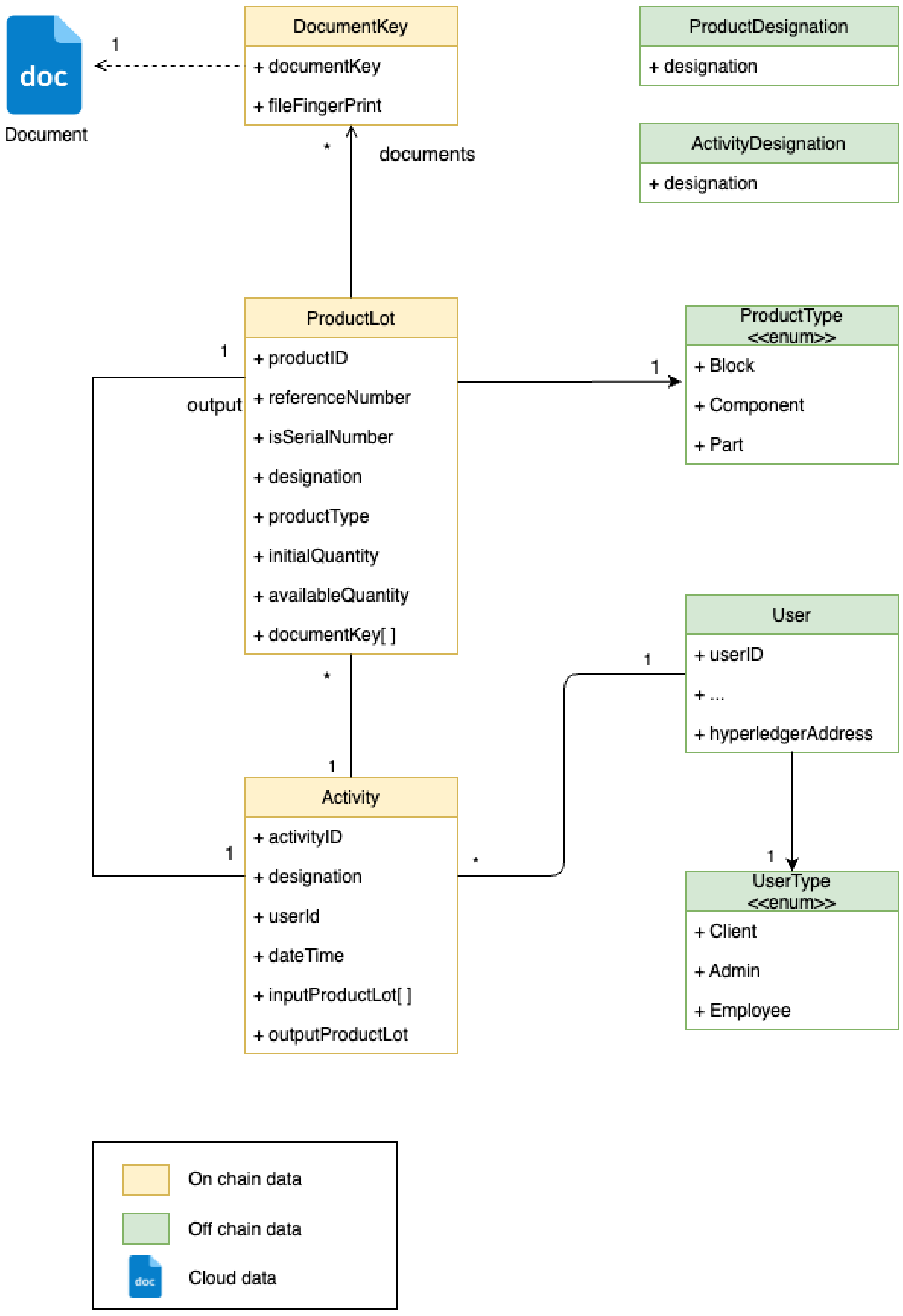


Figure 3.3: Proposed data model

Analyzing the data model, we can see that the documents are going to be stored in the Cloud using some SaaS like AWS EC2. The information about all the parts, components, and blocks of a ship along with the information on the activities that transform all of these components into other components will be stored on the blockchain because this is the information that allows the traceability to be built and so that is, essentially, the information that needs to be captured. The information about the users that can use the system will be stored on a NoSQL database off the chain, along with the extra information about the ship's components.

3.2 Proposed Architecture

As it was described earlier, two main actors have been identified for the problem at hand. For these, two applications will be created: a mobile application and a web application.

The web application can be used by every user with a valid account, this account can be created by a system administrator, and with it, the users can freely consult all the information about the ship, its components, and all the documentation related to it.

The mobile application will be used by the technical staff, and it can be used to scan RFIDs and QR codes as a measure to keep track of the information of each component and store it on the system, with the correspondent documents. This application can also accept manual codes instead of RFID tags/QR Code reading, depending on the device that is being used. This application will be used by the different company workers, and they may use the application in different ways. When the app is being used by, for example, a mechanical locksmith, he will be responsible for scanning the tag on the components that he is working on and, if a component is being fractured in several parts, he should also make the association of the smaller pieces with the parent component. On the other hand, when, for example, a mechanical engineer uses the app, he will make the association of the parts, including documents and certificates, with the corresponding parent components.

Figure 3.4 depicts the system architecture, with all adopted technologies identified. As mentioned earlier, there are two different applications for two different types of users. The web application is used by clients, and it allows them to consult the information about the ship components, blocks, parts, and materials through an interactive, responsive, fast, and productive interface. This web application can be accessed by any modern browser (Chrome, Edge, Firefox, Safari, . . .), and it will be created using a modern Front-end framework like ReactJS or Angular.

The mobile application will be used by technical users or employees of the company, and it allows the users to scan all types of components of the ship with a tag RFID or QR Code, and associate information and documents/certificates to that particular component. This mobile application will be Android native and created using Kotlin, and can be installed and used on any Android device, ideally one that can scan RFID and QR Code tags. If the device running the application is not capable of that, the system allows the manual insertion of data.

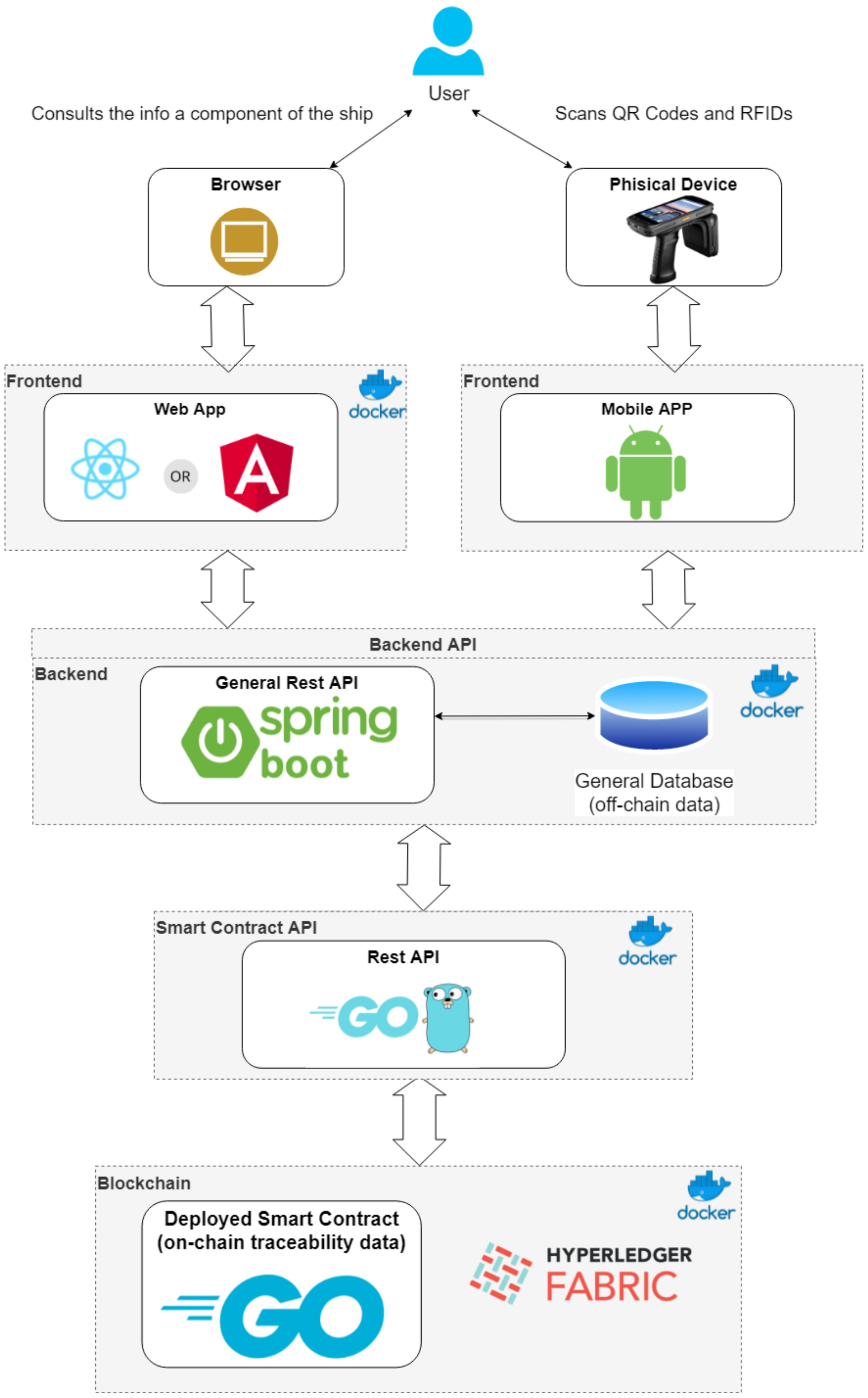


Figure 3.4: Architecture of the process traceability system

These two applications will be communicating with a backend through a REST API. This Backend, built with Spring Boot, will be responsible for all users and their interactions with the system. The Backend will make use of a database for the system users and the documents' storage. The documents may also come to be stored in the Amazon S3 cloud service. Finally, the Backend is also responsible to interact with the Smart contract API to save and retrieve traceability data to/from the Blockchain. The Smart Contract API is a REST API, created with GO Lang, and deployed on a Hyperledger Fabric Blockchain.

All these applications will be packaged into Docker containers except for the mobile application. This way, the application's source code, libraries, dependencies, and environment will be standardized into executable components that can run anywhere. Besides that, they will be secured with authentication and authorization protocols.

Regarding the Backend, the REST API will be secured using Spring Security which is a Spring framework to help with authentication, this way only authenticated users will be allowed to communicate with the Backend. The Backend will also be responsible to retrieve information from the Blockchain and the Cloud Service that stores the documents, but to get the correct documents, an association needs to exist between the blockchain data and the documents on the cloud. This association will be made upon the insertion of a document component data, the Backend will store the document on the cloud and retrieve an identifier, that identifier will be added to the component's data before it is stored on the Blockchain. To retrieve a component's document it will be the reverse, the Backend will get the data from the Blockchain, and it will use the document's identifier to retrieve the document from the cloud.

3.3 Implementation

In this section, it is presented how the different applications and parts of the system will be implemented, taking as the base the data model and the use case diagrams showed on the section 3.1 and the architecture presented in section 3.2. After this, all the tools that will be used will also be referenced from the section 2.1 with a brief explanation of how and why they will be used. With all this information, all the applications and parts of the system will be presented with a meticulous description of them and the system itself.

3.3.1 How the implementation was followed

According to all the information presented in chapter 2 & chapter 3, this complex system can be divided into different parts and applications. So, to build this system the following structure will be used:

- Backend;
  - Rest API;
  - Smart contract;

- Frontend;
  - Back office;
    - Web application;
  - Mobile application;
- Infrastructure;
  - Database;
  - Blockchain;
  - S3;

We can see that the system is divided into three main sections, that being the backend, the frontend, and the infrastructure. The backend, cf. Section 3.3.1.1, represents everything that has to do with the server side of the system, all the business logic lives in this section as well as the interaction with the database that will be used to store basic information, and the smart contract that contains the logic to interact and execute operations on the blockchain.

The frontend cf. Section 3.3.1.2, represents the applications that the users will have contact with, in this situation there are two applications, the mobile application and the back office that is a web application. This particular section of the frontend will not be done from the scratch like all the others, two pairs of finalist students from the IPVC(Instituto Politécnico de Viana do Castelo) will work on the base for each application with the supervision of the *Author* and the advisor *Prof. Doutor António Miguel Rosado da Cruz*. This way, these two base applications created by these students will only need some adaptations to the case study business logic, this will be described in the next sections.

The infrastructure cf. Section 3.3.1.3 of the system is a separate part itself, but it involves all the other parts as well. This part represents where and how the different parts of the system will run and its different environments. There in no production environment for any part of this system but, for some time, some parts will be deployed on the cloud for other people to access, this is important and essential specially because, as it was stated before, there are different people working on a base for the two frontend application (mobile and web) so they need a server they can access to get the information to use during their development.

3.3.1.1 Backend

By analyzing the architecture of the system, presented on the Fig. 3.4, we come to the conclusion that the Backend of this system will be composed by three different applications. The main REST API that will interact with an off-chain database and exchange requests with the Frontend and with the Blockchain (thought some middleware application). The Smart contract REST API that will be a secured REST API that has the only purpose of exchanging requests between the Smart Contract and the general REST API, so it can be automatically generated or created by hand. The Smart Contract that will be responsible for all interactions with the blockchain (CRUD functionalities) and for applying the necessary logic and operations to create the traceability of a part of the ship. All these applications will be containerized into docker containers in order to facilitate the execution and deployment of them unto different environments, this is important because besides the local environment, all these backend applications will have to run into virtual machines so that other stakeholder can consume them.

The main/general REST API will be the “interface“ consumed by the frontend applications, it will be the only source of any information for the outside world. It will interact with an off-chain database, that is represented on the data model presented on Fig. 3.3 with the color *green*. The tables on the data model that represent off-chain data are: User, this table stores all the basic information of a user including username, password, and the roles so that the user can log in to the applications that he has access; UserType, this table represents the roles that a use can have (EMPLOYEE, CLIENT, ADMIN), each role allows the user to access different parts of the system and execute different operations, a Client can only access the web application, and he can only see information of his ship, an Employee can access the mobile app and execute any operations there and can access the web applications, and check the designations of activities and products, while the admin can do all that and also CRUD operations on the activity/product designations and CRUD operations and change the roles of the users; ProductType, represents the type of product of the ship, it can be a Block, a component or a part, it can be augmented to fit different businesses; ProductDesignation, it represents the name/designation of a product, these designations are managed by the admins on the web applications and used by the employees on the mobile application to give a name to the product they are registering or tracing; ActivityDesignation, it represents the name/designation of an activity, and it is managed and used in the exact same way as the ProductDesignation table. Following the use case diagram of the technical staff presented on Fig. 3.1, this represents the actions that a user with the role of “Employee“ can perform on the mobile application, so the main REST API will have to implement the necessary business logic to execute them, the same applies to the use case diagram of the end user presented on Fig. 3.2, this represents the actions that a user with the role of “Client“ can perform on the web application as well as getting the traceability of any product of his ship, besides this, a user with the role of “Admin“ can access both mobile and web applications and perform actions on the web application to execute CRUD operations on the Users, activity designations and product designations as well as changing the roles of any user on the platform, also a user with the role of “Employee“ can also access the web applications and view all the users, product designations, activity designations and traceability of a ship without interacting with them.

The Smart Contract REST API will be automatically generated based on the Smart contract implementation, and it is simply a “bridge“ between the main REST API and the Smart Contract, an easy way of communication between these two applications, it is also worth mention that the main REST API is the only application with permissions to communicate with this application.

The Smart Contract is fully responsible for all the interactions with the blockchain and for building the traceability of a product. We can see on Fig. 3.3 the information that will be stored on the blockchain, that is all the sensitive information for this case study that is also the data used for building the traceability, that being the ProductLot, this entity represents a product (part) of the ship, and it can represent a unique part of the ship or a lot of parts which have the exact same specifications and which the quantity is more than zero. If we want to represent a unique product, the flag “isSerialNumber“ will be set to true, when that is the case, the “referenceNumber“ will represent the serial number of the product, otherwise, if the flag “isSerialNumber“ is set to false, the “referenceNumber“ will represent the reference number of the lot. Besides this, the “designation“ is the name that came from the “ProductDesignation“ entity, the “productType“ is the name that came from the “ProductType“, the “initialQuantity“ represents the amount of product that arrived or entered the system, if the product is unique, this amount will always be 1, otherwise it will be the inserted amount, while the “availableQuantity“ always starts with an equal value as the “initialQuantity“ and it is decreased every time that product is used to create another product. There is also the “documentKeys“, that is only filled if the product is associated with some document (certificate/invoice/receipt), they represent a list of keys, that correspond to a specific document that is stored somewhere in the cloud, and they can be used to retrieve those documents; and the Activity, that represents any registered activity that receives some product(s) of any quantity as an input and gives product, also of any quantity, as an output, it contains the “designation“ that is the name that came from the “ActivityDesignation“ entity, the “userId“ tha represents the ID of the user that performed this particular activity, the “dateTime“ that represents the date and time when this activity was performed, the “inputProductLots“ that represents the product(s) used as inputs for this activity, it can be read as a map where the key represents the ID of the product that is being referred, and the values is the amount of product used (in case of a unique product, this is always 1), an example can be found on Appendix A, Section A.1. Besides this, the “Activity“ entity also has the “outputProductLot“ property that represents the output resulting from the input product lots after the activity was performed.

This smart contract will only be consumed by the Smart Contract REST API, and it will allow only some operations to be performed, these are:

The smart contract contains all the operations shown in Table 3.1, all of them can be used internally, but they are not needed for all cases. All of them can also be used externally because, an HTTP endpoint is generated for each operation on the Smart Contract REST API, but yet again, not all of them need to be used, in a real situation, these endpoints would be disabled or converted to internal functions, but for this example we have them enabled for testing purposed thought an HTTP client (Swagger/Postman). The operations that are exclusively external are: “GetAllProductLot“, that will return all existing ProductLots; “GetAllActivities“, that will return all existing Activities; “GetTraceabilityByReferenceNum“, that expects the reference number of a product and will return the complete traceability of the referred product in JSON format, an example can be found on Appendix A, Section A.2, this includes all the information of the productLot, the activity that originated the product (it may not exist if the productLot was inserted manually without an activity origin, for example a new shipment), the productLots used as inputs of that activity and corresponding traceability of each one; “UpdateProductLotDocumentKeys“, that expects a list of document keys and updates directly on the blockchain the property “documentKeys“ of a Product-

Table 3.1: Table of Smart Contract Operations

	Is used internally	**Is used externally**
GetAllProductLot	No	Yes
GetAllActivities	No	Yes
GetTraceabilityByReferenceNum	No	Yes
ProductLotExists	Yes (CreateProductLot, ReadProductLot, ReadProductLotByReferenceNum, UpdateProductLotDocumentKeys, UpdateProductAvailableQuantity, CreateActivity)	No
CreateProductLot	Yes (CreateActivity)	Yes
ReadProductLot	Yes (CreateActivity)	Yes
ReadProductLotByReferenceNum	Yes (GetTraceabilityByReferenceNum, CreateProductLot)	No
UpdateProductLotDocumentKeys	No	Yes
UpdateProductAvailableQuantity	Yes (CreateActivity)	No
ActivityExists	Yes (CreateActivity, ReadActivity)	No
CreateActivity	No	Yes
ReadActivity	No	No

Lot by its ID, update operations are not common on on-chain data, but this specific situation is required, so the users can associate documents with a productLot; and “CreateActivity“, that expects the name of the activity performed, the input productLots with corresponding used quantities and the complete information of the output ProductLot alongside any documents to associate with it. The operations that are exclusively internal are: “ProductLotExixts“, that simply verifies if a productLot exists on the blockchain based on its ID and returns a boolean, this operation has this sole purpose, and it is used on other multiple operations like “CreateProductLot“, “ReadProductLot“, “ReadProductLotByReferenceNum“, “UpdateProductLotDocumentKeys“, “UpdateProductAvailableQuantity“ and “CreateActivity“; “ReadProductLotByReferenceNum“, that will return a ProductLot based on its reference number, this operation is used on “GetTraceabilityByReferenceNum“ for this exact purpose and on “CreateProductLot“ to verify that the reference number used to create a new ProductLot is not used by any other productLot; “UpdateProductAvailableQuantity“, that expects a new quantity and will update the property “availableQuantity“ of a ProductLot based on its ID, this operation is only performed during the “CreateActivity“ operation to reduce the quantity of the used input productLots; and “ActivityExists“, that simply verifies if an activity exists on the blockchain based on its ID and returns a boolean, this operation has this sole purpose, and it is used on other operations like “CreateActivity“ and “ReadActivity“. Some operations are used internally and externally to the Smart Contract, these are: “CreateProductLot“, that expects the all the information of a ProductLot except the ID and the property “availableQuantity“, those are defined internally and will create the new ProductLot on the blockchain; and “ReadProductLot“, that will return a ProductLot based on its ID. There is one other operation, the “ReadActivity“ that is not used at all, it was created to return an Activity based on its ID, but there was no need to use it. All these operations have the expected logic on them to not allow productLots or activities with the same reference numbers or IDs, and all the other common checks, verification, and error handling for common operations. This smart contract and smart contract API live on separate docker containers, the smart contract API consumes the smart contract and exposes one endpoint for each operation described above that can easily be consumed by any HTTP client.

3.3.1.2 Frontend

Has it has been shown on the Section 3.1.2 and based on the requirements presented on Section 3.1.1, this system will have two interfaces, each one is based on the requirements and use case diagram of each actor of the system, those are represented on graphical representation of the system, Fig. 3.4 on the Section 3.2.

It needs to be stated that these solutions were not totally made from scratch like the rest of the system, with collaboration with the IPVC(Instituto Politécnico de Viana do Castelo), two pairs of finalist students from the course of Computer Science and engineering worked on the base for each application with the supervision of the *Author* and the advisor *Prof. Doutor António Miguel Rosado da Cruz*. These students worked mainly on the interaction with the API and on several points of the corresponding Frontend/Mobile technologies in order to improve their skills, but has to be changed in order for their project to fit on the requirements of this dissertation, all those changes will be addressed on the Section 3.3.3.2. Besides this, it also needs to be addressed that the interfaces described here and shown on Section3.3.3.2 could easily be reworked to provide a better UI/UX in order to improve the overall usability and experience of the users, but in this dissertation, we focus on the actual system and backend as an all.

3.3.1.3 Infrastructure

For some parts of this system, an infrastructure was required because, has it was stated before, the web and mobile applications will not be built from scratch but from a base built by students from IPVC(Instituto Politécnico de Viana do Castelo), in order for them to work on these applications, they needed to communicate with the general REST API and to be kept up to date with any changes on any part of the backend, for that reason, the backend needs to be deployed. The ideal way is to deploy in each application on the same environment but in order to keep the project free of any charges, we had to go with free solutions, so AWS EC2 is the tool that will be used to host the general REST API but to keep it free we can't use to many resources so the Smart Contract and the Smart Contract API had to be somewhere else, namely Googles' Compute Engine because it offered a free tier. And because all the application on the backend use docker and docker-compose, it is very simple to set up everything on each virtual machine. On the general REST API, there is a Dockerfile that is used to prepare the API for release and a docker-compose that is used to define, not only the API service but two more, the MongoDB database and a MongoDB client that can be used by the students to manage the testing database this way, on the AWS machine by just running the docker-compose file all this services will be running. On the Googles' Compute Engine VM, just by using the Hyperledger Fablo described on 3.3.2, we already have a docker image that we can simply run.

Another part of the system that works on the cloud is the document storage, all the files that need to be associated with any part of the system are sent to the general REST API, the API will then create a fingerprint of that file based on SHA-256 digest calculation, generate a filename, send the file to the AWS S3 correspondent bucket, and finally it will send the filename and fingerprint to the blockchain where it will be associated with the corresponding ProductLot, has it is shown on Fig. 3.3. This will assure the authenticity of the file, making sure that the file is always the same, if there is any change it will be immediately detected, when the user tries to get the file, the fingerprint stored on the blockchain and the fingerprint of the file that was stored are always compared as a safety measure, this was any changes will be imminently seen.

3.3.2 Technologies

To build this system, a lot of technical decisions were made about the technologies to use to build the different applications, some tools inside each application, the databases to choose, and much more, in here we state the most relevant about each part of the system.

The main REST API was built with the Spring Boot framework using Java as the programming language with all the common developer dependencies of this tech stack (JPA, Hibernate, Spring Security, Lombok) and it was developing following the Spring Boot Architecture that is based on the MVC architecture. This Spring stack was chosen simply because the *Author* was proficient on it. To document the endpoints, the Open API Specification was used and implemented through a spring dependency, this generates an interface that documents each endpoint and provides an easy way to test them graphically. Regarding the database, the tables on the data model that will be stored on this off-chain database don't have any relationship between them, so it can be NoSQL, the choice was MongoDB because it is very fast to query from and very easy to work with and setup.

The smart contract was built with the Hyperledger Fabric framework, using Go as the programming language to develop the Smart Contract, Fablo [12] it was also used, this is a tool used to easily set up and generate a Hyperledger Fabric network that fully runs on docker, besides that, it also provides a Blockchain explorer (print screens can be found at Appendix C.5) and a REST API Client (Fablo REST) that generates a secured REST API interface to interact with the Smart Contract, that is the Smart Contract REST API described on the Section 3.3.1.1. For development purposes, Microfab that simply provides a single docker container image to quickly develop and test the smart contract. This all Hyperledger Fabric was described early on the Section 2.1.3.4, it is a great choice for the development of the smart contract due to the advantages described early but the main reason why it was chosen was because the development is faster and the tools like Fablo and Fablo REST provide a very easy way to interact with the smart contract and if we take into consideration that this system is already complex with a lot of moving parts, we tried to keep it simple when we could. The programming language of choice for the development of the smart contract was Go because it generates very fast and safe programs with all the modern syntax. The programming language could be JavaScript with Node or Java has Hyperledger Fabric has support for those three. The blockchain platform used for the Smart Contract could also be Ethereum, but the purpose of the Smart Contract created with it are more for completely decentralized applications and are for mass consumption, this is not the case for this system.

The web application was built using Angular and Typescript with some JavaScript libraries to perform certain tasks that are not natively supported by the Angular framework like build a graph (D3) or a PDF viewer (ng2-pdf-viewer). Some other options were considered to build this application, like Vue and React that are also JavaScript frameworks, but besides Angular only React was really considered because the development is way faster and the amount of community and packages that exist are way better, nonetheless Angular was chosen as it has native support for Typescript it forces the developer to follow their good pattern. React also supports Typescript, but it is not as good and easy to use, and it is easier to create spaghetti and confusing code because React does not follow a strict pattern like Angular.

The mobile application was built natively with Android and Kotlin, with an extra dependency in order to implement a PDF Viewer (AndroidPdfViewer). There were non-native options but, has it was explained in Section 3.2 this application will only run on Android devices, so it makes sense to make it native as native mobile apps are built for a specific operating system and are compiled using the platform's core programming language and APIs, so they are much efficient, faster and responsive. The obvious con of this choice is that if, for some reason, this application needs to run on an IOS device, the application needs to be totally built from scratch without being able to reuse almost anything beside logic. For the programming language, Java was also an option, but Kotlin is the recommended language to develop Android applications, it has a lot more modernities in terms of syntax and speed compared to Java, and they are both languages that run on the JVM, so the knowledge from Java can be transferred to Kotlin.

The infrastructure was composed with several tools, Docker and docker-compose were chosen to make it easier to run every part of the system in consistent manner in any machine and also to be easier to deploy using the docker-compose file it is just a matter of running it on a virtual machine, and we have everything setup. For the hosting, only the backend of the system was deployed but split into two different parts: the general REST API was hosted on AWS and the Smart Contract and Smart Contract API was hosted on Googles' Compute Engine, this was chosen because AWS allows a very cheap hosting of around €0.4/month for this API on a standard EC2 VM but if we included the Smart Contract and Smart Contract API the price would be much bigger as we had to increase the hardware usage on the VM, to solve this problem we made use of the 3 months free tier of Googles' Compute Engine and hosted the remaining parts of the backend there. To connect with these two different VMs it was used SSH and for transferring files it was used FTP. It was mentioned on the Section 3.2 that the documents associated with any part of the ship will be stored on the cloud, for that purpose AWS S3 2.1.4.5 was used, this tool was chosen because it was already part of an environment used on this system (AWS) and because it is easy to use and implement.

Some other tools used during the development of the system are FileZilla that is an FTP software tool that allowed the exchange of files between the AWS and the Googles' Compute Engine and the local system of the *Author*. To develop all the parts of the system, multiple IDEs and code editor were used but almost all of them are from the same provider, that is JetBrains, this was on purpose because this way, the knowledge from one IDE is transferred to another, the development is faster, and each different IDE is optimized to a specific set of Languages and Frameworks, so here follows the list of IDEs used for each part of the system:

- General REST API $\rightarrow$ IntelliJ;
- Smart Contract $\rightarrow$ GoLand;

- Mobile App $\rightarrow$ Android Studio;
- Web App $\rightarrow$ Visual Studio Code;
- Database interaction $\rightarrow$ DataGrip;

Besides the JetBrains IDEs, Visual Studio Code was the choice for the development of the web application because it provides the best environment to work with JavaScript even compare with the JetBrains solution, WebStorm.

3.3.3 Description and presentation

In this Subsection, the final version of the system will be presented, this presentation will be separated into three smaller sections in order to provide a more organized presentation. On the Section 3.3.3.1 some of the more complex parts of the general REST API and the Smart Contract will be shown in detail recurring to code, along with the Open API implementation and the off-chain database. The Section 3.3.3.2 will show the final versions of the user interfaces implemented on the web application and on the mobile application, some code used on the more complex part of these applications will also be shown and finally, the modifications made on the applications made by the students of IPVC will be shown/stated. The Section 3.3.3.3 will mainly show the docker and docker-compose configurations alongside the AWS S3 and some more information about the VMs used to deploy the Backend.

All the code used for this project can be found on the GitHub repositories presented on Appendix B.

3.3.3.1 Backend

Regarding the general REST API, the tools and architecture followed were already described on the Section 3.3.2, this API follows the Spring Boot architecture (based on MVC) that aims to separate the business layer (Services), the database access layer (Models & Repositories) and the controllers (Controllers). All the Security was built using Spring Security that allows the implementation of Authorization and Authentication, described on the Section 2.1.2.1. With this only authenticated users can access the endpoints of this application (unless it is a public endpoint like swagger and login endpoints) after they log in and get their token or refresh token. The CORS configuration was also set, which means only authorized hosts can use this API.

The authentication is based on roles, specially three roles: EMPLOYEE, CLIENT and ADMIN; these roles are based on the use cases specified before, and each role can some endpoints. With the CLIENT role, a user can simply see the information and parts of the ship alongside its documents and traceability of each one; With the EMPLOYEE role, a user can do everything a CLIENT does but also create ProductLots and Activities; An ADMIN can do everything that the other roles can, and he can also manage other users and their roles (CRUD). This kind of definition of roles can be simply done by using the @PreAuthorized annotation of Spring:

```
@PreAuthorize("hasAnyRole('ROLE_ADMIN', 'ROLE_EMPLOYEE')")
or
@PreAuthorize("hasRole('ROLE_ADMIN')")
```

Two complex cases of implementation on this part of the system were the check of the fingerprint and the download of the file, these two cases are directly connected because the check for the document fingerprint happens every time a file is reached out for download. Firstly, the download endpoint receives the ID of the ProductLot that has the associated the target file and the key identifier of that file, after that, the corresponding ProductLot is fetched, the DocumentKey entity too and the constant path for that document in the S3 Bucket is built in order to fetch the file. With all this information, the file is fetched from the S3 Bucket as an InputStream and immediately converted to a byte array:

```
public byte[] download(String path,
                       String documentKey,
                       String documentFingerPrint) {
    try {
        var object = s3.getObject(path, documentKey);
        var inputStream = object.getObjectContent();
        byte[] bytes = IOUtils.toByteArray(inputStream);
        inputStream.close();

        checkFileReliability(documentFingerPrint, bytes);

        return bytes;
    } catch (IOException e) {
        log.error("Failed to store file in S3, e: {}",
                  e.getMessage());
        throw new IllegalStateException(
                  "Failed to download file in S3", e);
    }
}
```

With this byte array we can calculate the SHA-256 digest (fingerprint) and compare it to the one stored on the blockchain (DocumentKey fetched earlier), if they are not the same an error is thrown and the operation stopped:

```
private void checkFileReliability(String documentFingerPrint,
                                  byte[] bytes)
                                  throws IOException {
    InputStream inputStream = new ByteArrayInputStream(bytes);
    String checksumSHA256 = DigestUtils.sha256Hex(inputStream);

    if (!checksumSHA256.equals(documentFingerPrint)) {
        throw new IllegalStateException(
        "The finger prints don’t match, so the file was modified");
    }

    inputStream.close();
}
```

If they are the same, the byte array is returned with some metadata:

```
@Override
@GetMapping("/product/{productLotUuid}/document/{documentKey}/download")
@PreAuthorize("hasAnyRole('ROLE_EMPLOYEE', 'ROLE_CLIENT', 'ROLE_ADMIN')")
public ResponseEntity<ByteArrayResource> getDocument(
                                    @PathVariable String productLotUuid,
                                    @PathVariable String documentKey) {
    var data = smartContractService.getDocument(productLotUuid, documentKey);
    var filename = "file.pdf"; //only pdf for now

    return ResponseEntity
            .ok()
            .contentLength(data.length)
            .header("Content-type", "application/octet-stream")
            .header("Content-disposition", "attachment; filename=\"" +
                    filename + "\"")
            .body(new ByteArrayResource(data));
}
```

The communication with the Smart Contract API is made thought and a client called Open Feign that allows the creation of an interface for each API we wish to consume and define the various endpoints of that API as methods that can receive parameters, those can be the body, the headers, the path variables or anything that needs to be included in the request:

```
@FeignClient(value = "smartContractTraceabilityApi",
             url = "${app.blockchain.api.invoke-url}")
public interface SmartContractTraceabilityApiClient {

    @PostMapping(value = "/", consumes = APPLICATION_JSON_VALUE)
    Map<String, String> createActivity(
            @RequestHeader("Authorization") String bearerToken,
            String jsonRequest);
}
```

The Swagger (Open API) implementation allows the easy documentation of the endpoints that were implemented, by just specifying in code the operation that each endpoint performs, the possibilities of response and some examples for the body, a very clean and good documentation with a REST client is generated which also allows the easy testing of each endpoint, it even allows the authentication with bearer token, some example print screens can be found at Appendix C.1.

The Smart Contract defines the on-chain entities shown on the data model of Fig. 3.3, the definition of those entities can be found on Fig. C.5 of the Appendix C.2 and all the operations were already defined on the Table 3.1 on the Section 3.3.1.1. The most complex operation was the "GetTraceabilityByReferenceNum", this operation receives the reference number of a ProductLot and returns its traceability in the format exemplified on the Appendix A.2, the first step is to use the operation "ReadProductLotByReferenceNum" to check if the ProductLot exists and if it does, to get it. After that, a new function is called to build the traceability, this function will start by fetching all the activities and get the one that has the OutputProductLot equals to the one found previously, that is the activity that gave origin to the product that was requested to be traced, after this, this method will be called recursively in order to get the origin on the inputs and quantities used to build the product to be traces, find the complete implementation on Fig. C.6 of the Appendix C.2. This recursive call can be a problem if the ProductLot that is being trace has is very deep into the supply chain, but this will be approached in Section 4.

3.3.3.2 Frontend

In order to showcase all the implemmented funcionalities throught the web page and the mobile application, we can imagine a use case where three different users exist and are using the applications:

- The Administrator with the username "admin" will use the web application to:
    - Create designations for parts of the ship (products);
    - Create designations for activities;
    - Create users (and assign roles);
- The Employee 21 with the username "func21" will use the mobile application to:

- Scan components;
- Create components;
- Associate a component with other components;
- Associate documents with a component;
- And see all components

- The Client Pedro Araujo with the username “user“ will use the web application to:
    - See all parts of the ship (products) and each respective traceability;
    - See the documents associated with each part of the ship;

It is also important to refer that, like it was stated before, the Admin can do all the actions and the Employee can also see users, products and its traceabilities but in order to avoid repetition, the functionalities were splitted between the three users. Besides this, on Chapter 4, some tests are performed based on the supply chain created in this showcase.

Starting by the mobile application, a user will start by logging in, only an existing user with the “EMPLOYEE“ role can login, otherwise error messages will be shown, the user can also hide/show his password (Figure 3.5).

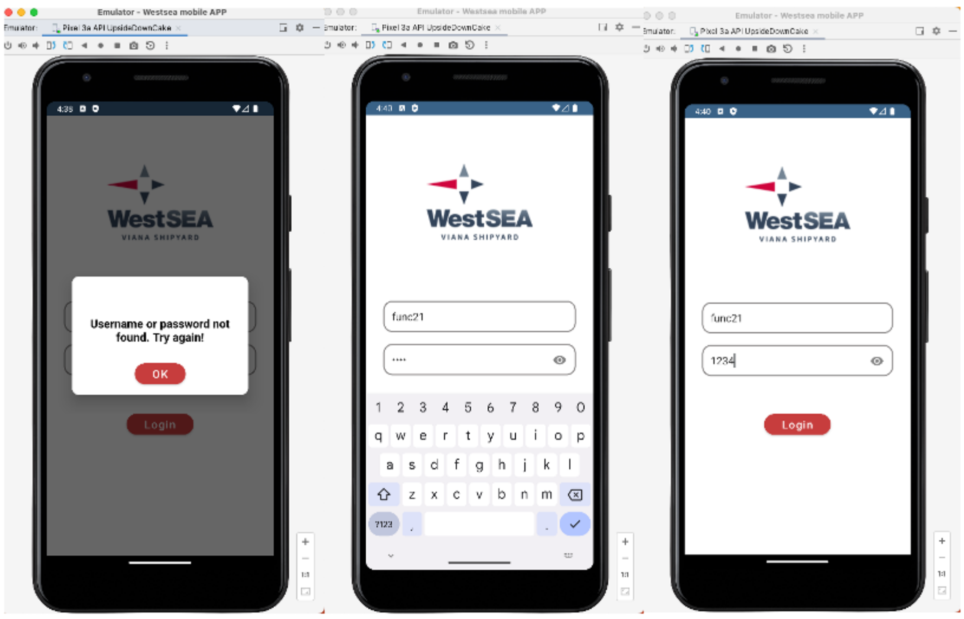

Emulator - Westsea mobile APP
WestSEA
VIANA SHIPYARD
Username or password not found. Try again!
OK
Login
func21
••••
1234

Figure 3.5: Mobile login screen

Right after the login, the user will be redirected to the home screen where he can Scan components by QR Code or RFID (Figure 3.6), this scans are simply examples of what the goal is, they were not fully implemmented because for this use case, we do not possess a real server that can provide this real information. So if the RFID button is clicked, nothing happens but if the QR Code button is clicked, there is the possibility for scanning a code, once the scan is a success, a toast is shown and the user is redirected to the “Add Component“ screen in order to simulate the scan, if it was a real one, the information would be auto filled.

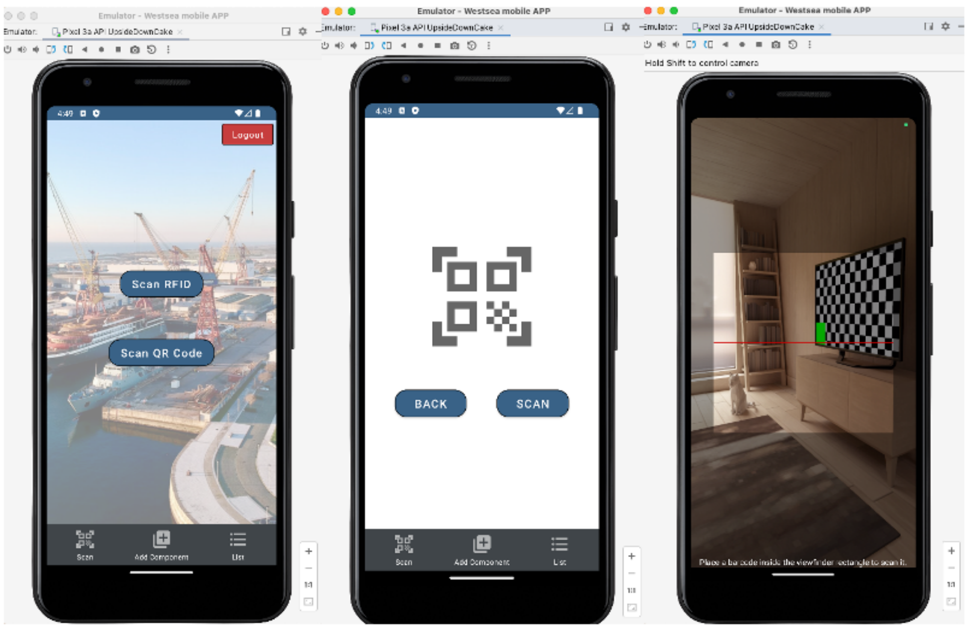

Emulator - Westsea mobile APP
Logout
Scan RFID
Scan QR Code
Scan
Add Component
List
BACK
SCAN
Hold Shift to control camera
Place a bar code inside the viewfinder rectangle to scan it.

Figure 3.6: Mobile scan screens

Following the bottom navigation bar, there is the “List“ page that shows to the user all the existing products, for each it is shown the the product designation, the reference number right bellow, the product type on the right and a button right bellow that button that shows more information about the product (Figure 3.7). To warn that some of this printscreens will contain trash data, whish is essecially data radomly and automatically added to the application in order to fill it.

Still on this screen, there is the possiblity of search by reference number, designation and product type, by writting something on the input field, the list will be automaticcly filtered down.

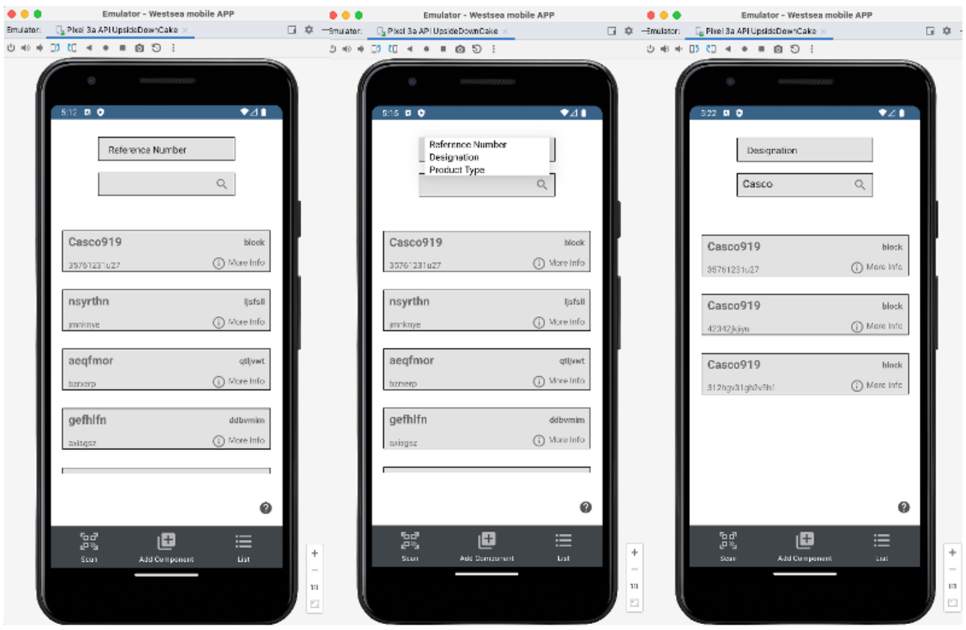
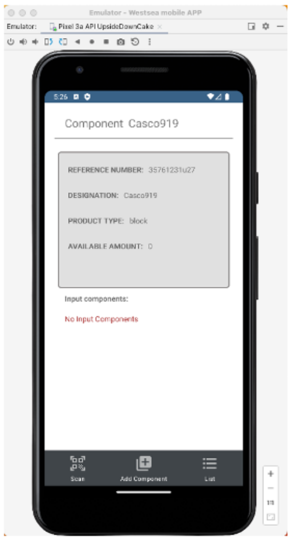

Emulator - Westsea mobile APP
Pixel 3a API UpsideDownCake
Reference Number
Designation
Product Type
Casco
Casco919 block 35761231u27 More Info
nsyrthn ljsfsll More Info
aeqfmor qtljvwt More Info
gefhlfn ddbvmim More Info
Casco919 block 42342jkiyu More Info
Casco919 block 312hgv31gh2v3h1 More Info
Scan
Add Component
List

Figure 3.7: Mobile list and search screen

When a user clicks on the component, the information is presented with more detail, including the available amount of that component that can be used and the input components that gave origin to that particular component, if any (Figure 3.8).


Component Casco919

REFERENCE NUMBER: 35761231u27

DESIGNATION: Casco919

PRODUCT TYPE: block

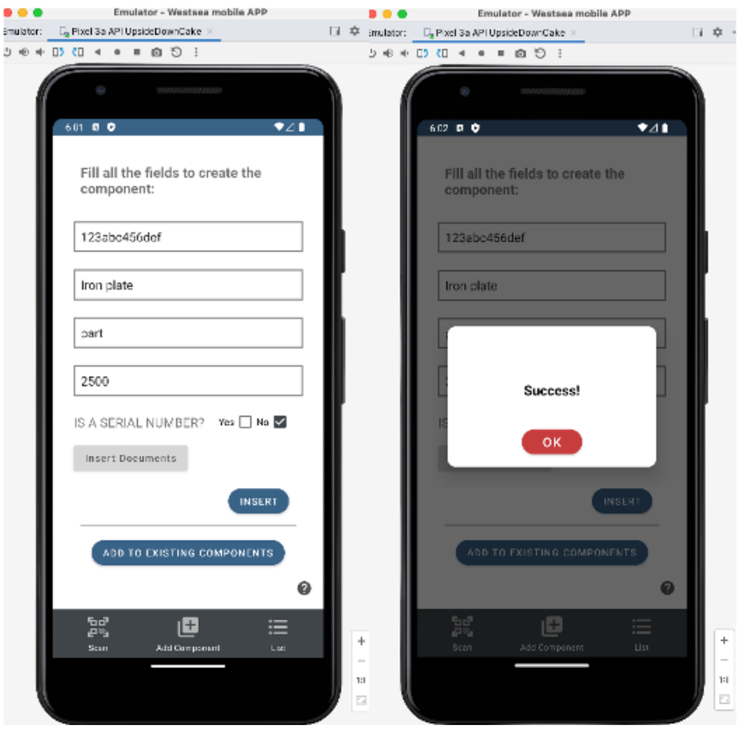
AVAILABLE AMOUNT: 0

Input components:

No Input Components

Figure 3.8: Mobile list info screen

The “Add component“ screen is used to create components by filling the input fields with a unique reference number, a designation, a product type, and an initial amount, there is also a checkbox to choose if that component has a serial number or not, if it has, the initial and total amounts will always be 1 because the component itself is unique, if not that component will represent a lot of equal components. A simple component without any input components or documents can be created just like this (Figure 3.9).


Emulator - Westsea mobile APP
Emulator: Pixel 3a API UpsideDownCake
Insert Documents
INSERT
ADD TO EXISTING COMPONENTS
OK
Scan
Add Component
List

Fill all the fields to create the component:

123abc456def

Iron plate

part

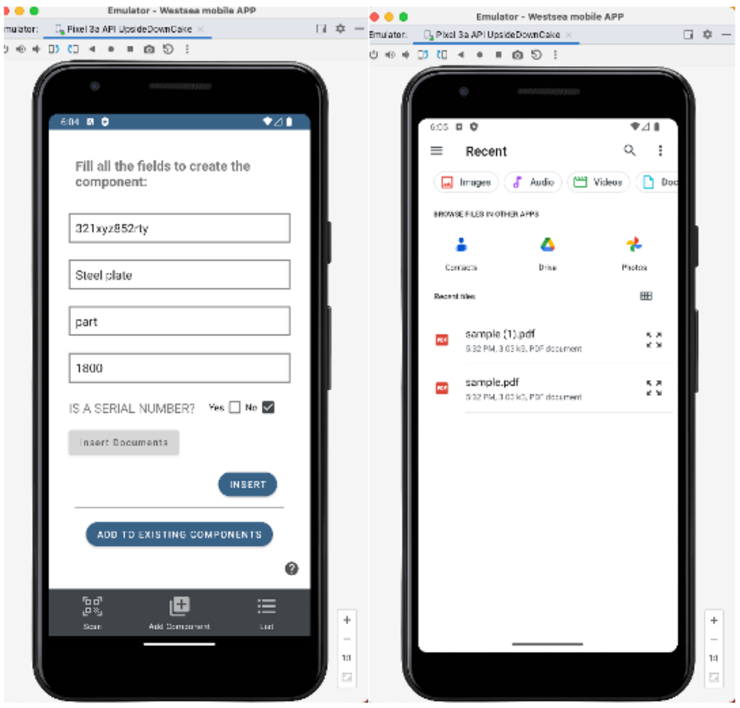
2500

IS A SERIAL NUMBER? Yes [✓ No]

Success!

Figure 3.9: Mobile simple component creation (Iron plate example)

Besides providing this information, there is also the possiblity to associate documents to the component, that can be later visualised on the web application, for that the user need to click on "Insert documents" and choose some from his storage (Figure 3.10).


Emulator - Westsea mobile APP
Pixel 3a API UpsideDownCake
Insert Documents
INSERT
ADD TO EXISTING COMPONENTS
Scan
Add Component
List
Emulator - Westsea mobile APP
Pixel 3a API UpsideDownCake
Recent
Images
Audio
Videos
BROWSE FILES IN OTHER APPS
Contacts
Drive
Photos
Recent files

Fill all the fields to create the component:

321xyz852rty

Steel plate

part

1800

IS A SERIAL NUMBER? Yes ☐ No ☑

sample (1).pdf
5:32 PM, 3.03 kB, PDF document

sample.pdf
5:32 PM, 3.03 kB, PDF document

Figure 3.10: Mobile component creation with documents (Steel plate example)

There is also the possibility to associate input components to the new component, that means the new component was built using those input componets, for that, after the input fields are field and the documents are associated, the user must click on “Add to existing components“. On that new page, the reference number of the input component must be used, for that, there is a dropdown with the designations of each component is order to ease the choices and a quantity used of that component to build this new one (Figure 3.11). All the choosed components will be added to a list, and when everything is ready, the user can click "Finish", that action will show a popup where the name of the activity that originated the new component needs to be choosen, this name comes from the “Activity Designations“ created by the admin on the web application. After all this, the new component can be created with all this associations (Figure 3.12).

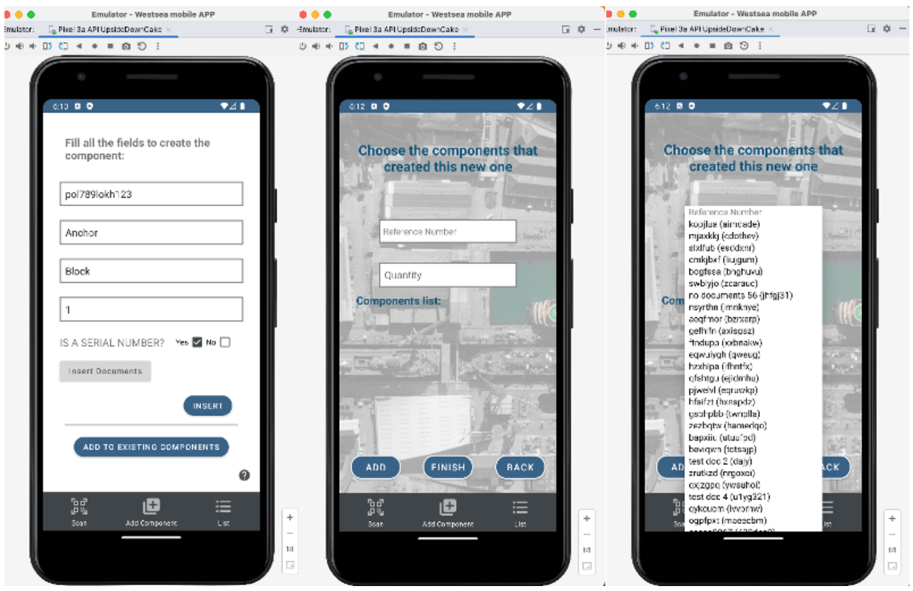

Fill all the fields to create the component:
pol789lokh123
Anchor
Block
1
IS A SERIAL NUMBER? Yes No
Insert Documents
INSERT
ADD TO EXISTING COMPONENTS
Choose the components that created this new one
Reference Number
Quantity
Components list:
ADD
FINISH
BACK
Scan
Add Component
List

Figure 3.11: Mobile component creation with input components (Anchor example)

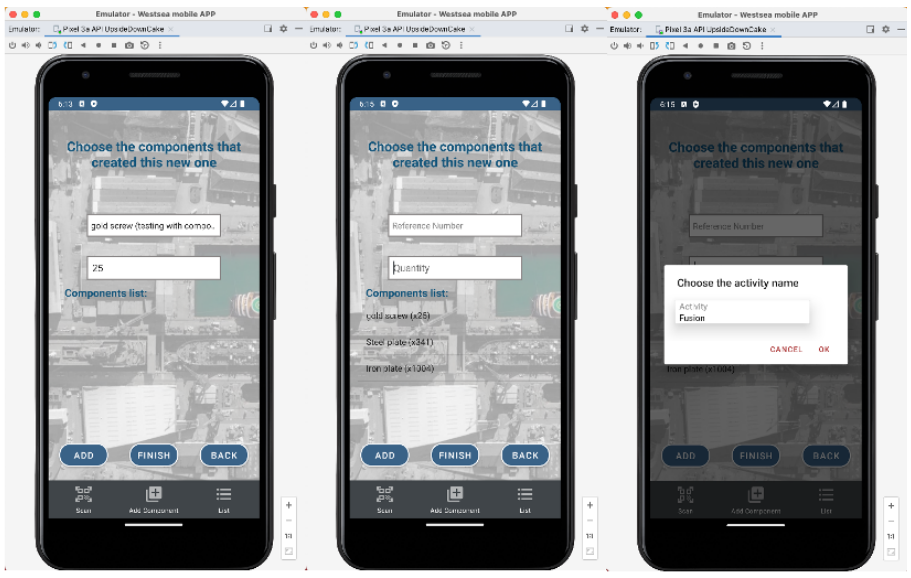
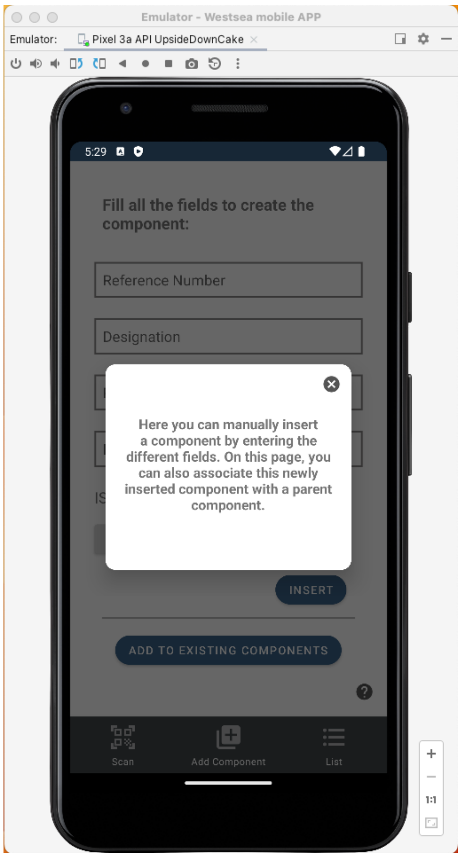


Figure 3.12: Mobile component creation input components association (Anchor example)

In many screens there is the interrogation mark sign “¿‘ that can be clicked and it will show a popup with information about the screen that is currently open (Figure 3.13).


Emulator - Westsea mobile APP

Emulator: Pixel 3a API UpsideDownCake

5:29

Fill all the fields to create the component:

Reference Number

Designation

Here you can manually insert a component by entering the different fields. On this page, you can also associate this newly inserted component with a parent component.

Figure 3.13: Mobile help popup on the “Add component“ screen

On the web application, a user also starts by logging in and depending on the type of role that the logge in user has, some parts of the platform will be hidded, some printscreens can be found on Apendix C.6 with the different views of each user. A client will only have acccess to the page where he can see the components and their information. An employee will be able to see the components, its information, all the designations and the users but he won’t be able to create/update/delete any of them. While an admin has access to everything. Once more, there are verifications in order to make sure the user exists (Figure 3.15) and adds all the needed information (Figure 3.14).

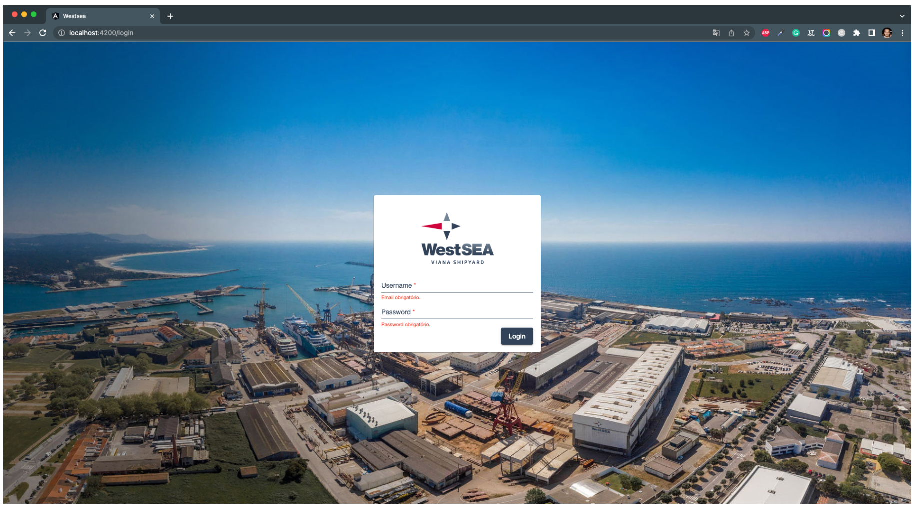
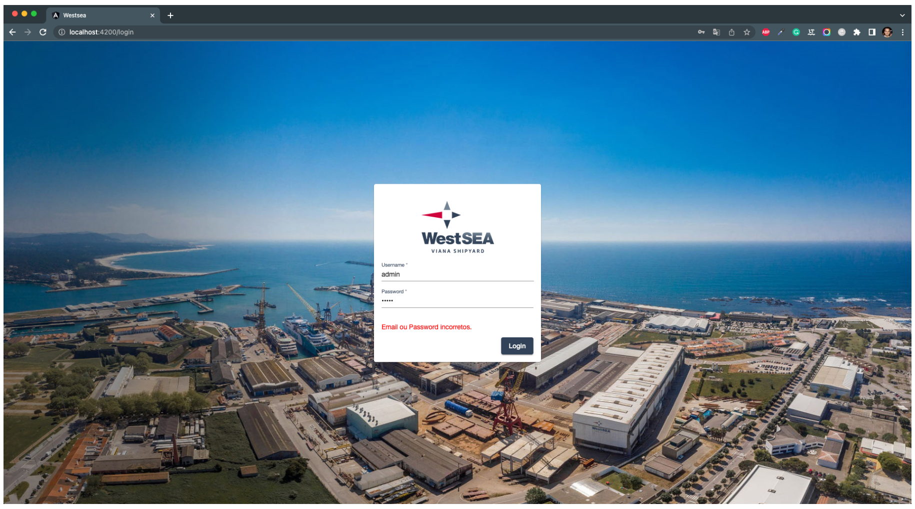


Figure 3.14: Web login with basic validation



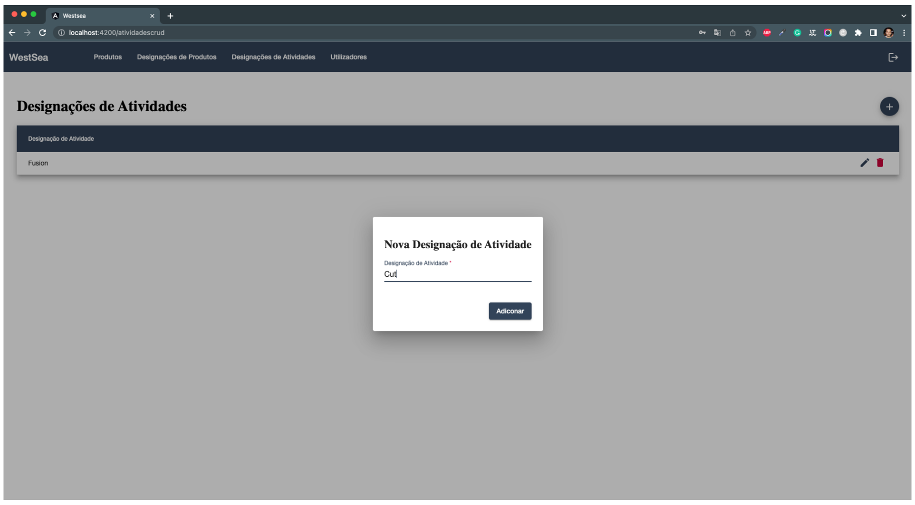
admin

•••••

Email ou Password incorretos.

Figure 3.15: Web login with user validation

Firstly, lets see the option that both Admin and Employee can see but only the Admin can create/update/delete. Following the navigation bar above, we can see the “Designações de Produtos“ and “Designações de Atividades“ those options will respectivly redirect to the products designation page and activities designation page, both allow to do the same thing in the exact same way, so for this example lets see only the activities designation page. A list is shown with all the activity designations, this is the exact same list that appears on the mobile application uppon the addition on input components to a new component. Here there the possibility to add an activity designation by clicking on the top right corner button with the plus icon “+“, this will show a popup with an input field, by filling this and clicking on Add (Figure 3.16), the designation will be created (Figure 3.17).



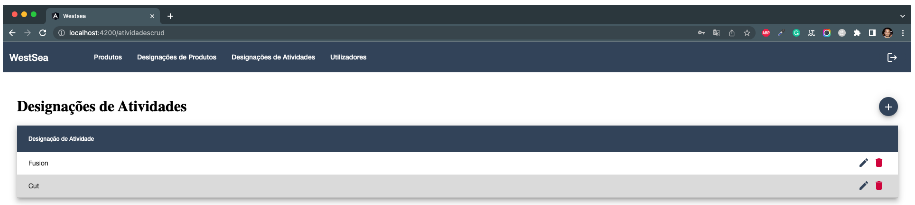
Designações de Atividades

Designação de Atividade
Fusion

Nova Designação de Atividade

Designação de Atividade *
Cut

Figure 3.16: Activity Designation creation popup

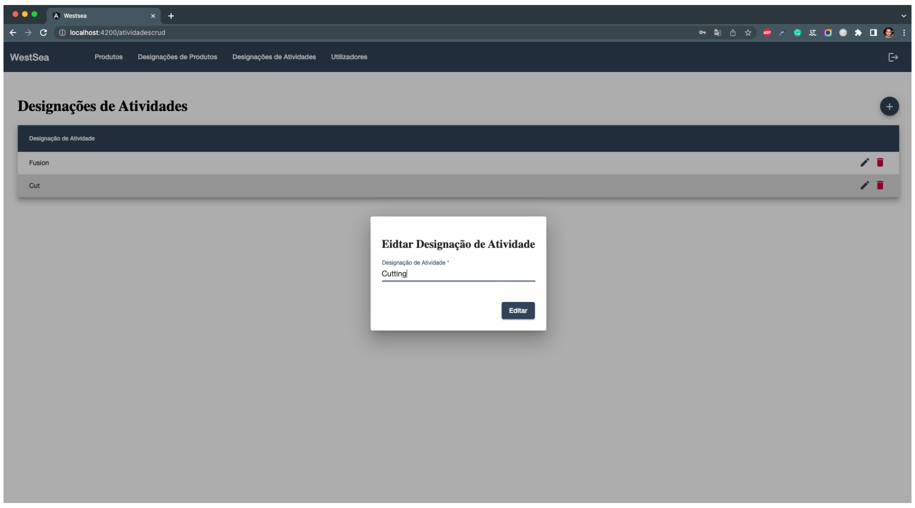


Designações de Atividades

Designação de Atividade
Fusion
Cut

Figure 3.17: Activity Designation list after creation

There is also the possibility of updating an existing activity designation by clicking on the pencil icon on the right side of the target activity designation. And just like the creation, there will be a popup with an input field, after adding the new designations and clicking on Add (Figure 3.18), the activity designation will be updated (Figure 3.19).



Designações de Atividades

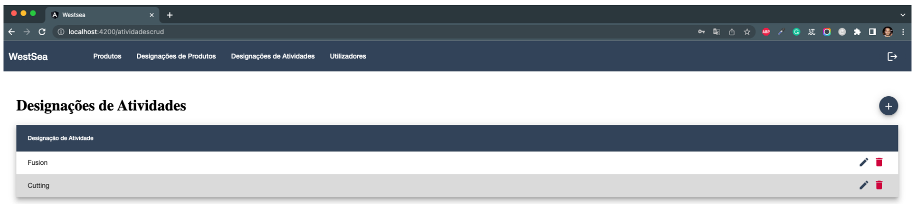
Designação de Atividade
Fusion
Cut

Eidtar Designação de Atividade

Designação de Atividade *

Cutting

Figure 3.18: Activity Designation update popup

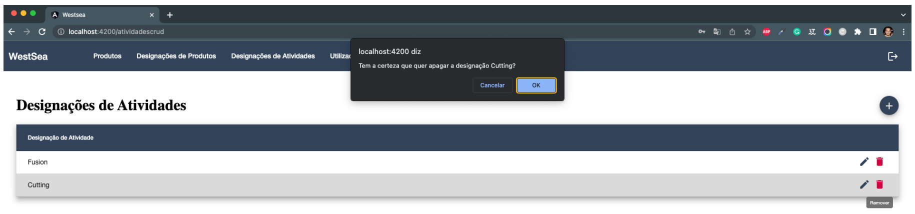


Designações de Atividades

Designação de Atividade
Fusion
Cutting

Figure 3.19: Activity Designation list after update

The final action on this page is to delete an activity designation by clicking on the trash can icon on the right side of the target activity designation. An alert will appear in order to confirm this decision (Figure 3.20), after accepting, the designation will be deleted (Figure 3.21).



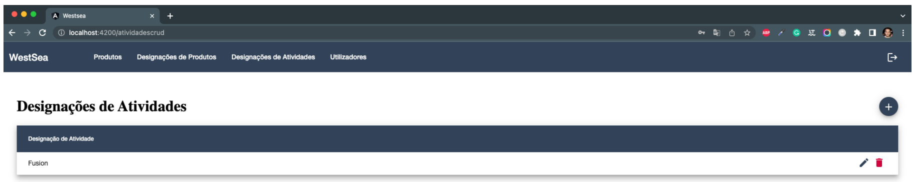
localhost:4200 diz

Tem a certeza que quer apagar a designação Cutting?

Designações de Atividades

Designação de Atividade
Fusion
Cutting

Figure 3.20: Activity Designation delete alert

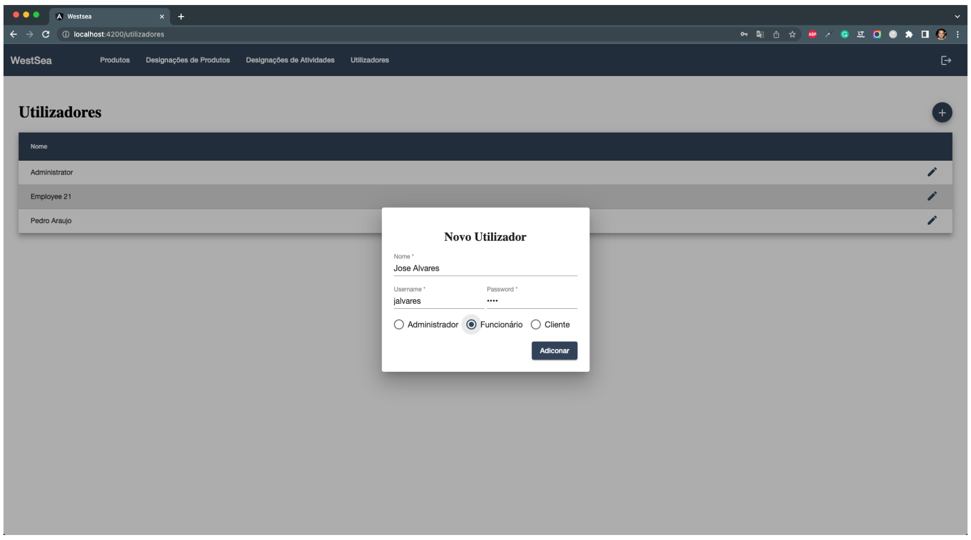


Designações de Atividades

Designação de Atividade
Fusion

Figure 3.21: Activity Designation list after delete

On the navigation bar above there is the page "Utilizadores", that option redirects the user to the users page, where a list is shown with all users on the system. Here there is the possibility to add an user by clicking on the top right corner button with the plus icon "+", this will show a popup requesting some information about the user and role that user should have, by filling this and clicking on Add (Figure 3.22), the user will be created (Figure 3.23).



Utilizadores

Nome
Administrator
Employee 21
Pedro Araujo

Novo Utilizador

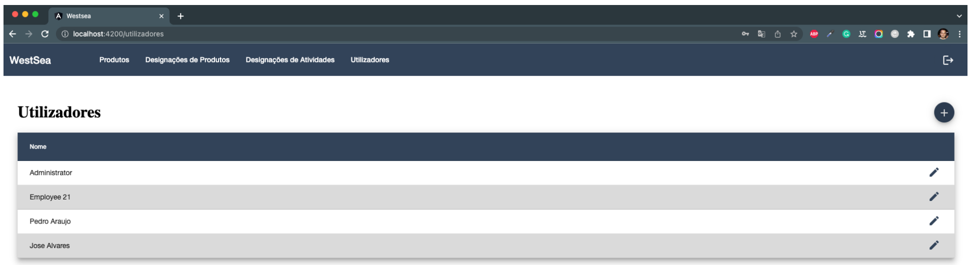
Nome * Jose Alvares

Username * jalvares

Password * ••••

Administrador [✓ Funcionário] Cliente

Figure 3.22: Users create popup

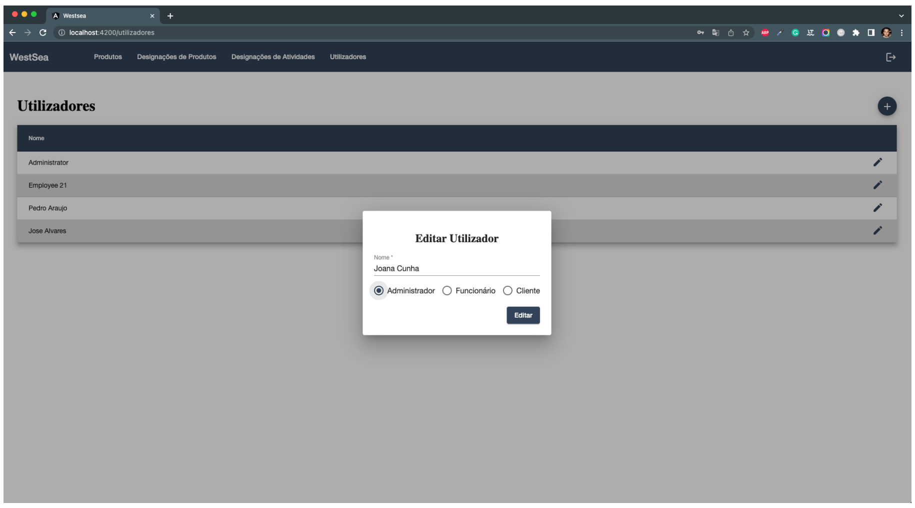


Utilizadores

Nome
Administrator
Employee 21
Pedro Araujo
Jose Alvares

Figure 3.23: Users list after create

There is also the possibility of updating an existing user by clicking on the pencil icon on the right side of the target user. And just like the creation, there will be a popup that will allow the change of name and role, after filling this and clicking on “Editar“ (Figure 3.24), the user will be updated (Figure 3.25).



Utilizadores

Nome
Administrator
Employee 21
Pedro Araujo
Jose Alvares

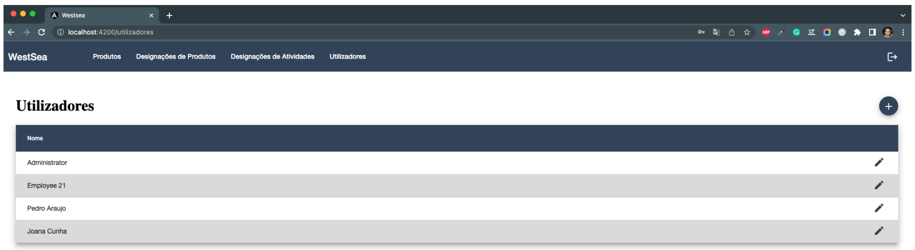
Editar Utilizador

Nome *
Joana Cunha

[✓ Administrador] Funcionário Cliente

Editar

Figure 3.24: Users update popup

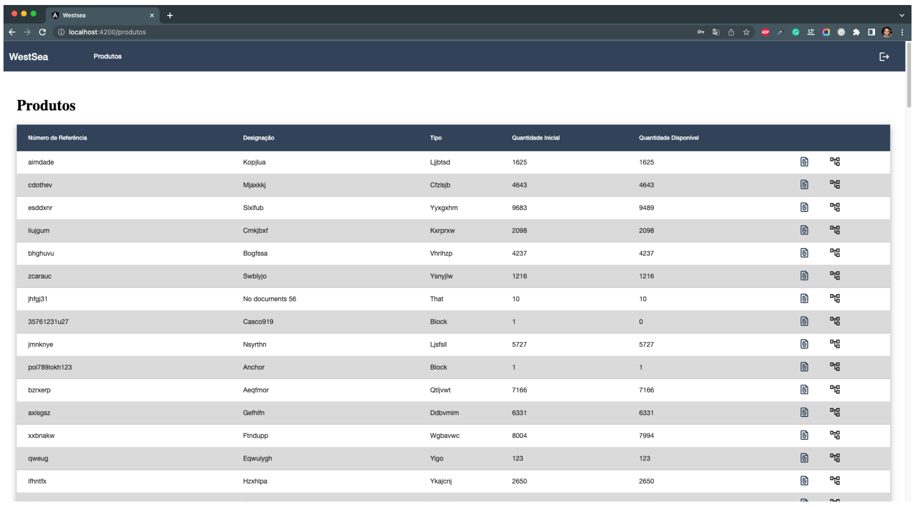


Utilizadores

Nome
Administrator
Employee 21
Pedro Araujo
Joana Cunha

Figure 3.25: Users list after update

In order to show a different view of user, lets use the client “user“ for the next section. A client can only see the page “Produtos“ that shows all products of the system on a table with their designation, type, initial and avalable quatity and also two options for each, one to see the associated documents and another to see the associatiated components (Figure 3.26).

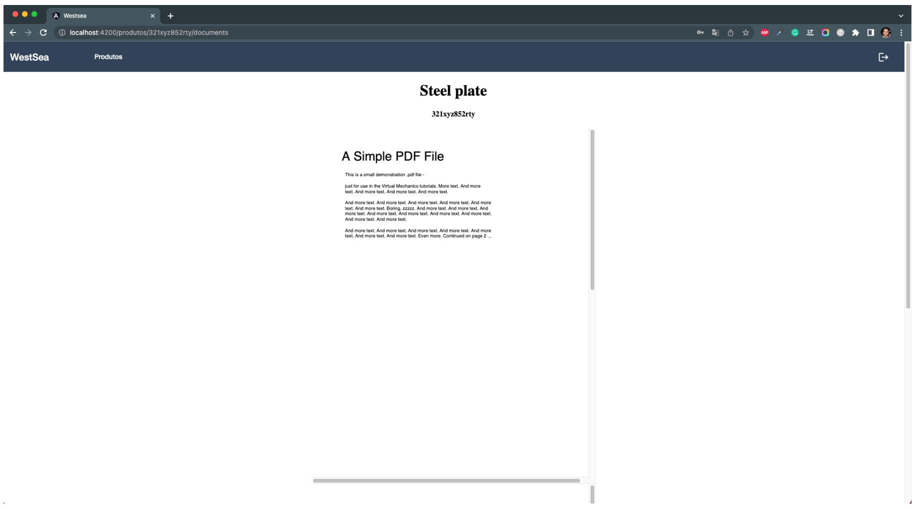


Produtos

Número de Referência	Designação	Tipo	Quantidade Inicial	Quantidade Disponível
aimdade	Kopjlua	Ljjbtsd	1625	1625
cdothev	Mjaxkkj	Cfzlsjb	4643	4643
esddxnr	Slxlfub	Yyxgxhm	9683	9489
liujgum	Cmkjbxf	Kxrprxw	2098	2098
bhghuvu	Bogfssa	Vhrihzp	4237	4237
zcarauc	Swblyjo	Ysnyjlw	1216	1216
jhfgj31	No documents 56	That	10	10
35761231u27	Casco919	Block	1	0
jmnknye	Nsyrthn	Ljsfsll	5727	5727
pol789lokh123	Anchor	Block	1	1
bzrxerp	Aeqfmor	Qtljvwt	7166	7166
axisgsz	Gefhlfn	Ddbvmim	6331	6331
xxbnakw	Ftndupp	Wgbavwc	8004	7994
qweug	Eqwuiygh	Yigo	123	123
ifhntfx	Hzxhlpa	Ykajcnj	2650	2650

Figure 3.26: List of products (components)

By clicking on the first button on the right side of a product, the list of associated documents will be shown, lets click on the component created before the “Steel plate“ and see the two documents that were associated earlier (Figures 3.27 and 3.28).



Steel plate

321xyz852rty

A Simple PDF File

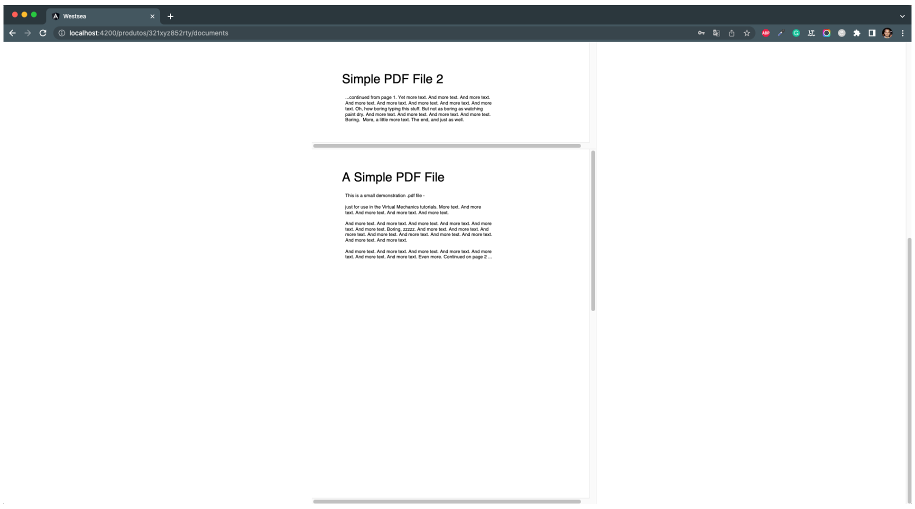
This is a small demonstration .pdf file -

just for use in the Virtual Mechanics tutorials. More text. And more text. And more text. And more text. And more text.

And more text. And more text. And more text. And more text. And more text. And more text. Boring, zzzzz. And more text. And more text. And more text. And more text. And more text. And more text. And more text. And more text. And more text.

And more text. And more text. And more text. And more text. And more text. And more text. And more text. Even more. Continued on page 2 ...

Figure 3.27: Products (components) documents - 1



Simple PDF File 2

...continued from page 1. Yet more text. And more text. And more text. And more text. And more text. And more text. And more text. And more text. Oh, how boring typing this stuff. But not as boring as watching paint dry. And more text. And more text. And more text. And more text. Boring. More, a little more text. The end, and just as well.

A Simple PDF File

This is a small demonstration .pdf file -

just for use in the Virtual Mechanics tutorials. More text. And more text. And more text. And more text. And more text.

And more text. And more text. And more text. And more text. And more text. And more text. Boring, zzzzz. And more text. And more text. And more text. And more text. And more text. And more text. And more text. And more text. And more text.

And more text. And more text. And more text. And more text. And more text. And more text. And more text. Even more. Continued on page 2 ...

Figure 3.28: Products (components) documents - 2

By clicking on the second button on the right side of a product, the traceability of that component will be shown in the form a graph, lets click on the component created before the “Anchor“ and see its traceability (Figure 3.29). In this page there is the possibility of hovering the mouse on a component and see the quantity of that product that were used. A user can also click on a component to its traceability. By analysing the graph, we can see the input documents of the product selected and the input documents of each of those products, getting the full traceability.

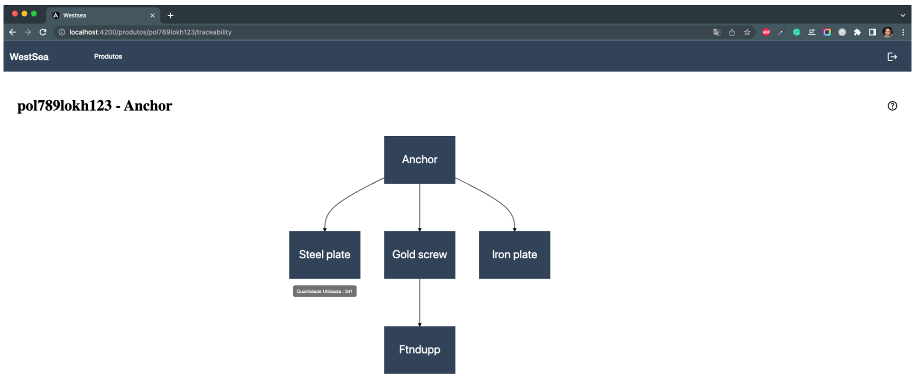


Figure 3.29: Products (components) traceability representation with a graph

Regarding the implemmentation of the web and mobile applications, no major chalanges were faced since the basic implemmentation was already done but there was a lot of refactoring on both applications.

As it was stated before, the bases for the mobile and web applications were made by students but there was a lot of refactor involving both applications.

On the mobile application, in terms of UI, it looked good so there were no changes needed but in terms of functionalities, there were some very important ones that had to be added and others that had to be removed. There was a functionality to list pending products that basicly stored locally the products that were added and they could only be submitted in a different screen in order to make sure there was no wrong data, that is a good way of handeling the process of product creation but for this scope it is just to much complexity and it could be added later if needed by the business. It was not possible to associate more then one input component to a new component before, so that functionality also had to be added also side the functionalitty to see the input components of a component when opening the "More info" screen. Before it wasn't possible to create a component without an activity (without inputs), that was also a new functionalitty that was added. And finally, regarding the documents, it was not possible to associate documents to a component, this functionalitty was addded too, there was the possibility of checking the documents associated with each component but it was removed, that should be done only on the web application. So, on this application a lot of functionalitties were missing and some were not suppose to exist but all of that was included on the refactoring.

On the web application, there was also lots of changes but in this case they were mostly about the design and user experience, the UI was not good and the application was very confusing and hard to navigate. It terms of functionalities, the only one missing was a page to view the documents associated each component, that was added. There were also a few things to fixed regarding the authorization on the page for each type of user. Besides that, the application design was completly rethought and implemmented, a few print screens can be found on Appendix C.7 that ilustrate how the application was before.

3.3.3.3 Infrastructure

In order to create an easy way to run the general REST API in any machine and consequently easily deploy it in any VM, docker was used. A Dockerfile was created to simply run the compiled JAR file, which is the result of the project compilation:

```
FROM openjdk:17
COPY target/westsea-traceability-0.0.1-SNAPSHOT.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","/app.jar"]
```

That Dockerfile will be used on the docker-compose file, that file defines three services that are needed for the general REST API, there is the MongoDB database, which runs with all the default configurations; the actual API that is provided by the Dockerfile, which will run on the port 8080; and the mongo-express service that runs on the port 8081, this service is a user interface used to easily interact with a MongoDB instance, the main purpose of this service is to ease the life of the IPVC students, so they can delete and insert any data for their tests, usually a service like this is not required, a few print screens of this interface can be found on the Section C.3 of the

Appendix C. The docker-compose file used for the system looks like this but with other credentials for the correct environment:

```
version: "3.3"
services:
  mongodb:
    hostname: mongodb
    image: mongo
    ports:
      - 27017:27017
    volumes:
      - data:/data
    environment:
      - MONGO_INITDB_ROOT_USERNAME=rootuser
      - MONGO_INITDB_ROOT_PASSWORD=rootpass
  westsea-traceability:
    build: .
    restart: always
    ports:
      - 8080:8080
    depends_on:
      - mongodb
  mongo-express:
    image: mongo-express
    restart: always
    ports:
      - 8081:8081
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME=rootuser
      - ME_CONFIG_MONGODB_ADMINPASSWORD=rootpass
      - ME_CONFIG_MONGODB_SERVER=mongodb
volumes:
  data: { }
```

Besides the VMs, this API also runs on Local environment, and there are some different configurations for that environment and for the VMs, for example, the URL where the Smart Contract API runs is different, on local should be “localhost:SOMEPORT“ but on a VM should be a fixed IP; the bucket name for the S3 connection should be different on every environment because we don’t want the files from Local to be mixed with the files from the VMs; and also the MongoDB connection can be different, on the VMs it will be used the service from the docker-compose file but on Local maybe we want to use our local instance. On Spring, there is a main configuration file called “application.properties“, this file usually contains all the configurations for the application to work the way we want to, but in this case, this file will be used to define some generic configurations and to define the profile of the application, depending on the profile in use, another “.properties“ file will be used along with this one. The profiles map to each environment that exists, for this case there is the Local environment and the VM environment, so the ‘application.properties‘ file looks like this:

```
server.port=8080
jwt.secret=secret
spring.servlet.multipart.max-file-size=100MB
spring.profiles.active=local
```

And it defines the local as the active profile so the “application-local.properties“ will be also used, this one looks like this (with the correct keys and credentials):

```
spring.data.mongodb.authentication-database=admin
spring.data.mongodb.database=wstrace
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.auto-index-creation=true
spring.mvc.pathmatch.matching-strategy=ant_path_matcher
app.http.security.enabled=false
app.http.security.cors=http://localhost:4200
app.http.security.headers=*
app.http.security.methods=GET,POST,PUT,DELETE,OPTIONS,HEAD
app.http.security.exposedHeaders=Page-Content
app.blockchain.network-admin.user=admin
app.blockchain.network-admin.password=adminpw
app.blockchain.api.base-url=http://localhost:8801
app.blockchain.api.channel=my-channel1
app.blockchain.api.chaincode=trace-westsea
app.blockchain.api.invoke-url=${app.blockchain.api.base-url}\
  /invoke\
  /${app.blockchain.api.channel}\
  /${app.blockchain.api.chaincode}
app.aws.s3.bucketName=westsea-trace-docs-bucket-local
app.aws.s3.accessKey=AKIARZIVDAGRWC7BOPML
app.aws.s3.secretKey=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
```

And if the defined profile was VM, the “application-vm.properties“ would be used, and that file looks like this (with the correct keys and credentials):

```
spring.data.mongodb.authentication-database=admin
spring.data.mongodb.username=rootuser
spring.data.mongodb.password=rootpass
spring.data.mongodb.database=wstrace
spring.data.mongodb.host=mongodb
spring.data.mongodb.port=27017
spring.data.mongodb.auto-index-creation=true
spring.mvc.pathmatch.matching-strategy=ant_path_matcher
app.http.security.enabled=false
app.http.security.cors=http://localhost:4200
app.http.security.headers=*
app.http.security.methods=GET,POST,PUT,DELETE,OPTIONS,HEAD
app.http.security.exposedHeaders=Page-Content
app.blockchain.network-admin.user=admin
app.blockchain.network-admin.password=adminpw
app.blockchain.api.base-url=http://34.142.64.171:8801
app.blockchain.api.channel=my-channel1
app.blockchain.api.chaincode=trace-westsea
app.blockchain.api.invoke-url=${app.blockchain.api.base-url}\
  /invoke\
  /${app.blockchain.api.channel}\
  /${app.blockchain.api.chaincode}
app.aws.s3.bucketName=westsea-trace-docs-bucket-dev
app.aws.s3.accessKey=AKIARZIVDAGRWC7BOPML
app.aws.s3.secretKey=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
```

Regarding the VMs, the AWS EC2 and Google's Compute Engine interfaces won't be shown because they contain too much sensitive information about the IPs and several other configurations, but we can assume that the only extra configurations that were made were to expose the required ports to the public, add HTTPS and firewall.

The AWS S3 did not need any special configuration on the Cloud, the access and secret keys were extracted from there into the code and everything was configured from there. A few print screens of the AWS S3 can be found on the Section C.4 of the Appendix C

Chapter 4

Experimental Validation

In this chapter, it is presented the validation of the system, a few points will be discussed, namely: if this system satisfies the requirements of a traceability system; problems that need to be fixed; and some performance tests on the backend (API). It is also important to mention that this system does not have an evaluation because it could not be tested on a real professional environment as it would have been too costly, but there are unit tests for the most important methods used o n the General API, those being the methods to interact with the Smart Contract API and the method to download a document that also certifies if the document was adulterated, or it is legit. Those unit tests are presented on Appendix C.10. This chapter is separated into three sections, the first section will elaborate about how to validate a traceability system and how this system fits; the second section will state some problems already identified that need to be fixed; and the final section will show some tests that evaluate the overall performance of the systems' backend.

4.1 Traceability System validation

Validating a traceability system is an important step in ensuring that it is effective and meets the requirements of the industry and product being traced, after a lot of research and analysis it is possible to compile the most important requirements to consider on a traceability system:

1. Unique identification: Each product or material must have a unique identifier that can be used to track its movement through the supply chain. Depending on the business, it could be a serial number, barcode, RFID tag, or any other type of identification.

2. Data capture: The traceability system must be able to capture data at various points in the supply chain, including at the point of origin, during transportation, and at the point of sale. This data should include any relevant information about the product that is being traced.

3. Data storage and management: The data captured by the traceability system must be stored and managed in a way that allows it to be easily accessed and analyzed. This usually means that some kind of database or data management system is required.

4. Integration with other systems: The traceability system may need to integrate with other systems of the company, such as inventory management or logistics systems, in order to provide a complete picture of the movement of products or materials through the supply chain.

5. Security: The traceability system must be secure to prevent unauthorized access to sensitive data. This may require the use of encryption, authentication, or other security measures.

6. Compliance: Depending on the industry and product being traced, there may be regulatory requirements for traceability systems. The system must be designed to meet these requirements.

7. Transparency: The traceability system should provide transparency to all stakeholders involved in the supply chain, including consumers, regulators, and other interested parties. This can help build trust and ensure that products are safe and of high quality.

By analyzing all those points, it is possible to map them into the system developed:

1. Unique identification: This depends on the company, in this case study we know that The West Sea [20] has an unique identifier in each component of the ship and on the lot itself and the system built supports those identifiers and reference numbers by storing them and the scan of codes is also partially implemented on the mobile application.

2. Data capture: With this system, it is possible to capture the several components in their several states by simply using the mobile application to scan the code or manually insert the information and documents.

3. Data storage and management: The traced data is stored on a blockchain, and it is also provided a very simple way to access that information through an authenticated API. To ease even more this process there is the web application that presents all that information in a simple organized way.

4. Integration with other systems: This point also depends on the business, this system is not integration with any of The West Sea [20] systems, but it can be done quite easily.

5. Security: All the applications and APIs are secured thought authentication and the data is fully safe by being stored on the blockchain, as it was stated several times before, it provides extreme safety for the data that is stored there, some more information can be found on Section 2.1.3.1.

6. Compliance: This point is not fully adopted on this system mainly because there is no real data being used, and we had no knowledge to the regulatory requirements, but it can be implemented upon the need.

7. Transparency: Through the web application, all the information is available to the concerned stakeholders.

Taking all this into consideration, we can say that the system developed follows almost all the requirements of a traceability system and what is nor achieve is mainly due to the fact that we are not handling real data on a professional environment, if that was the case, we could easily ensure that all the requirements were followed.

4.2 Identified problems

The main problem in this system is the usage of recursion on the Smart Contract in order to gather the information required to form the traceability object, recursion is explained on Section 2.1.2.3 and their usage on the Smart Contract is explained on Section 3.3.3.1 and the complete implementation is shown on Fig. C.6 of the Appendix C.2. In this case, recursion can introduce some critical issues if it's too deep like stack overflow, that means if the recursion is too deep, that stack will be called too many times in order to keep track of function calls, and it usually results in an error; recursion will use a lot of memory each function call adds a new stack frame to the call stack, resulting in memory usage issues and maybe memory leaks; the performance is also greatly affected because of the overhead in managing function calls, resulting in a lengthy operation. All this is bad as it will slow down the operation of calculating and retrieving the traceability of a product, and it also makes it unstable and unreliable if the recursion depth is too long.

Some tests have been executed on the endpoint that calls the traceability method from the API to the Smart Contract in order to fully determine how viable this recursive method is. If we take into consideration a *depth of n*, n being the number of times the function calls itself, we can see that for *n = 1* the call takes 3.67 seconds (Appendix C.8.1), for *n = 5* it takes 4.08 seconds (Appendix C.8.2) and for *n = 15* it takes 4.27 seconds (Appendix C.8.3), with this information we can conclude that the recursive function has exponential complexity. The difference in execution time between a depth of 15 and a depth of 5 is significant. Depth 15 is three times greater than depth 5, but the execution time is more than one and a half times higher. This suggests that the execution time of the function is increasing exponentially as the depth of recursion increases. Furthermore, the absolute difference in execution time between depths 15 and 5 is quite large, which suggests that the function may not be very efficient for higher depths. If the depth of recursion needs to be increased further, it is possible that the function may become unfeasible to execute in a reasonable time.

Nevertheless, this problem can be addressed and there are a number of ways to solve it to make this operation reliable and fast, the best for this case is to not use it at all and replace it by an interactive approach, there are many solutions that fit this problem, two of them would be: to use a stack, this is a data structure that will manage the state of the operation avoiding the usage of the call stack and consequently avoiding the stack overflow error; or to use memoization, this is consists on storing the results on cache or locally (it can involve a local database) in order to reuse them. Both approaches are interesting but in this case, to solve the recursion problem, we could simply remove it and *apply the stack data structure*.

Recursion is a powerful tool to solve certain problems, but in this context it is a trap because it will always make the operation unreliable, as there it is not possible to know how deep the recursion will be, we can estimate that a recursion around 15 of depth would already trigger a stack overflow error or make the operation extremely slow to the point that is not usable.

4.3 Performance Tests

In order to test the overall performance of this system, some load tests, these types of tests are used to determine how well a system or application can perform under heavy user traffic or high volume usage by simulating numerous concurrent users or transactions to measure the system's ability to handle the increased load without degrading its performance or causing it to crash. Performance was not a goal of this system, so it is not expected to perform very well, in a professional environment, this would be an important goal/mark it is definitely something that can and should be improved in this system.

In order to test the complete backend including interactions with the database, the Smart Contract and the blockchain, the load tests will run against the main general API using a tool called Apache JMeter [13] and it takes into consideration the use case explained in Section 3.3.3.2.

By looking at the print screens on the Appendix C.9, we can conclude that the performance is not good at all, the concurrency is not optimal at all as it fails if at least ten users are calling the same endpoint or if the same endpoint is called ten times in the same timestamp. But it is also possible to conclude that this performance issues are a result of the communication between the API and the Blockchain, in a production ready environment, the Blockchain would be optimized with a lot more resources, so we can expect that the performance would be much higher. Nonetheless, this is a problem that should be targeted quickly if this was to deploy and use.

Chapter 5

Conclusions and Future Work

5.1 Pros, Cons, and Pain points to make the change for a system like this

5.1.1 Is it worth the change?

Implementing a traceability system based on the blockchain technology has the potential to bring numerous benefits to companies. These benefits include enhanced transparency, improved traceability, increased efficiency, reduced fraud and errors, and better customer satisfaction. However, implementing a blockchain-based traceability system can also be complex, costly, and time-consuming. Companies must conduct a thorough analysis of the potential benefits and risks to determine whether a blockchain-based traceability system is appropriate for their supply chain, and that is the only way to know if this type of system is worth it or not for a particular company.

5.1.2 Pros

Implementing a traceability system based on the blockchain technology has several advantages. The benefits of a blockchain-based traceability system include enhanced transparency, improved traceability, increased efficiency, reduced fraud and errors, and better customer satisfaction, these advantages are achieved by creating an immutable and transparent record of transactions. Each transaction on the blockchain is verified and recorded by multiple nodes, and once added to the blockchain, it cannot be altered or deleted. This makes the system highly secure and trustworthy, as it eliminates the possibility of fraud or errors caused by human manipulation or error. Additionally, the transparent nature of the blockchain allows for greater accountability and visibility throughout the supply chain, enabling companies to quickly identify and address any issues or discrepancies These advantages can help companies to achieve greater supply chain visibility, streamline processes, and enhance customer trust and loyalty.

5.1.3 Cons

While implementing a traceability system based on the blockchain technology can bring significant benefits to companies, there are also some challenges that need to be considered. The disadvantages of a blockchain-based traceability system include high complexity, cost, interoperability, scalability, and security risks. So, besides the many benefits a system like this can provide, one of the main risks is the potential for a 51% attack, where a single entity gains control of the majority of the nodes on the blockchain network and can manipulate the data. Another risk is the use of smart contracts, they can have bugs or vulnerabilities that hackers could exploit to steal funds or data. Additionally, the use of public keys and wallets for transactions could potentially lead to identity theft or fraud if these are not properly secured. Finally, the use of private blockchains could create centralized control, making the system more vulnerable to attacks or manipulation by a single entity, also confronting the decentralized way that should be followed. It's essential to consider these risks and take appropriate measures to mitigate them when implementing a blockchain-based traceability system. These challenges can make it difficult for companies to implement a blockchain-based traceability system, requiring them to invest in technology infrastructure, train employees, and update existing systems.

5.1.4 Pain Points

Implementing a traceability system based on the blockchain technology can be a complex and costly process, requiring technical expertise, data integration, standardization, legal compliance, and change management. These pain points can make it challenging for companies to implement a blockchain-based traceability system successfully. Companies must assess these pain points and develop a comprehensive plan to address them to ensure the success of their implementation, those would be:

- **Technical expertise** - it requires a high level of technical expertise, including knowledge of distributed ledger technology, smart contracts, and other relevant technologies. Companies may have to hire external experts or train existing staff to develop the necessary technical capabilities.

- **Data integration** - The implementation of a system like this requires the integration of data from different sources and systems in the supply chain. This can be a challenging task, as the data must be standardized and reconciled to ensure its accuracy and consistency.

- **Cost** - it can be expensive, and the cost of investment in technology infrastructure, training employees, and updating existing systems can be significant. Companies must conduct a thorough cost-benefit analysis to determine if the investment is justified.

- **Standardization** - blockchain technology is still in its early stages, and there is a lack of standardization in the industry. This can lead to interoperability issues with other stakeholders in the supply chain, making it difficult to achieve a seamless and integrated supply chain.
- **Legal and regulatory compliance** - it may require compliance with legal and regulatory requirements related to data privacy, security, and transparency. Companies must ensure that their implementation is compliant with all relevant regulations and standards.
- **Resistance to change** - it can be a significant change for employees and stakeholders in the supply chain, and resistance to change can be a significant challenge specially if a company chooses to adopt a document management system like the one implemented in this case study, where documents are stored on the cloud and associated through a reference on the blockchain. Companies must ensure that they have a robust change management plan in place to address this challenge.

In summary, implementing a blockchain-based traceability system can be a complex and challenging process, requiring careful consideration of the potential benefits and risks, as well as careful planning and execution. However, if implemented successfully, a blockchain-based traceability system can bring significant benefits to companies, including enhanced transparency, improved traceability, increased efficiency, reduced fraud and errors, and better customer satisfaction. To mitigate the pain points associated with implementing a blockchain-based traceability system, companies must develop a comprehensive plan that addresses the technical expertise required, data integration, cost, standardization, legal and regulatory compliance, and change management. With careful planning and execution, companies can successfully implement a blockchain-based traceability system and reap the benefits of enhanced supply chain transparency and efficiency.

5.2 Achievements

Following the goals defined in the Section 1.3, we can say that almost all goals were successfully achieved except for the *Experimental Validation*, this goal was focused on testing the system with real data from the company, that was not possible due to confidentiality patterns/rules and to get their evaluation about the system that was also not possible due to time incompatibility of the *Author*. Instead, some other tests were performed, all that developed on the Chapter 4.

Besides this, a scientific paper was published with the title *IoT and Blockchain Technologies for Process Traceability in the Shipbuilding Industry* on the Conference *Iberian Conference on Information Systems and Technologies (CISTI) – 2022* [3].

5.3 Future Work

Regarding some future development, there are already problems that were identified with possible solutions to them described in Section 4.2, these problems should be addressed by implementing one of the proposed solutions. By analyzing the Section 4.3, we can also say that the performance of the system should be improved on the communications between the general REST API and the Smart Contract API, this could easily be fixed by making the Smart Contract production ready and deploy it on a proper place.

The web and mobile applications can be improved or rethought upon the need but in a general way, these applications can make the essential processes/calls to the system in an easy and interactive way.

The final point to improve is the validation, this system should be tested with real data and real people to ensure the reliability.

References

[1] Akanksha and Akshay Chaturvedi. Comparison of different authentication techniques and steps to implement robust jwt authentication. In *2022 7th International Conference on Communication and Electronics Systems (ICCES)*, pages 772–779, 2022.

[2] Luís Alves, Estrela Ferreira Cruz, Sérgio I Lopes, Pedro M Faria, and António Miguel Rosado da Cruz. Towards circular economy in the textiles and clothing value chain through blockchain technology and iot: A review. *Waste Management & Research*, 40(1):3–23, 2022. PMID: 34708680.

[3] Pedro Araújo, A. M. Rosado da Cruz, and Sérgio Ivan Lopes. Iot and blockchain technologies for process traceability in the shipbuilding industry. In *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, 2022.

[4] Mohamed Awwad, Sohit Reddy, Varun Kazhana Airpulli, Madhubala Santosh Zambre, Aniket Marathe, and Prasham Jain. Blockchain technology for efficient management of supply chain. In *Proceedings of the International Conference on Industrial Engineering and Operations Management*, pages 1–10, Washington DC, USA, September 27-29 2018.

[5] Paul Bailes and Leighton Brough. Making sense of recursion patterns. In *2012 First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, pages 16–22, 2012.

[6] Prajakta Patil; Chiradeep BasuMallick. What is cloud computing? definition, benefits, types, and trends, February 2022. [Online; last updated February 9, 2022].

[7] Michail J. Beliatis, Kasper Jensen, Lars Ellegaard, Annabeth Aagaard, and Mirko Presser. Next generation industrial iot digitalization for traceability in metal manufacturing industry: A case study of industry 4.0. *Electronics*, 10(5), 2021.

[8] Thomas Beyhl, Gregor Berg, and Holger Giese. Why innovation processes need to support traceability. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 1–4, 2013.

[9] Soumya Chakraborty. Shipbuilding process : Finalising and launching the ship. https://www.marineinsight.com/naval-architecture/shipbuilding-process-finalising-the-ship/. Accessed: 2021-12-10.

[10] Heyder Coelho and Andre Araujo. Segurança na construção e reparos navais em estaleiro da região metropolitana de Belém. *Revista Mundi Engenharia, Tecnologia e Gestão (ISSN: 2525-4782)*, 5, 07 2020.

[11] Sylvere Krima; Thomas Hedberg; Allison Barnard Feeney. Nist: Blockchain provides security, traceability for smart manufacturing, February 2019. [Online; released February 11, 2019, updated October 25, 2021].

[12] Hyperledger Foundation. Fablo. `https://labs.hyperledger.org/labs/fablo.html`.

[13] The Apache Software Foundation. Apache jmeter. `https://jmeter.apache.org/`.

[14] Paula Fraga-Lamas, José Varela-Barbeito, and Tiago M. Fernández-Caramés. Next generation auto-identification and traceability technologies for industry 5.0: A methodology and practical use case for the shipbuilding industry. *IEEE Access*, 9:140700–140730, 2021.

[15] Iván Froiz-Míguez, Paula Fraga-Lamas, José Varela-Barbeito, and Tiago M. Fernández-Caramés. Lorawan and blockchain based safety and health monitoring system for industry 4.0 operators. *Proceedings*, 42(1), 2020.

[16] Ke Liu and Ke Xu. Oauth based authentication and authorization in open telco api. In *2012 International Conference on Computer Science and Electronics Engineering*, volume 1, pages 176–179, 2012.

[17] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. Cloud computing: An overview. In Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong, editors, *Cloud Computing*, pages 626–631, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[18] Pratik Rupareliya. What are decentralized applications (dapps)? — explained with examples, November 2018.

[19] Yahya Shahsavari, Kaiwen Zhang, and Chamseddine Talhi. Performance modeling and analysis of the bitcoin inventory protocol. 04 2019.

[20] West Sea Viana Shipyard. West sea viana shipyard. `https://west-sea.pt/`. Accessed: 2010-09-30.

[21] RxJS Team. Rxjs overview.

[22] Bureau Veritas. Requirements for survey of materials and equipment for the classification of ships and offshore units. `https://marine-offshore.bureauveritas.com/nr266-requirements-survey-materials-and-equipment-classification-ships-and-`
Accessed: 2010-09-30.

[23] Shuhai Wang, Jingjing Shi, Dong Jiang, and Zhaohui Qi. Research for traceability model of material supply quality in construction project. In *2012 Fifth International Symposium on Computational Intelligence and Design*, volume 2, pages 398–401, Oct 2012.

[24] Wei Zhou, Li Li, Min Luo, and Wu Chou. Rest api design patterns for sdn northbound api. In *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pages 358–365, 2014.

Appendix A

JSON examples

In this appendix, it is stored some input/output examples in JSON format.

A.1 The “Activity“ property “inputProductLots“

```
{
    "8d2098a2-c9eb-4f51-b125-f51f137588a3": 13,
    "a3ce92a6-8514-4d6f-824b-6e95d0181e17" : 1,
}
```

A.2 The return of the Smart Contract operation “GetTraceability-ByReferenceNum“

```
{
    "ID": "71b98a2f-ce99-4176-95fc-0ebdc5f51436",
    "referenceNumber": "1231564",
    "isSerialNumber" : false ,
    "designation" : "Steel Plate",
    "productType" : "Part",
    "initialQuantity" : 583,
    "availableQuantity" : 175,
    "usedQuantityAsInput" : null ,
    "documentKeys" : [
        {
            "documentKey": "key132",
            "fileFingerPrint": "238168267g1s671gs1f2t3gj1"
        },
        {
            "documentKey": "key615",
            "fileFingerPrint": "655q8wf46q815w7d16q5w1"
        }
    ],
    "activity" : {
        "ID": "e8e809cb-208b-4113-bc31-b451011e2013"
        "designation": "Cut"
        "userId": "8877c40e-4e1f-49c1-8c52-937718320f9f"
        "date": "2022-05-01T19:07:51.216946"
        "inputProductLots": [
            {
                "ID": "bf024971-12d8-4d2c-92f7-cd297b7b711c",
                "referenceNumber": "569653",
                "isSerialNumber" : false ,
                "designation" : "Steel Ingot",
                "productType" : "Part",
                "initialQuantity" : 7512,
                "availableQuantity" : 5168,
                "usedQuantityAsInput" : 2175,
                "documentKeys" : [
                    {
                        "documentKey": "key168",
                        "fileFingerPrint": "6w4re89wd816d4wa58"
                    }
                ],
                "activity" : null
            }
        ]
    }
}
```

Appendix B

GitHub

In this appendix it can be found the GitHub repositories with all the code of this system.

B.1 General REST API

https://github.com/pedroei/westsea-traceability

B.2 Smart Contract

https://github.com/pedroei/westsea-trace-fabric

B.3 Web Application

https://github.com/pedroei/westsea-traceability-web

B.4 Mobile Application

https://github.com/pedroei/-westsea-traceability-mobile

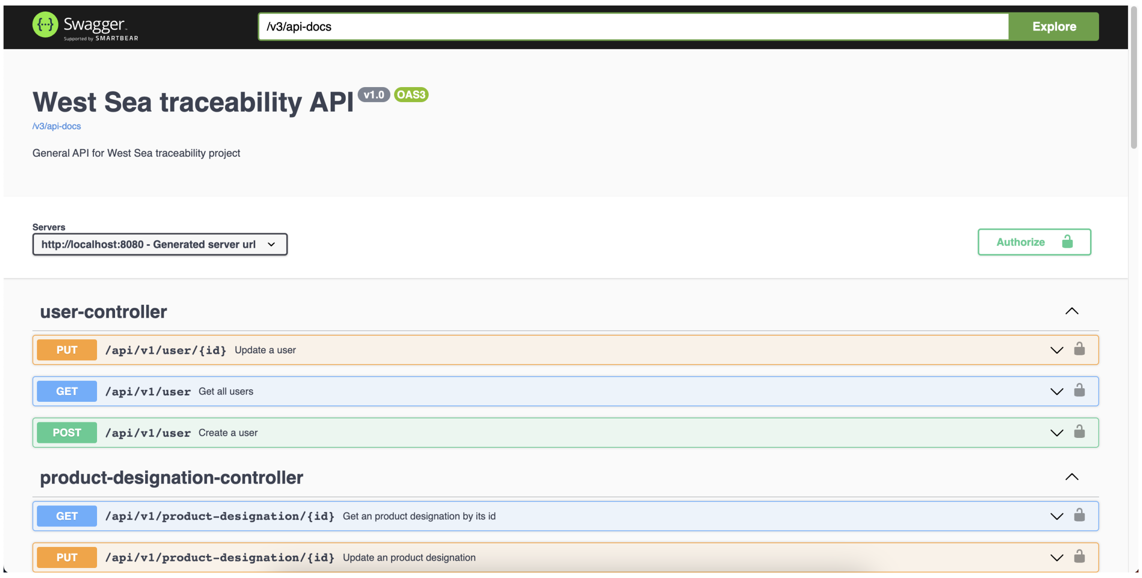
Appendix C

Printscreens

In this appendix, some relevant print screens can be found to be used as examples

C.1 Swagger (Open API) implementation

Figures C.1, C.2, C.3 and C.4 show the swagger interface.



West Sea traceability API v1.0 OAS3

/v3/api-docs

General API for West Sea traceability project

Servers

http://localhost:8080 - Generated server url

user-controller

PUT `/api/v1/user/{id}` Update a user

GET `/api/v1/user` Get all users

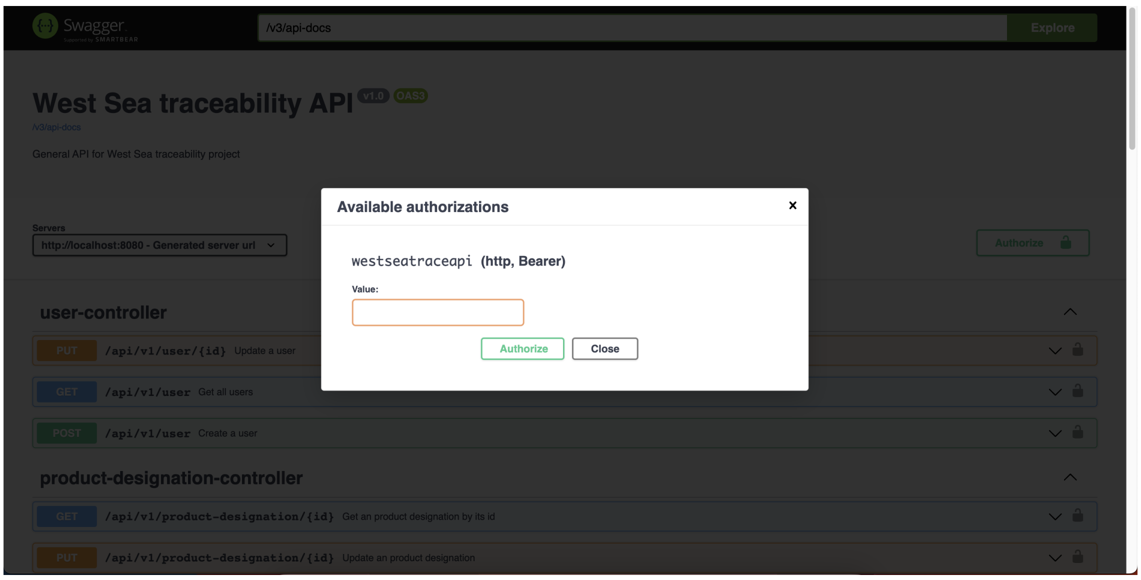
POST `/api/v1/user` Create a user

product-designation-controller

GET `/api/v1/product-designation/{id}` Get an product designation by its id

PUT `/api/v1/product-designation/{id}` Update an product designation

Figure C.1: Swagger UI



/v3/api-docs

West Sea traceability API v1.0 OAS3

/v3/api-docs

General API for West Sea traceability project

Servers
http://localhost:8080 - Generated server url

Available authorizations

`westseatraceapi` **(http, Bearer)**

Value:

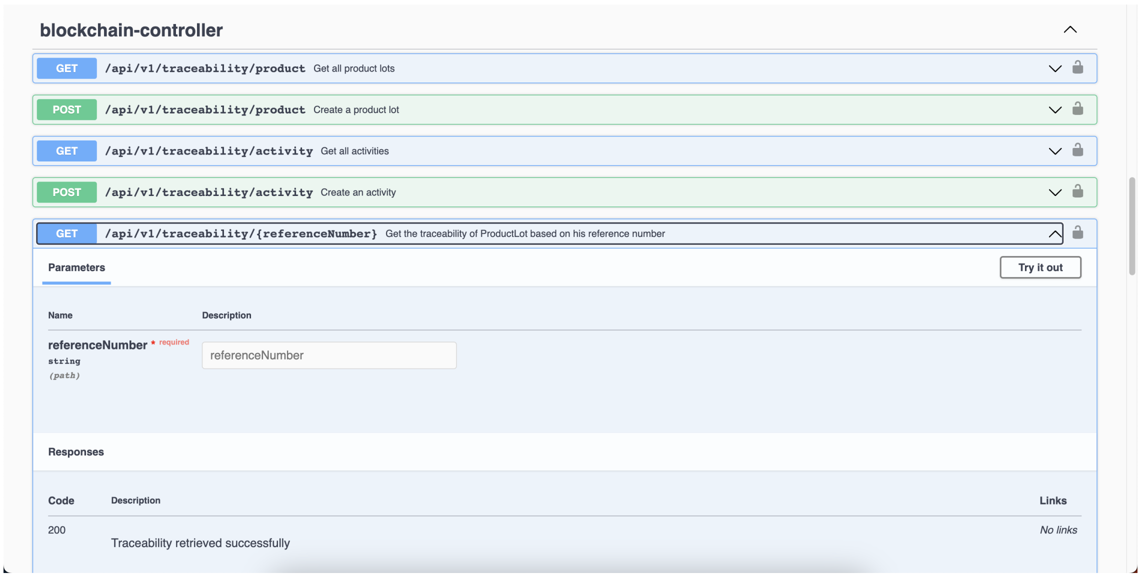
user-controller

- PUT `/api/v1/user/{id}` Update a user
- GET `/api/v1/user` Get all users
- POST `/api/v1/user` Create a user

product-designation-controller

- GET `/api/v1/product-designation/{id}` Get an product designation by its id
- PUT `/api/v1/product-designation/{id}` Update an product designation

Figure C.2: Swagger authentications with bearer token



blockchain-controller

- GET `/api/v1/traceability/product` Get all product lots
- POST `/api/v1/traceability/product` Create a product lot
- GET `/api/v1/traceability/activity` Get all activities
- POST `/api/v1/traceability/activity` Create an activity
- GET `/api/v1/traceability/{referenceNumber}` Get the traceability of ProductLot based on his reference number

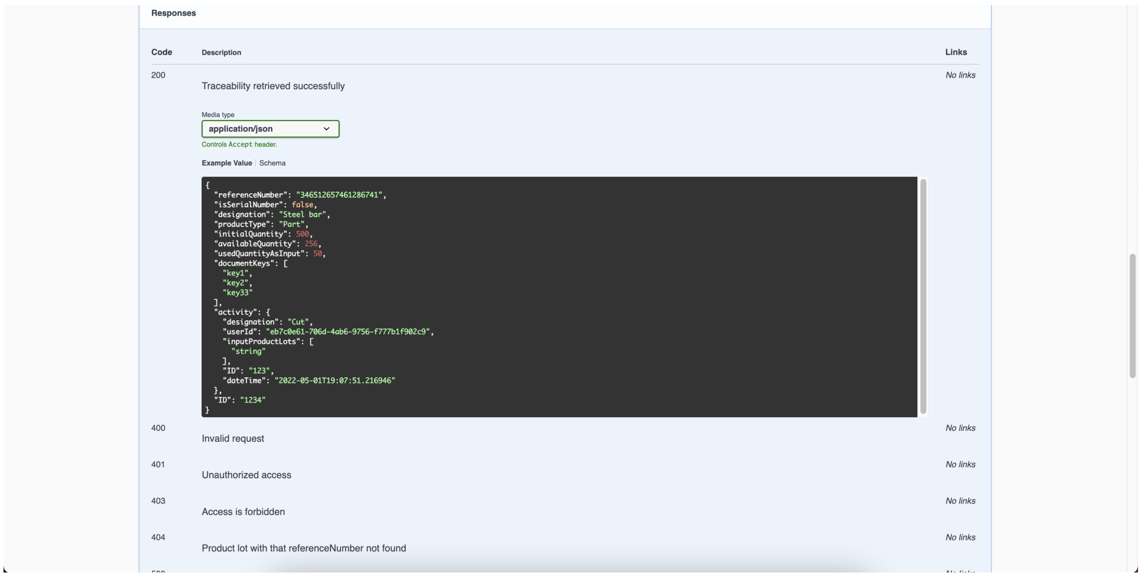
Parameters

Name	Description
referenceNumber * required string *(path)*	referenceNumber

Responses

Code	Description	Links
200	Traceability retrieved successfully	*No links*

Figure C.3: Swagger example request



Responses

Code	Description	Links
200	Traceability retrieved successfully	*No links*
400	Invalid request	*No links*
401	Unauthorized access	*No links*
403	Access is forbidden	*No links*
404	Product lot with that referenceNumber not found	*No links*

Media type: application/json

Controls Accept header.

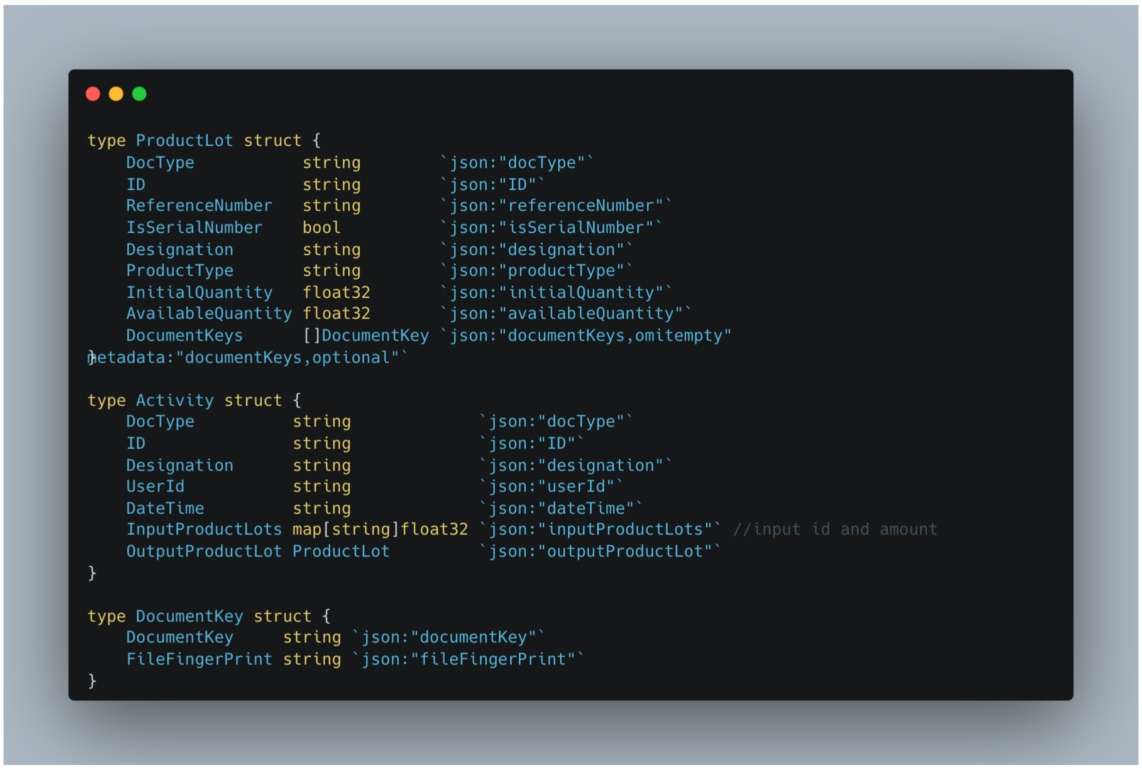
Example Value | Schema

```
{
  "referenceNumber": "346512657461286741",
  "isSerialNumber": false,
  "designation": "Steel bar",
  "productType": "Part",
  "initialQuantity": 500,
  "availableQuantity": 256,
  "usedQuantityAsInput": 50,
  "documentKeys": [
    "key1",
    "key2",
    "key33"
  ],
  "activity": {
    "designation": "Cut",
    "userId": "eb7c0e61-706d-4ab6-9756-f777b1f902c9",
    "inputProductLots": [
      "string"
    ],
    "ID": "123",
    "dateTime": "2022-05-01T19:07:51.216946"
  },
  "ID": "1234"
}
```

Figure C.4: Swagger example response

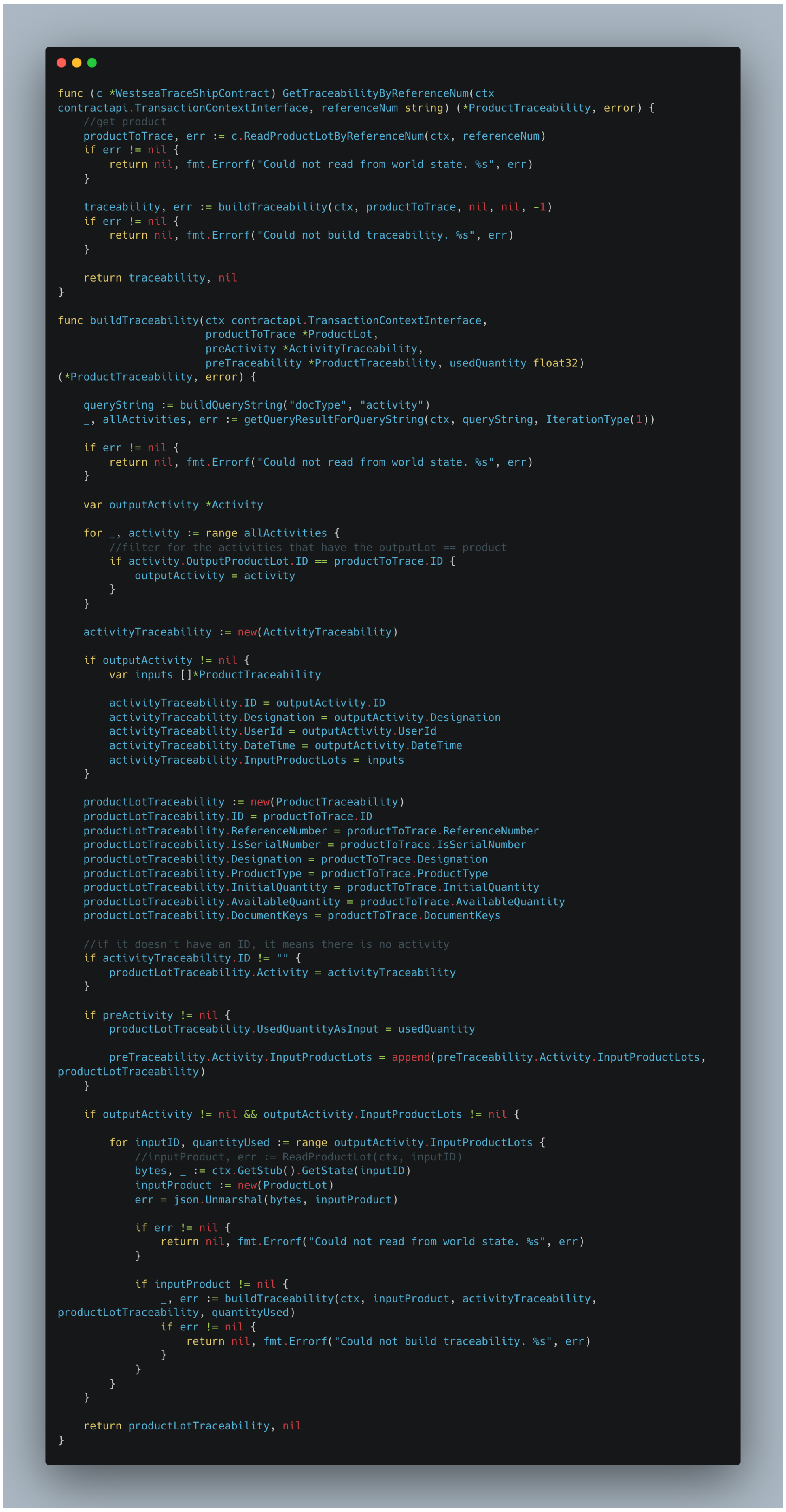
C.2 Code examples

Figures C.5 and C.6 show code examples.



```
type ProductLot struct {
    DocType           string        `json:"docType"`
    ID                string        `json:"ID"`
    ReferenceNumber   string        `json:"referenceNumber"`
    IsSerialNumber    bool          `json:"isSerialNumber"`
    Designation       string        `json:"designation"`
    ProductType       string        `json:"productType"`
    InitialQuantity   float32       `json:"initialQuantity"`
    AvailableQuantity float32       `json:"availableQuantity"`
    DocumentKeys      []DocumentKey `json:"documentKeys,omitempty"
}etadata:"documentKeys,optional"`

type Activity struct {
    DocType          string             `json:"docType"`
    ID               string             `json:"ID"`
    Designation      string             `json:"designation"`
    UserId           string             `json:"userId"`
    DateTime         string             `json:"dateTime"`
    InputProductLots map[string]float32 `json:"inputProductLots"` //input id and amount
    OutputProductLot ProductLot         `json:"outputProductLot"`
}

type DocumentKey struct {
    DocumentKey     string `json:"documentKey"`
    FileFingerPrint string `json:"fileFingerPrint"`
}
```

Figure C.5: Smart Contract Entities

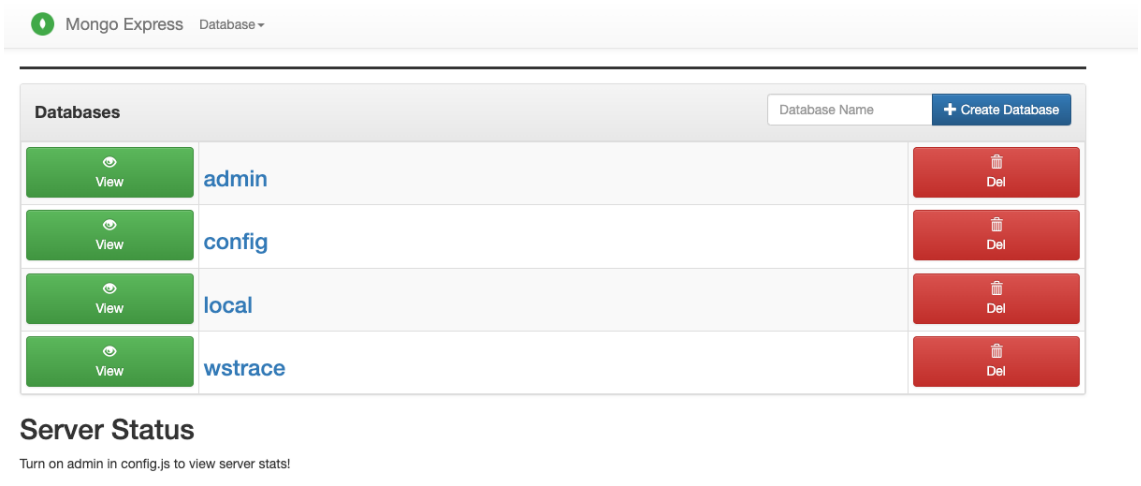


```go
func (c *WestseaTraceShipContract) GetTraceabilityByReferenceNum(ctx
contractapi.TransactionContextInterface, referenceNum string) (*ProductTraceability, error) {
    //get product
    productToTrace, err := c.ReadProductLotByReferenceNum(ctx, referenceNum)
    if err != nil {
        return nil, fmt.Errorf("Could not read from world state. %s", err)
    }

    traceability, err := buildTraceability(ctx, productToTrace, nil, nil, -1)
    if err != nil {
        return nil, fmt.Errorf("Could not build traceability. %s", err)
    }

    return traceability, nil
}

func buildTraceability(ctx contractapi.TransactionContextInterface,
                       productToTrace *ProductLot,
                       preActivity *ActivityTraceability,
                       preTraceability *ProductTraceability, usedQuantity float32)
(*ProductTraceability, error) {

    queryString := buildQueryString("docType", "activity")
    _, allActivities, err := getQueryResultForQueryString(ctx, queryString, IterationType(1))

    if err != nil {
        return nil, fmt.Errorf("Could not read from world state. %s", err)
    }

    var outputActivity *Activity

    for _, activity := range allActivities {
        //filter for the activities that have the outputLot == product
        if activity.OutputProductLot.ID == productToTrace.ID {
            outputActivity = activity
        }
    }

    activityTraceability := new(ActivityTraceability)

    if outputActivity != nil {
        var inputs []*ProductTraceability

        activityTraceability.ID = outputActivity.ID
        activityTraceability.Designation = outputActivity.Designation
        activityTraceability.UserId = outputActivity.UserId
        activityTraceability.DateTime = outputActivity.DateTime
        activityTraceability.InputProductLots = inputs
    }

    productLotTraceability := new(ProductTraceability)
    productLotTraceability.ID = productToTrace.ID
    productLotTraceability.ReferenceNumber = productToTrace.ReferenceNumber
    productLotTraceability.IsSerialNumber = productToTrace.IsSerialNumber
    productLotTraceability.Designation = productToTrace.Designation
    productLotTraceability.ProductType = productToTrace.ProductType
    productLotTraceability.InitialQuantity = productToTrace.InitialQuantity
    productLotTraceability.AvailableQuantity = productToTrace.AvailableQuantity
    productLotTraceability.DocumentKeys = productToTrace.DocumentKeys

    //if it doesn't have an ID, it means there is no activity
    if activityTraceability.ID != "" {
        productLotTraceability.Activity = activityTraceability
    }

    if preActivity != nil {
        productLotTraceability.UsedQuantityAsInput = usedQuantity

        preTraceability.Activity.InputProductLots = append(preTraceability.Activity.InputProductLots,
productLotTraceability)
    }

    if outputActivity != nil && outputActivity.InputProductLots != nil {

        for inputID, quantityUsed := range outputActivity.InputProductLots {
            //inputProduct, err := ReadProductLot(ctx, inputID)
            bytes, _ := ctx.GetStub().GetState(inputID)
            inputProduct := new(ProductLot)
            err = json.Unmarshal(bytes, inputProduct)

            if err != nil {
                return nil, fmt.Errorf("Could not read from world state. %s", err)
            }

            if inputProduct != nil {
                _, err := buildTraceability(ctx, inputProduct, activityTraceability,
productLotTraceability, quantityUsed)
                if err != nil {
                    return nil, fmt.Errorf("Could not build traceability. %s", err)
                }
            }
        }
    }

    return productLotTraceability, nil
}
```

Figure C.6: Smart Contract function to build the traceability of a ProductLot

C.3 Mongo Express Service

Figures C.7, C.8 and C.9 show the Mongo Express interface.



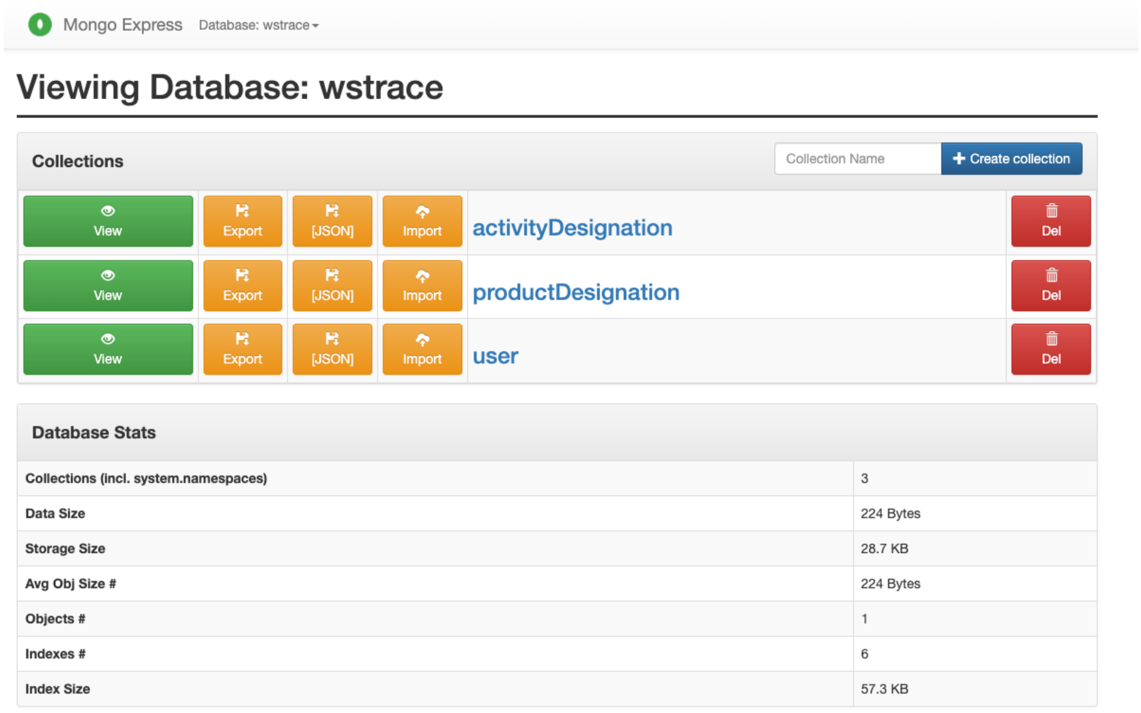
Databases

View	admin	Del
View	config	Del
View	local	Del
View	wstrace	Del

Server Status

Turn on admin in config.js to view server stats!

Figure C.7: Mongo Express



Viewing Database: wstrace

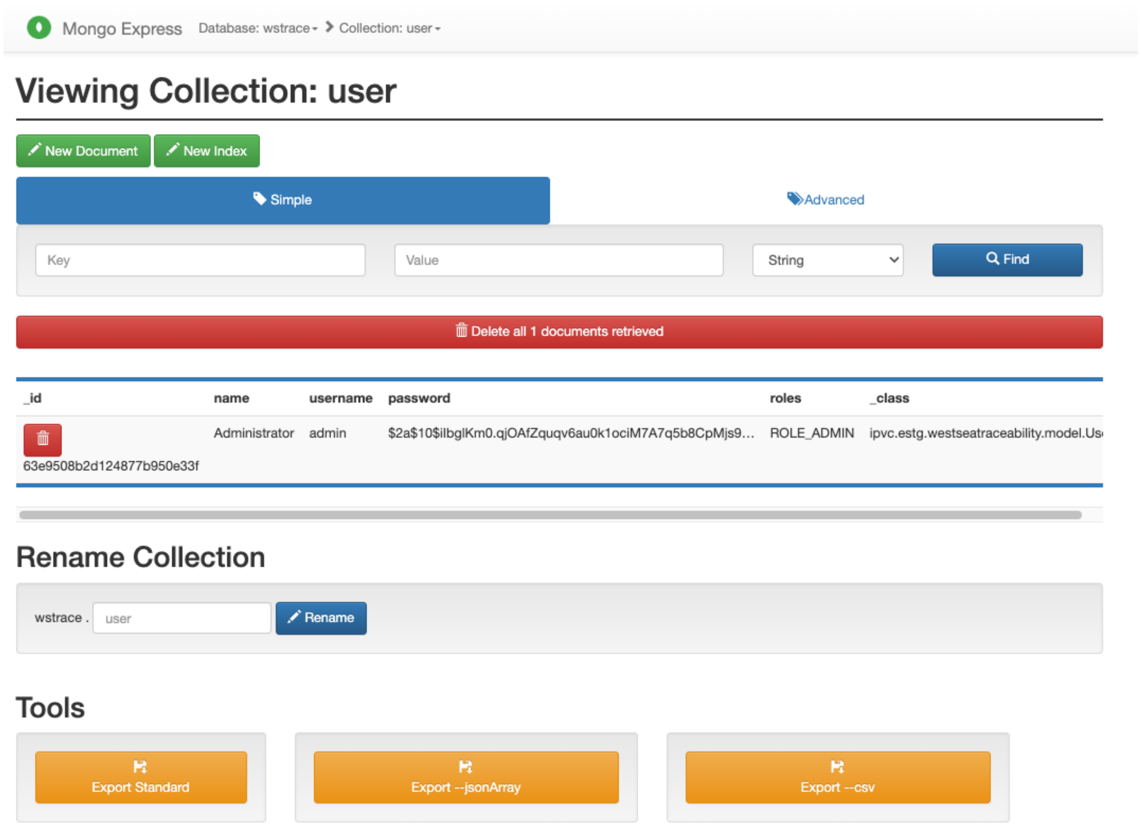
Collections

View	Export	[JSON]	Import	activityDesignation	Del
View	Export	[JSON]	Import	productDesignation	Del
View	Export	[JSON]	Import	user	Del

Database Stats

Collections (incl. system.namespaces)	3
Data Size	224 Bytes
Storage Size	28.7 KB
Avg Obj Size #	224 Bytes
Objects #	1
Indexes #	6
Index Size	57.3 KB

Figure C.8: Collections on Mongo Express



Viewing Collection: user

_id	name	username	password	roles	_class
63e9508b2d124877b950e33f	Administrator	admin	$2a$10$ilbglKm0.qjOAfZquqv6au0k1ociM7A7q5b8CpMjs9...	ROLE_ADMIN	ipvc.estg.westseatraceability.model.Us

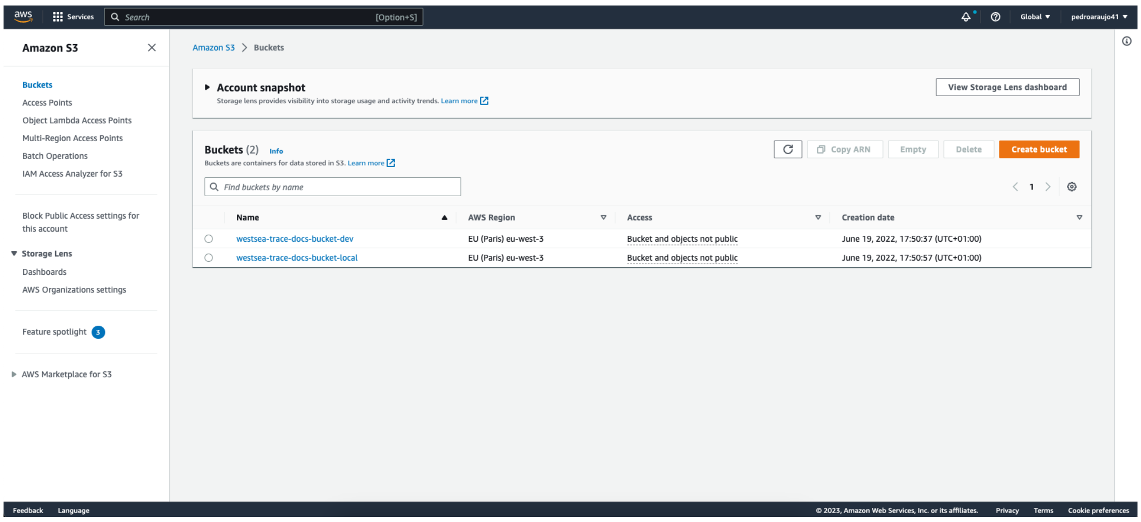
Rename Collection

Tools

Figure C.9: Documents on Mongo Express

C.4 AWS S3

Figures C.10, C.11, C.12 and C.13 show the interface of S3 on AWS.



Account snapshot

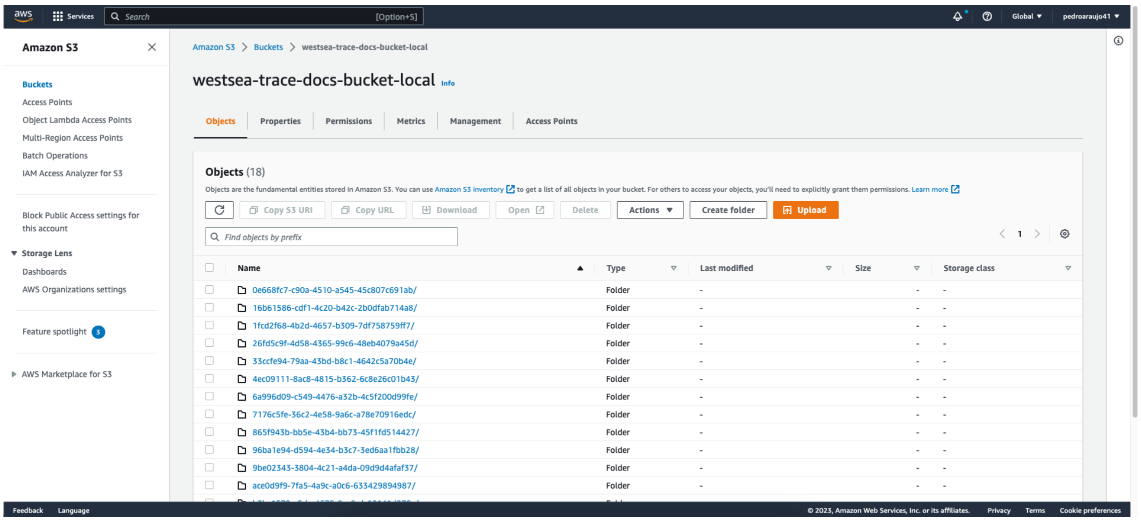
Storage lens provides visibility into storage usage and activity trends. Learn more

Buckets (2)

Buckets are containers for data stored in S3. Learn more

Name	AWS Region	Access	Creation date
westsea-trace-docs-bucket-dev	EU (Paris) eu-west-3	Bucket and objects not public	June 19, 2022, 17:50:37 (UTC+01:00)
westsea-trace-docs-bucket-local	EU (Paris) eu-west-3	Bucket and objects not public	June 19, 2022, 17:50:57 (UTC+01:00)

Figure C.10: Buckets on S3



Amazon S3 > Buckets > westsea-trace-docs-bucket-local

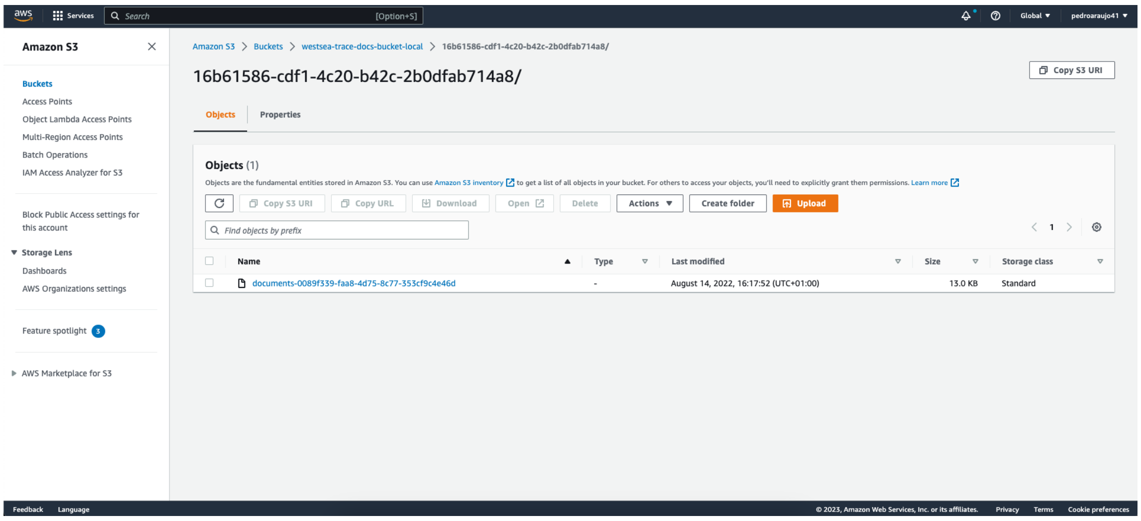
westsea-trace-docs-bucket-local Info

Objects (18)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
0e668fc7-c90a-4510-a545-45c807c691ab/	Folder	-	-	-
16b61586-cdf1-4c20-b42c-2b0dfab714a8/	Folder	-	-	-
1fcd2f68-4b2d-4657-b309-7df758759ff7/	Folder	-	-	-
26fd5c9f-4d58-4365-99c6-48eb4079a45d/	Folder	-	-	-
33ccfe94-79aa-43bd-b8c1-4642c5a70b4e/	Folder	-	-	-
4ec09111-8ac8-4815-b362-6c8e26c01b43/	Folder	-	-	-
6a996d09-c549-4476-a32b-4c5f200d99fe/	Folder	-	-	-
7176c5fe-36c2-4e58-9a6c-a78e70916edc/	Folder	-	-	-
865f943b-bb5e-43b4-bb73-45f1fd514427/	Folder	-	-	-
96ba1e94-d594-4e34-b3c7-3ed6aa1fbb28/	Folder	-	-	-
9be02343-3804-4c21-a4da-09d9d4afaf37/	Folder	-	-	-
ace0d9f9-7fa5-4a9c-a0c6-633429894987/	Folder	-	-	-

Figure C.11: Folders on S3



Amazon S3 > Buckets > westsea-trace-docs-bucket-local > 16b61586-cdf1-4c20-b42c-2b0dfab714a8/

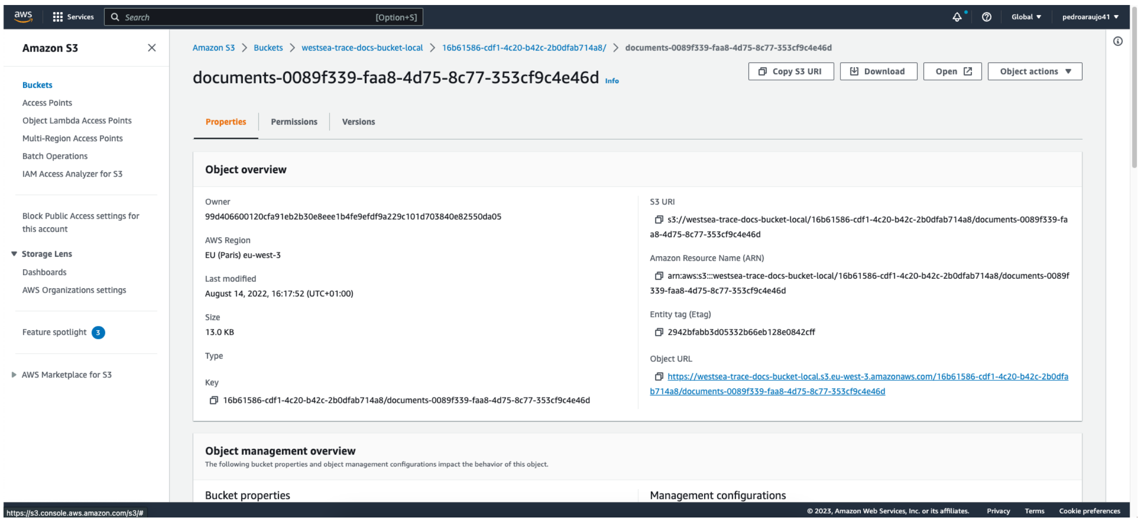
16b61586-cdf1-4c20-b42c-2b0dfab714a8/

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
documents-0089f339-faa8-4d75-8c77-353cf9c4e46d	-	August 14, 2022, 16:17:52 (UTC+01:00)	13.0 KB	Standard

Figure C.12: Documents on S3



Amazon S3 > Buckets > westsea-trace-docs-bucket-local > 16b61586-cdf1-4c20-b42c-2b0dfab714a8/ > documents-0089f339-faa8-4d75-8c77-353cf9c4e46d

documents-0089f339-faa8-4d75-8c77-353cf9c4e46d Info

Object overview

Owner
99d406600120cfa91eb2b30e8eee1b4fe9efdf9a229c101d703840e82550da05

AWS Region
EU (Paris) eu-west-3

Last modified
August 14, 2022, 16:17:52 (UTC+01:00)

Size
13.0 KB

Type

Key
16b61586-cdf1-4c20-b42c-2b0dfab714a8/documents-0089f339-faa8-4d75-8c77-353cf9c4e46d

S3 URI
s3://westsea-trace-docs-bucket-local/16b61586-cdf1-4c20-b42c-2b0dfab714a8/documents-0089f339-faa8-4d75-8c77-353cf9c4e46d

Amazon Resource Name (ARN)
arn:aws:s3:::westsea-trace-docs-bucket-local/16b61586-cdf1-4c20-b42c-2b0dfab714a8/documents-0089f339-faa8-4d75-8c77-353cf9c4e46d

Entity tag (Etag)
2942bfabb3d05332b66eb128e0842cff

Object URL
https://westsea-trace-docs-bucket-local.s3.eu-west-3.amazonaws.com/16b61586-cdf1-4c20-b42c-2b0dfab714a8/documents-0089f339-faa8-4d75-8c77-353cf9c4e46d

Object management overview
The following bucket properties and object management configurations impact the behavior of this object.

Bucket properties

Management configurations

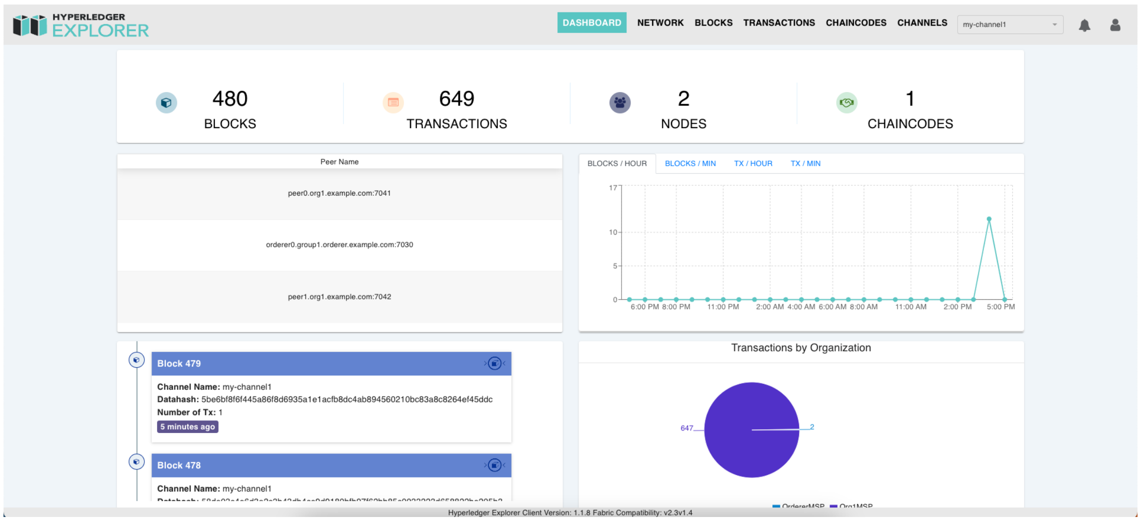
https://s3.console.aws.amazon.com/s3/#

© 2023, Amazon Web Services, Inc. or its affiliates.

Figure C.13: Document Information on S3

C.5 Blockchain Explorer

Figures C.14, C.15 and C.16 show the Blockchain Explorer.



480 BLOCKS

649 TRANSACTIONS

2 NODES

1 CHAINCODES

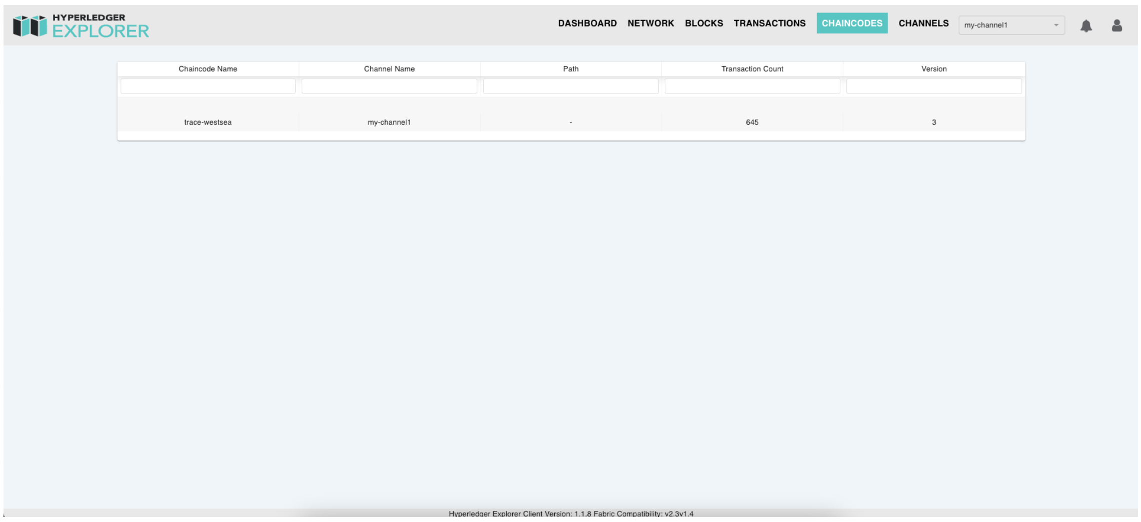
Peer Name
peer0.org1.example.com:7041
orderer0.group1.orderer.example.com:7030
peer1.org1.example.com:7042

Block 479
Channel Name: my-channel1
Datahash: 5be6bf8f6f445a86f8d6935a1e1acfb8dc4ab894560210bc83a8c8264ef45ddc
Number of Tx: 1
5 minutes ago

Block 478
Channel Name: my-channel1

Hyperledger Explorer Client Version: 1.1.8 Fabric Compatibility: v2.3v1.4

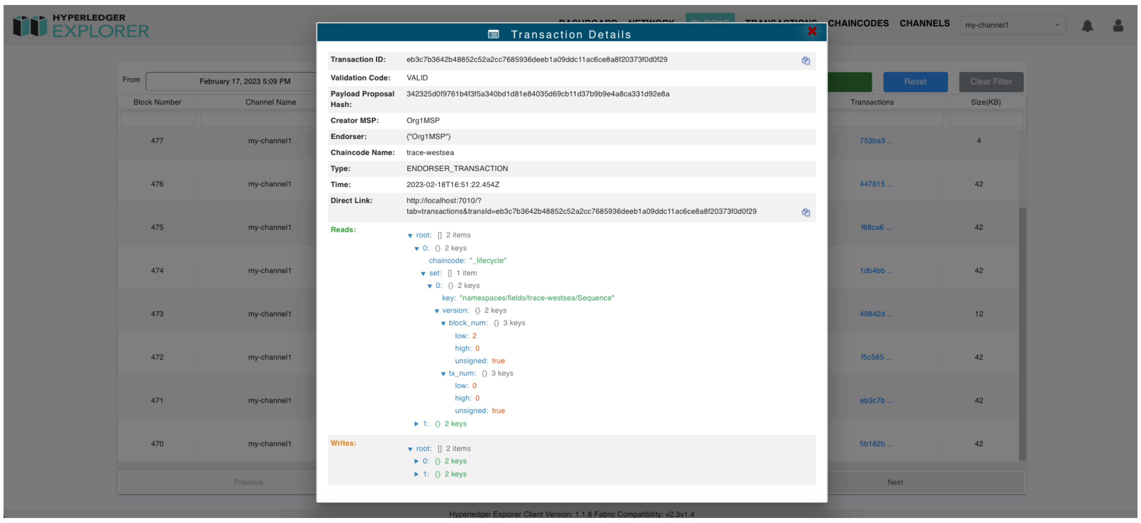
Figure C.14: Blockchain Explorer Dashboard



Chaincode Name	Channel Name	Path	Transaction Count	Version
trace-westsea	my-channel1	-	645	3

Hyperledger Explorer Client Version: 1.1.8 Fabric Compatibility: v2.3v1.4

Figure C.15: Blockchain Explorer Chain codes



Transaction Details

Transaction ID:	eb3c7b3642b48852c52a2cc7685936deeb1a09ddc11ac6ce8a8f20373f0d0f29
Validation Code:	VALID
Payload Proposal Hash:	342325d0f9761b4f3f5a340bd1d81e84035d69cb11d37b9b9e4a8ca331d92e8a
Creator MSP:	Org1MSP
Endorser:	{"Org1MSP"}
Chaincode Name:	trace-westsea
Type:	ENDORSER_TRANSACTION
Time:	2023-02-18T16:51:22.454Z
Direct Link:	http://localhost:7010/?tab=transactions&transId=eb3c7b3642b48852c52a2cc7685936deeb1a09ddc11ac6ce8a8f20373f0d0f29

Reads:

```
▼ root: [] 2 items
  ▼ 0: {} 2 keys
      chaincode: "_lifecycle"
    ▼ set: [] 1 item
      ▼ 0: {} 2 keys
          key: "namespaces/fields/trace-westsea/Sequence"
        ▼ version: {} 2 keys
          ▼ block_num: {} 3 keys
              low: 2
              high: 0
              unsigned: true
          ▼ tx_num: {} 3 keys
              low: 0
              high: 0
              unsigned: true
  ▶ 1: {} 2 keys
```

Writes:

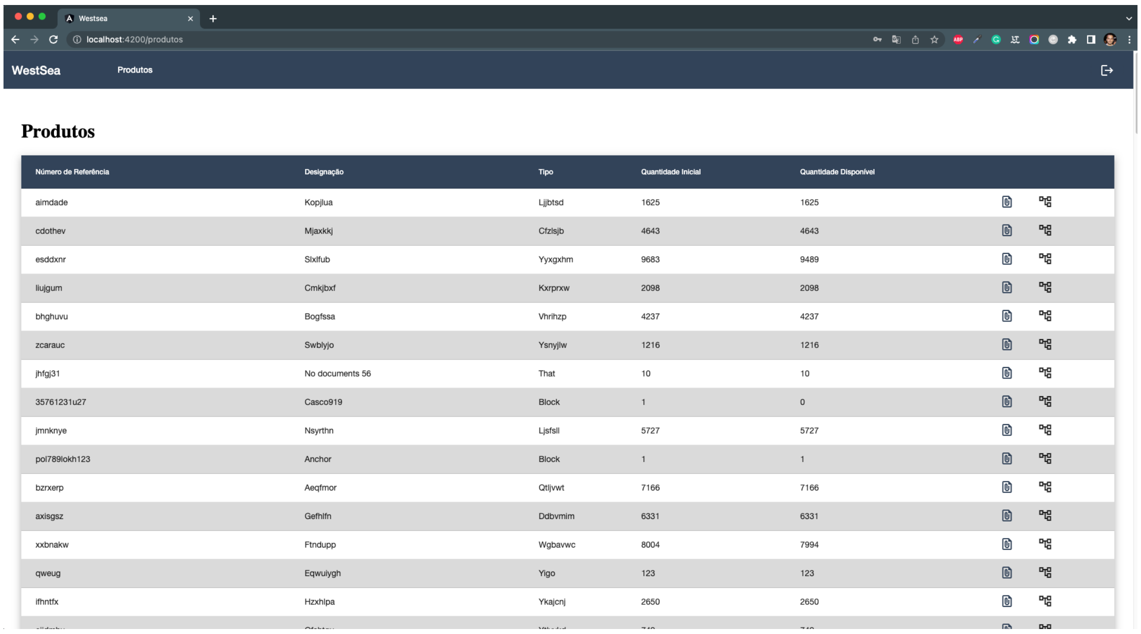
```
▼ root: [] 2 items
  ▶ 0: {} 2 keys
  ▶ 1: {} 2 keys
```

Hyperledger Explorer Client Version: 1.1.8 Fabric Compatibility: v2.3v1.4

Figure C.16: Blockchain Explorer Transaction details

C.6 Web Application

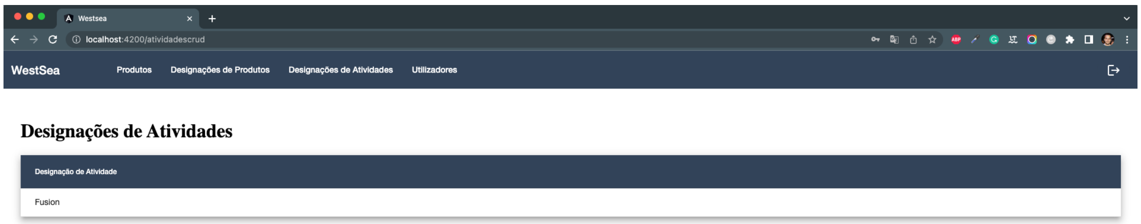
Figures C.17, C.18 and C.19 show the different views for different users on the web application.



Produtos

Número de Referência	Designação	Tipo	Quantidade Inicial	Quantidade Disponível
aimdade	Kopjlua	Ljjbtsd	1625	1625
cdothev	Mjaxkkj	Cfzlsjb	4643	4643
esddxnr	Slxlfub	Yyxgxhm	9683	9489
liujgum	Cmkjbxf	Kxrprxw	2098	2098
bhghuvu	Bogfssa	Vhrihzp	4237	4237
zcarauc	Swblyjo	Ysnyjlw	1216	1216
jhfgj31	No documents 56	That	10	10
35761231u27	Casco919	Block	1	0
jmnknye	Nsyrthn	Ljsfsll	5727	5727
pol789lokh123	Anchor	Block	1	1
bzrxerp	Aeqfmor	Qtljvwt	7166	7166
axisgsz	Gefhlfn	Ddbvmim	6331	6331
xxbnakw	Ftndupp	Wgbavwc	8004	7994
qweug	Eqwuiygh	Yigo	123	123
ifhntfx	Hzxhlpa	Ykajcnj	2650	2650

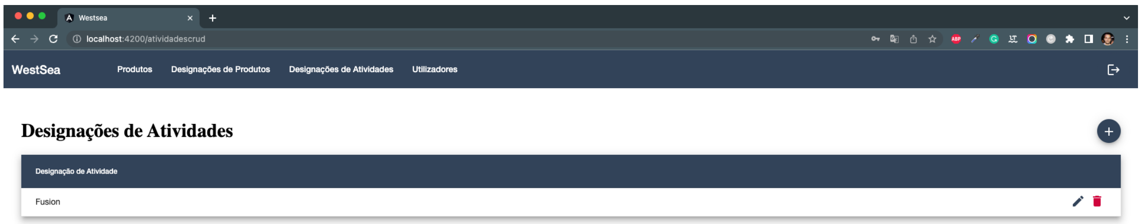
Figure C.17: Client View



Designações de Atividades

Designação de Atividade
Fusion

Figure C.18: Employee View



Designações de Atividades

Designação de Atividade
Fusion

Figure C.19: Admin View

C.7 Web Application before refactoring

Figures C.20, C.21, C.22, C.23, C.24 and C.25 show the old views of the web application before the refactoring.

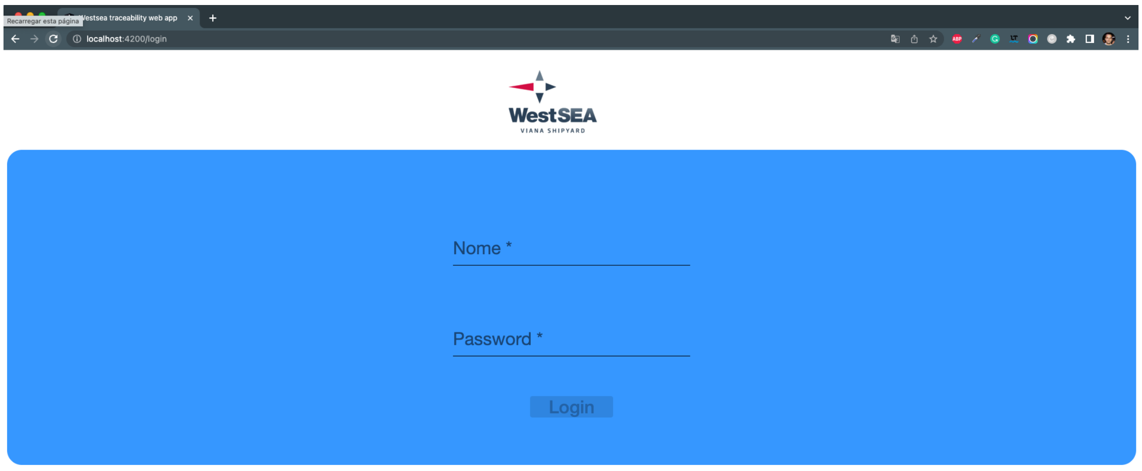


Figure C.20: Old web login page

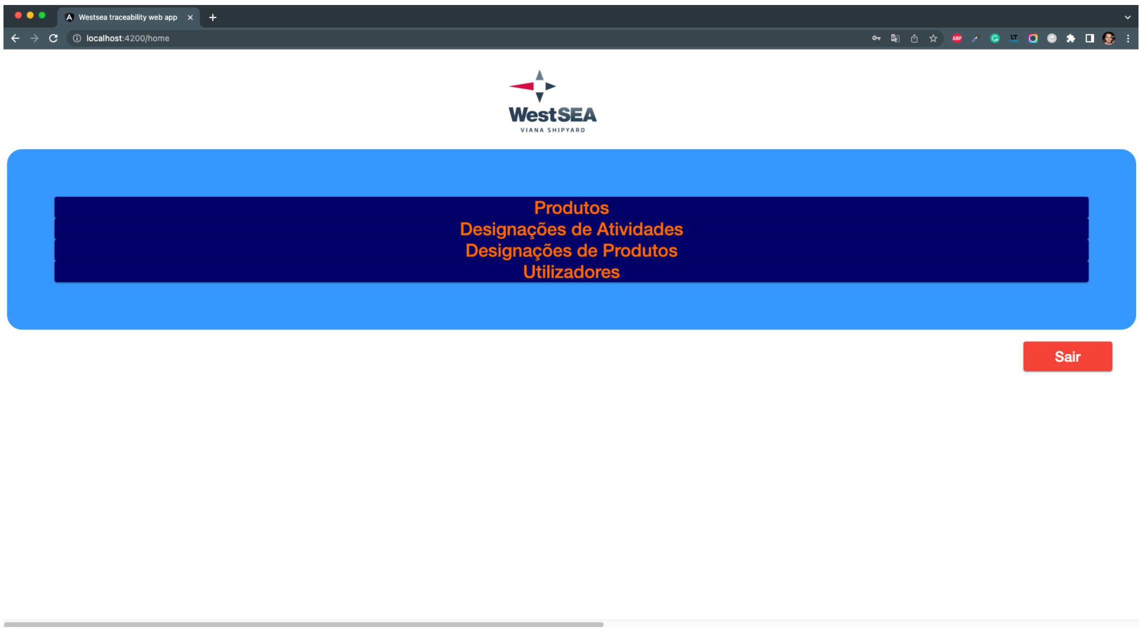


Figure C.21: Old web menu

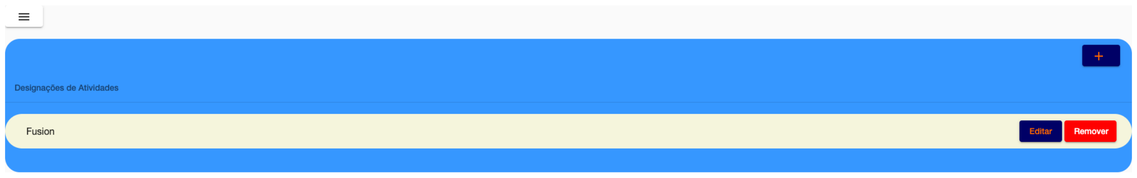


Figure C.22: Old web activity designations page

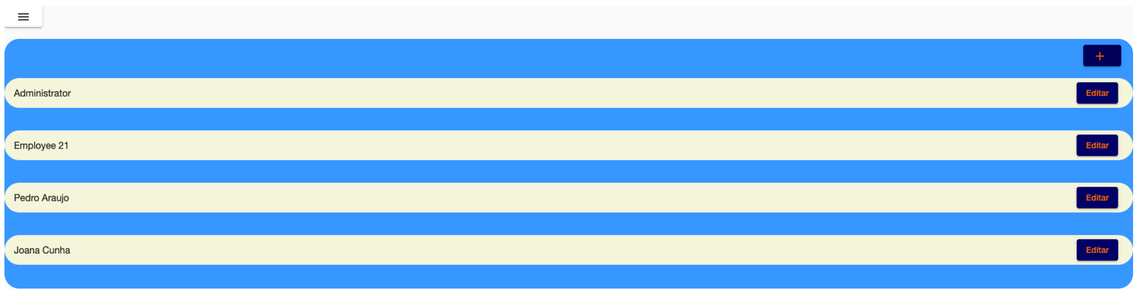
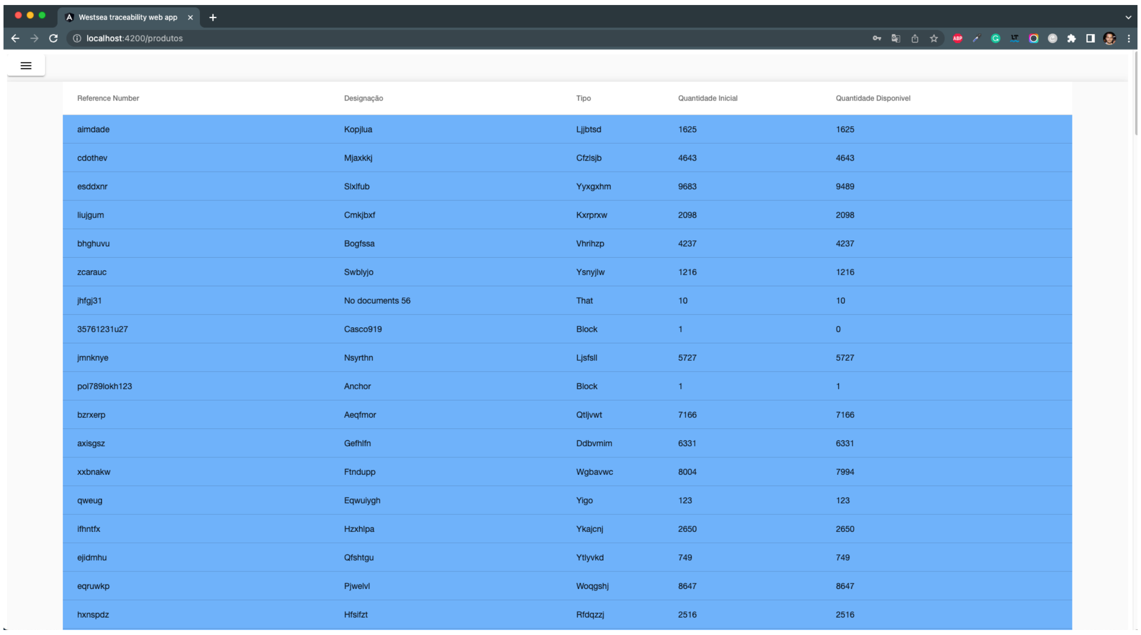
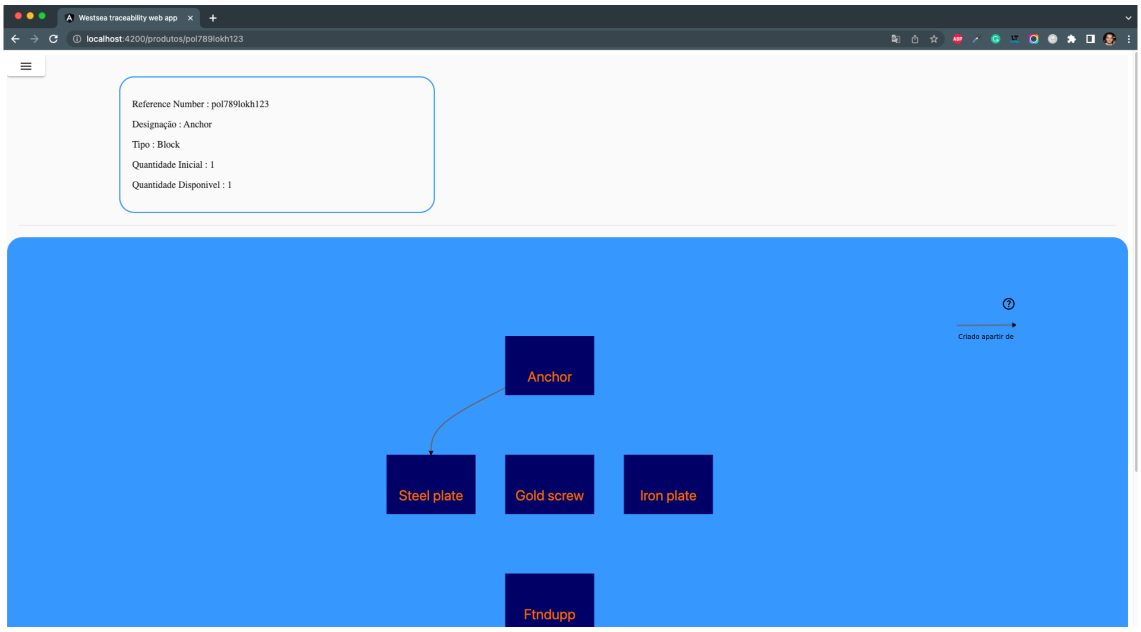


Figure C.23: Old web users page



Reference Number	Designação	Tipo	Quantidade Inicial	Quantidade Disponivel
aimdade	Kopjlua	Ljjbtsd	1625	1625
cdothev	Mjaxkkj	Cfzlsjb	4643	4643
esddxnr	Slxlfub	Yyxgxhm	9683	9489
liujgum	Cmkjbxf	Kxrprxw	2098	2098
bhghuvu	Bogfssa	Vhrihzp	4237	4237
zcarauc	Swblyjo	Ysnyjlw	1216	1216
jhfgj31	No documents 56	That	10	10
35761231u27	Casco919	Block	1	0
jmnknye	Nsyrthn	Ljsfsll	5727	5727
pol789lokh123	Anchor	Block	1	1
bzrxerp	Aeqfmor	Qtljvwt	7166	7166
axisgsz	Gefhlfn	Ddbvmim	6331	6331
xxbnakw	Ftndupp	Wgbavwc	8004	7994
qweug	Eqwuiygh	Yigo	123	123
ifhntfx	Hzxhlpa	Ykajcnj	2650	2650
ejidmhu	Qfshtgu	Ytlyvkd	749	749
eqruwkp	Pjwelvl	Woqgshj	8647	8647
hxnspdz	Hfsifzt	Rfdqzzj	2516	2516

Figure C.24: Old web products page


Westsea traceability web app
localhost:4200/produtos/pol789lokh123

Reference Number : pol789lokh123
Designação : Anchor
Tipo : Block
Quantidade Inicial : 1
Quantidade Disponivel : 1

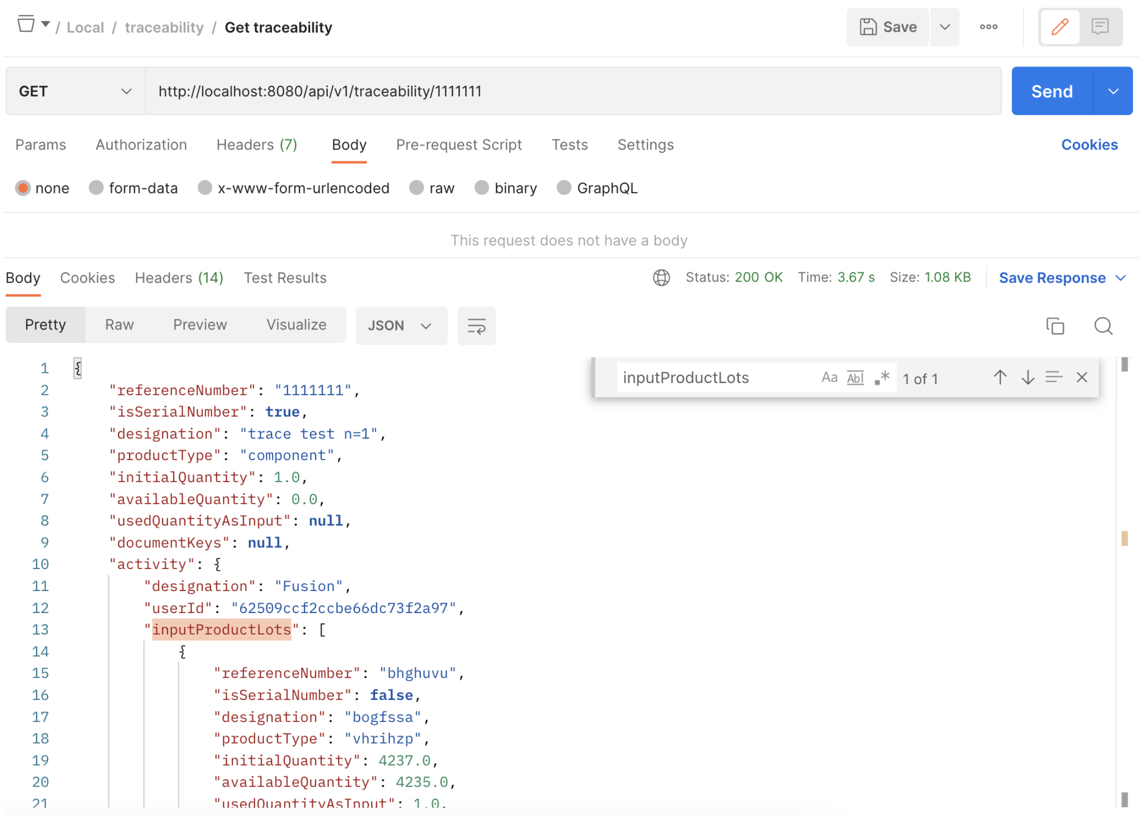
Criado apartir de
Anchor
Steel plate
Gold screw
Iron plate
Ftndupp

Figure C.25: Old web traceability page

C.8 Performance Tests with JMeter

Figures from Sections C.8.1, C.8.2 and C.8.3 show the tests on the recursive call with different depths.

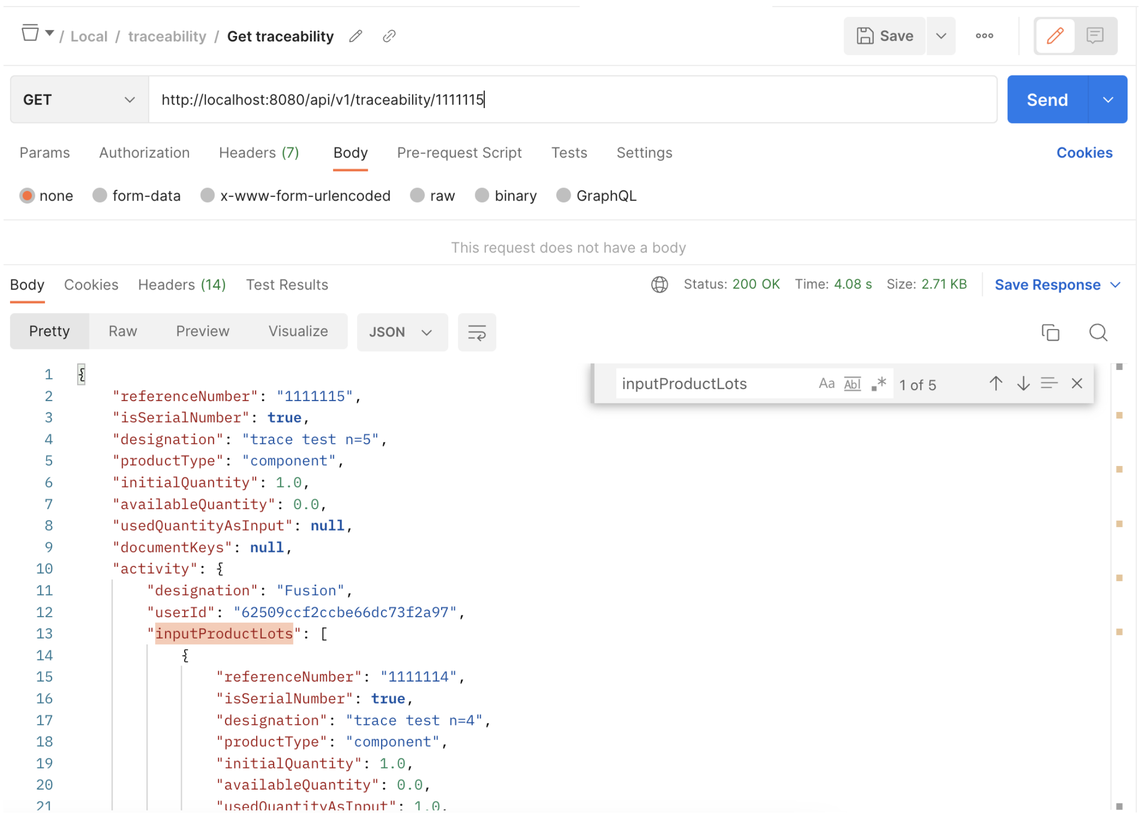
C.8.1 Tests on the recursive function with *depth n = 1*



```
{
    "referenceNumber": "1111111",
    "isSerialNumber": true,
    "designation": "trace test n=1",
    "productType": "component",
    "initialQuantity": 1.0,
    "availableQuantity": 0.0,
    "usedQuantityAsInput": null,
    "documentKeys": null,
    "activity": {
        "designation": "Fusion",
        "userId": "62509ccf2ccbe66dc73f2a97",
        "inputProductLots": [
            {
                "referenceNumber": "bhghuvu",
                "isSerialNumber": false,
                "designation": "bogfssa",
                "productType": "vhrihzp",
                "initialQuantity": 4237.0,
                "availableQuantity": 4235.0,
                "usedQuantityAsInput": 1.0,
```

Figure C.26: Tests with depth *n* = *1*

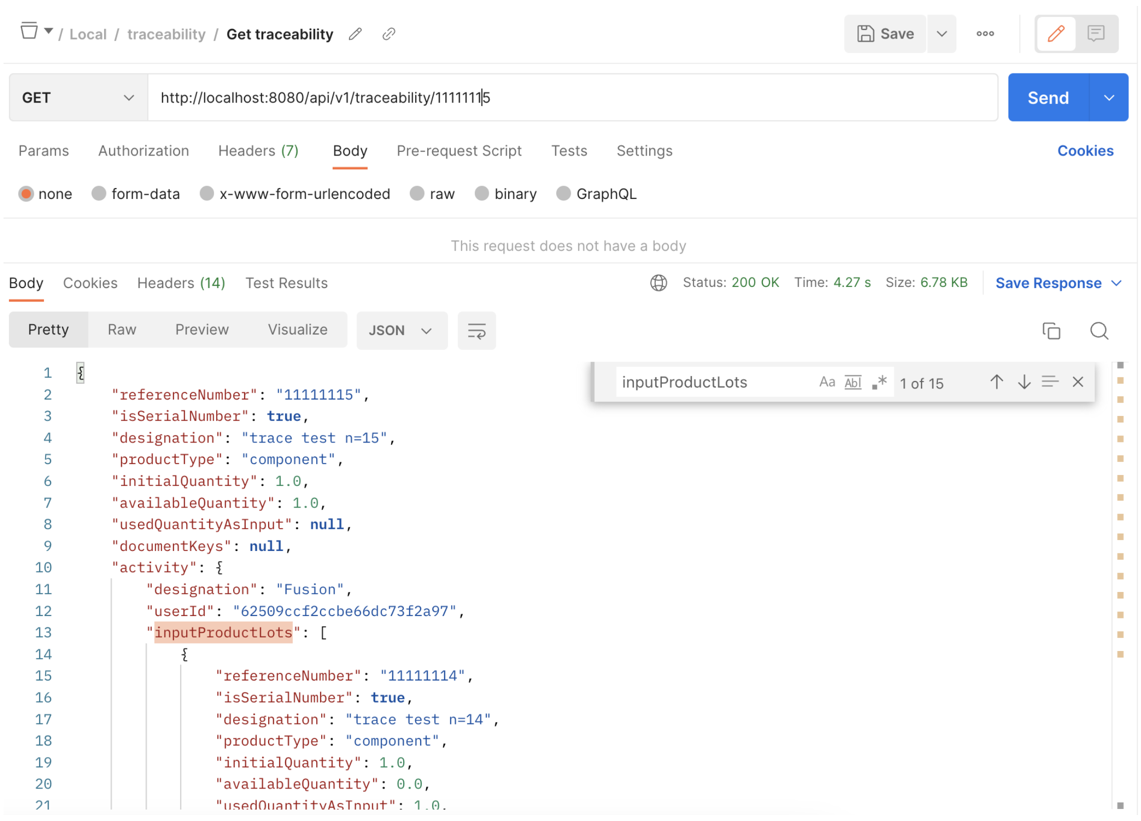
C.8.2 Tests on the recursive function with *depth n = 5*


/ Local / traceability / Get traceability
Save
GET
http://localhost:8080/api/v1/traceability/1111115
Send
Params Authorization Headers (7) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL
This request does not have a body
Body Cookies Headers (14) Test Results
Status: 200 OK Time: 4.08 s Size: 2.71 KB
Save Response
Pretty Raw Preview Visualize JSON
inputProductLots
1 of 5

```
{
    "referenceNumber": "1111115",
    "isSerialNumber": true,
    "designation": "trace test n=5",
    "productType": "component",
    "initialQuantity": 1.0,
    "availableQuantity": 0.0,
    "usedQuantityAsInput": null,
    "documentKeys": null,
    "activity": {
        "designation": "Fusion",
        "userId": "62509ccf2ccbe66dc73f2a97",
        "inputProductLots": [
            {
                "referenceNumber": "1111114",
                "isSerialNumber": true,
                "designation": "trace test n=4",
                "productType": "component",
                "initialQuantity": 1.0,
                "availableQuantity": 0.0,
                "usedQuantityAsInput": 1.0,
```

Figure C.27: Tests with depth $n = 5$

C.8.3 Tests on the recursive function with *depth n = 15*



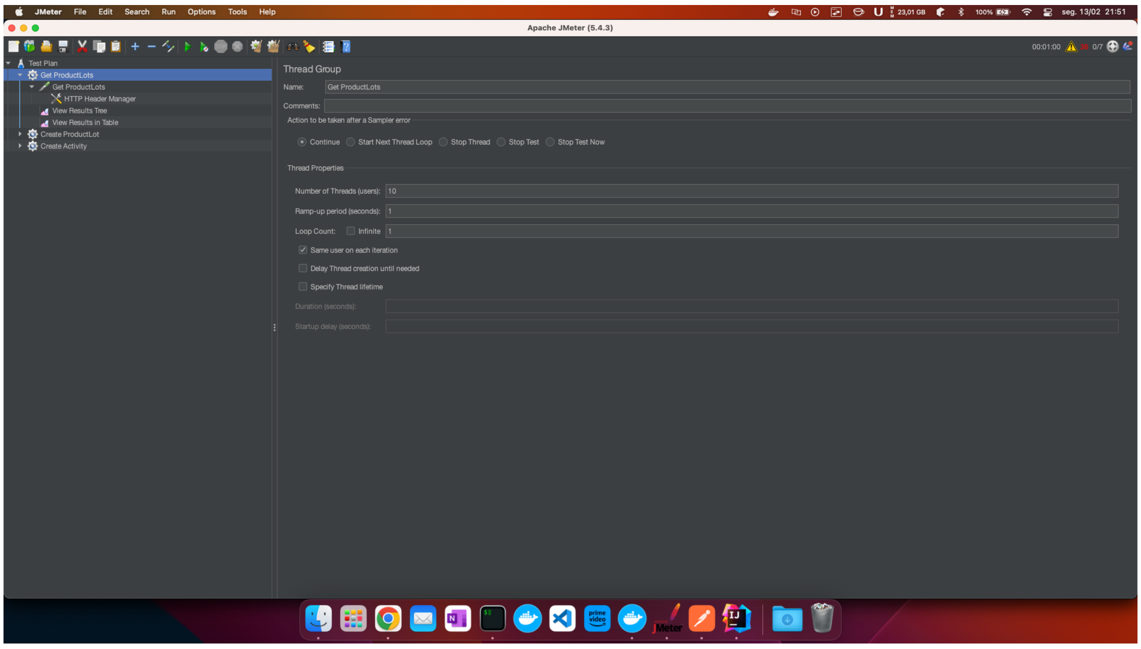
```
{
    "referenceNumber": "11111115",
    "isSerialNumber": true,
    "designation": "trace test n=15",
    "productType": "component",
    "initialQuantity": 1.0,
    "availableQuantity": 1.0,
    "usedQuantityAsInput": null,
    "documentKeys": null,
    "activity": {
        "designation": "Fusion",
        "userId": "62509ccf2ccbe66dc73f2a97",
        "inputProductLots": [
            {
                "referenceNumber": "11111114",
                "isSerialNumber": true,
                "designation": "trace test n=14",
                "productType": "component",
                "initialQuantity": 1.0,
                "availableQuantity": 0.0,
                "usedQuantityAsInput": 1.0,
```

Figure C.28: Tests with depth *n* = *15*

C.9 Performance Tests with JMeter

Figures from the Sections C.9.1, C.9.2 and C.9.3 show the JMeter tests.

C.9.1 Tests on operation “GET ProductLots“



Test Plan
- Get ProductLots
  - Get ProductLots
    - HTTP Header Manager
  - View Results Tree
  - View Results in Table
- Create ProductLot
- Create Activity

Thread Group

Name: Get ProductLots

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 10

Ramp-up period (seconds): 1

Loop Count: ☐ Infinite 1

☑ Same user on each iteration

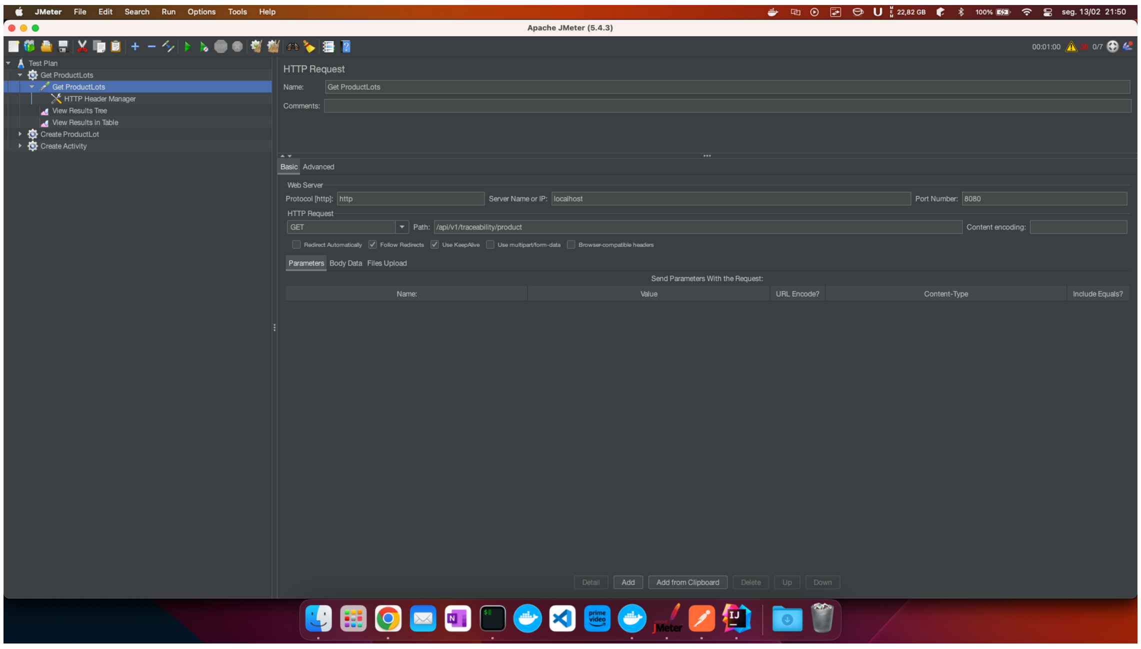
☐ Delay Thread creation until needed

☐ Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

Figure C.29: Setup for the operation “GET ProductLots“



Test Plan
- Get ProductLots
  - Get ProductLots
    - HTTP Header Manager
  - View Results Tree
  - View Results in Table
- Create ProductLot
- Create Activity

HTTP Request

Name: Get ProductLots

Comments:

Basic Advanced

Web Server

Protocol [http]: http Server Name or IP: localhost Port Number: 8080

HTTP Request

GET Path: /api/v1/traceability/product Content encoding:

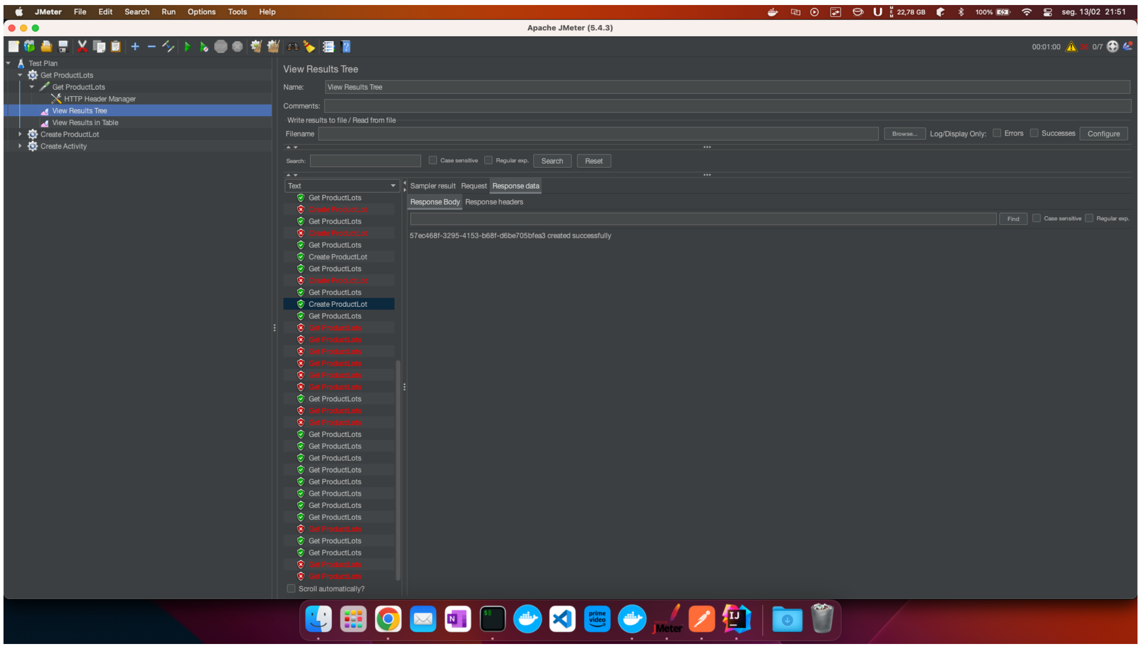
☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Parameters Body Data Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?

Figure C.30: Request used on the operation “GET ProductLots“


Apache JMeter (5.4.3)

View Results Tree

Name: View Results Tree

Comments:

Write results to file / Read from file

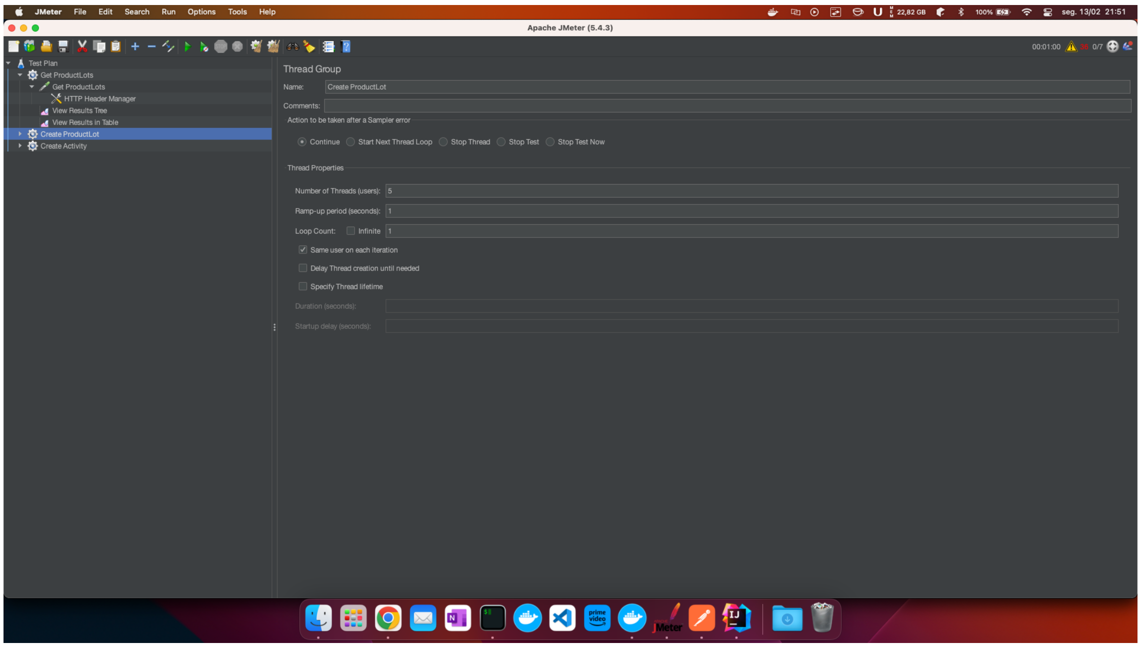
Filename

Response data — Response Body:

```
57ec468f-3295-4153-b68f-d6be705bfea3 created successfully
```

Figure C.31: Results of the operation “GET ProductLots“

C.9.2 Tests on operation “CREATE ProductLot“


Apache JMeter (5.4.3)

Thread Group

Name: Create ProductLot

Comments:

Action to be taken after a Sampler error: [✓ Continue] Start Next Thread Loop / Stop Thread / Stop Test / Stop Test Now

Thread Properties

Number of Threads (users): 5

Ramp-up period (seconds): 1

Loop Count: Infinite ☐ 1

☑ Same user on each iteration

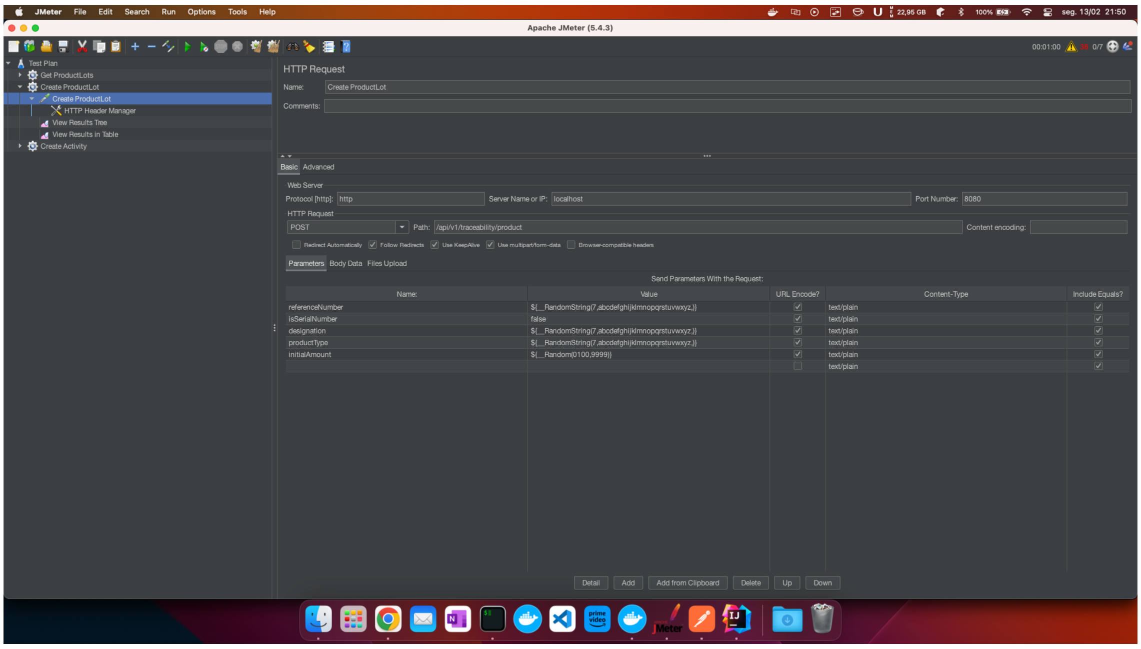
☐ Delay Thread creation until needed

☐ Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

Figure C.32: Setup of the operation “CREATE ProductLot“



HTTP Request

Name: Create ProductLot

Comments:

Web Server

Protocol [http]: http Server Name or IP: localhost Port Number: 8080

HTTP Request

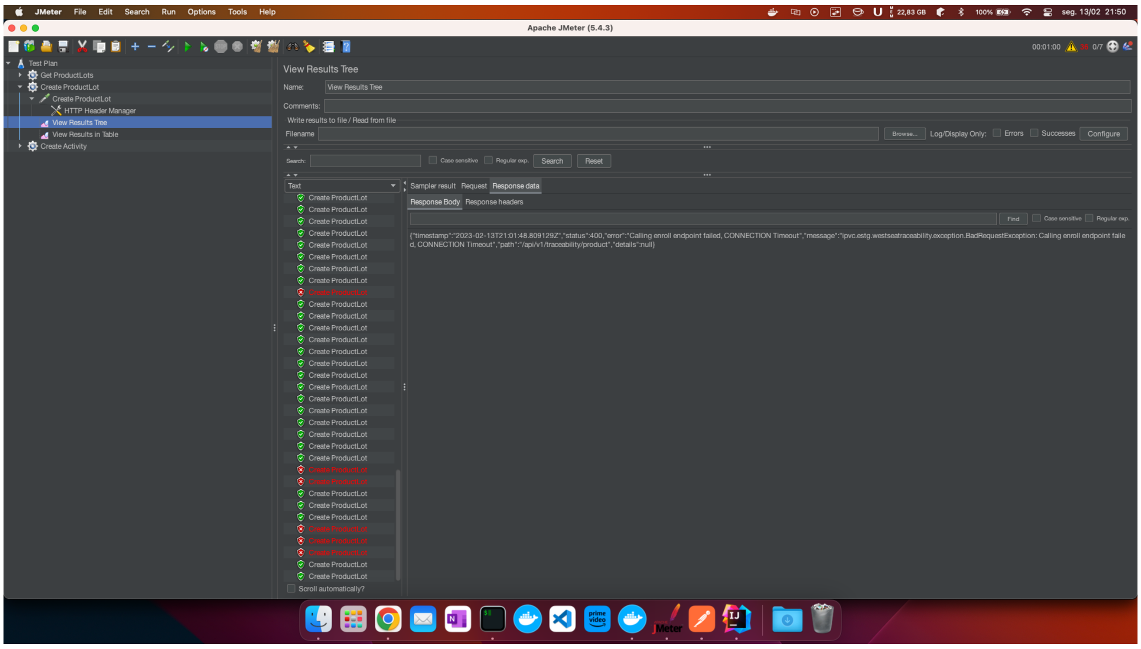
POST Path: /api/v1/traceability/product Content encoding:

☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☑ Use multipart/form-data ☐ Browser-compatible headers

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?
referenceNumber	${__RandomString(7,abcdefghijklmnopqrstuvwxyz,)}	✓	text/plain	✓
isSerialNumber	false	✓	text/plain	✓
designation	${__RandomString(7,abcdefghijklmnopqrstuvwxyz,)}	✓	text/plain	✓
productType	${__RandomString(7,abcdefghijklmnopqrstuvwxyz,)}	✓	text/plain	✓
initialAmount	${__Random(0100,9999)}	✓	text/plain	✓
		☐	text/plain	✓

Figure C.33: Request used on the operation “CREATE ProductLot“



View Results Tree

Name: View Results Tree

Comments:

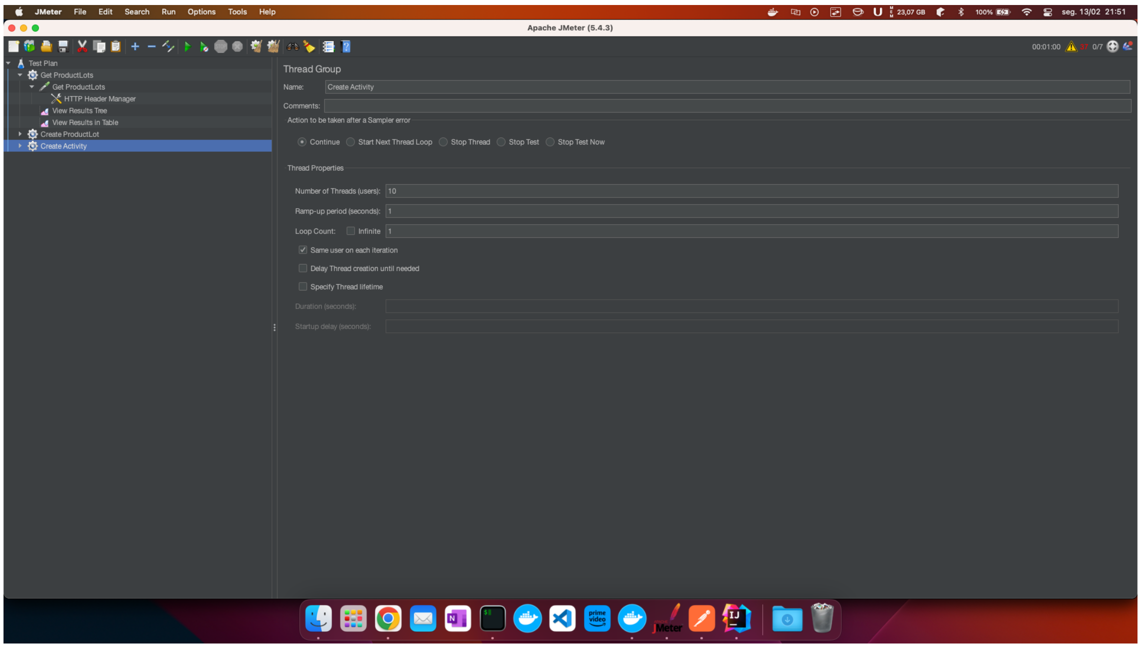
Write results to file / Read from file

Filename Log/Display Only: ☐ Errors ☐ Successes

```
{"timestamp":"2023-02-13T21:01:48.809129Z","status":400,"error":"Calling enroll endpoint failed, CONNECTION Timeout","message":"ipvc.estg.westseatraceability.exception.BadRequestException: Calling enroll endpoint faile
d, CONNECTION Timeout","path":"/api/v1/traceability/product","details":null}
```

Figure C.34: Results of the operation “CREATE ProductLot“

C.9.3 Tests on operation “CREATE Activity“



Thread Group

Name: Create Activity

Comments:

Action to be taken after a Sampler error: [✓ Continue] Start Next Thread Loop / Stop Thread / Stop Test / Stop Test Now

Thread Properties

Number of Threads (users): 10

Ramp-up period (seconds): 1

Loop Count: Infinite ☐ 1

☑ Same user on each iteration

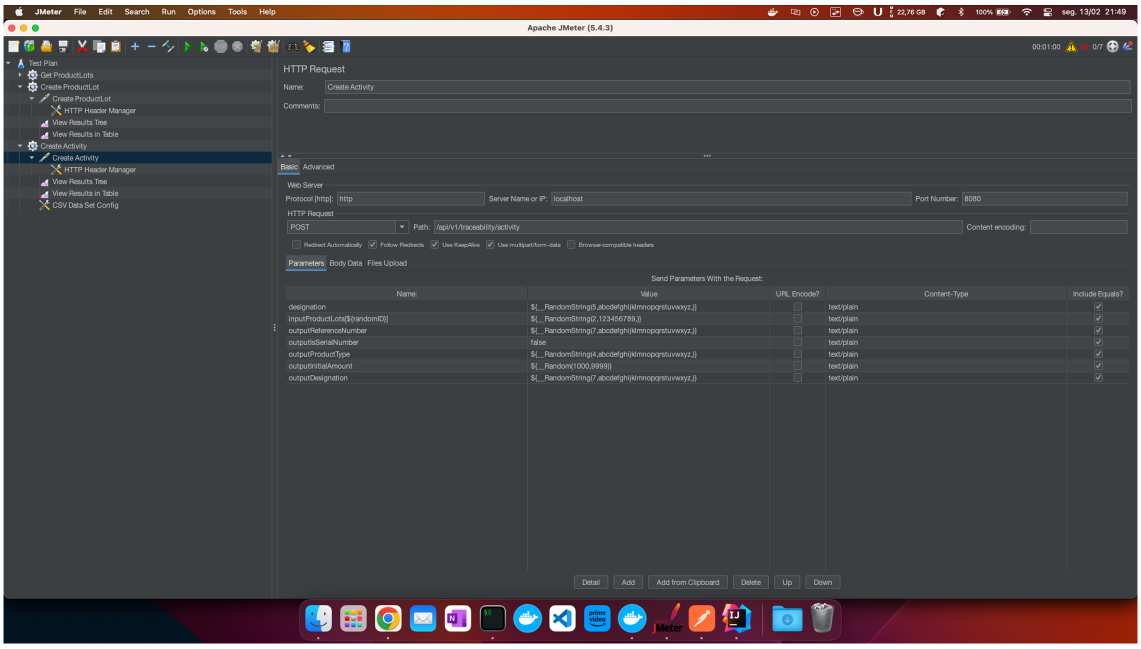
☐ Delay Thread creation until needed

☐ Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

Figure C.35: Setup of the operation “CREATE Activity“



HTTP Request

Name: Create Activity

Comments:

Web Server — Protocol [http]: http Server Name or IP: localhost Port Number: 8080

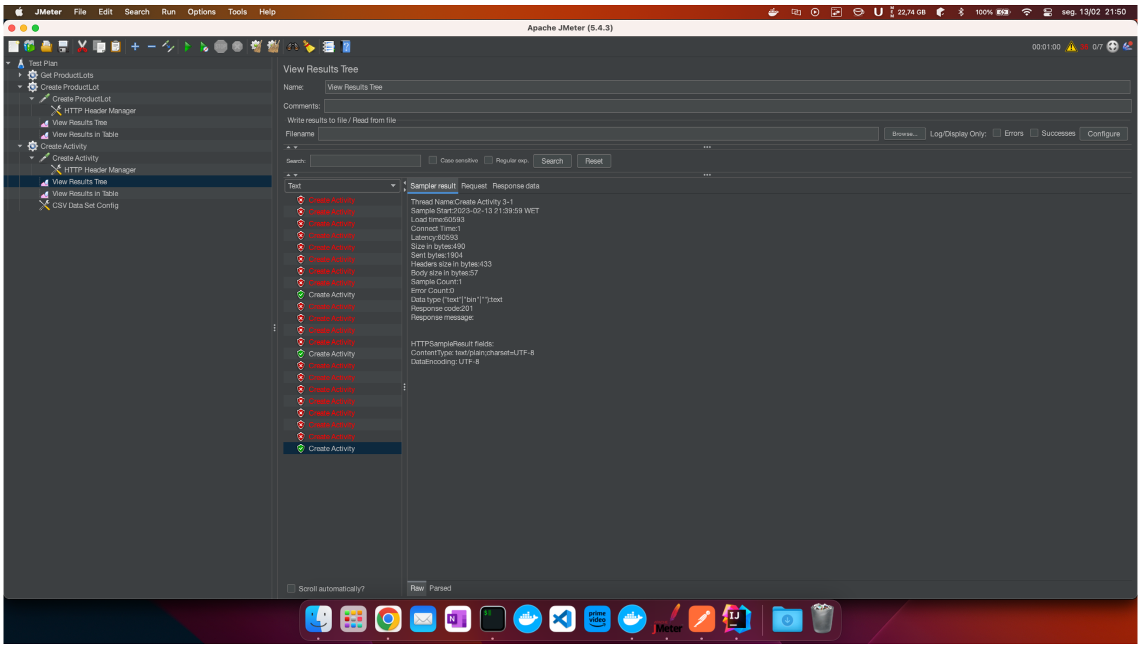
HTTP Request — POST Path: /api/v1/traceability/activity Content encoding:

☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☑ Use multipart/form-data ☐ Browser-compatible headers

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?
designation	${__RandomString(5,abcdefghijklmnopqrstuvwxyz,)}	☐	text/plain	✓
inputProductLots[${randomID}]	${__RandomString(2,123456789,)}	☐	text/plain	✓
outputReferenceNumber	${__RandomString(7,abcdefghijklmnopqrstuvwxyz,)}	☐	text/plain	✓
outputIsSerialNumber	false	☐	text/plain	✓
outputProductType	${__RandomString(4,abcdefghijklmnopqrstuvwxyz,)}	☐	text/plain	✓
outputInitialAmount	${__Random(1000,9999)}	☐	text/plain	✓
outputDesignation	${__RandomString(7,abcdefghijklmnopqrstuvwxyz,)}	☐	text/plain	✓

Figure C.36: Request used on the operation “CREATE Activity“



```
Thread Name:Create Activity 3-1
Sample Start:2023-02-13 21:39:59 WET
Load time:60593
Connect Time:1
Latency:60593
Size in bytes:490
Sent bytes:1904
Headers size in bytes:433
Body size in bytes:57
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:201
Response message:


HTTPSampleResult fields:
ContentType: text/plain;charset=UTF-8
DataEncoding: UTF-8
```

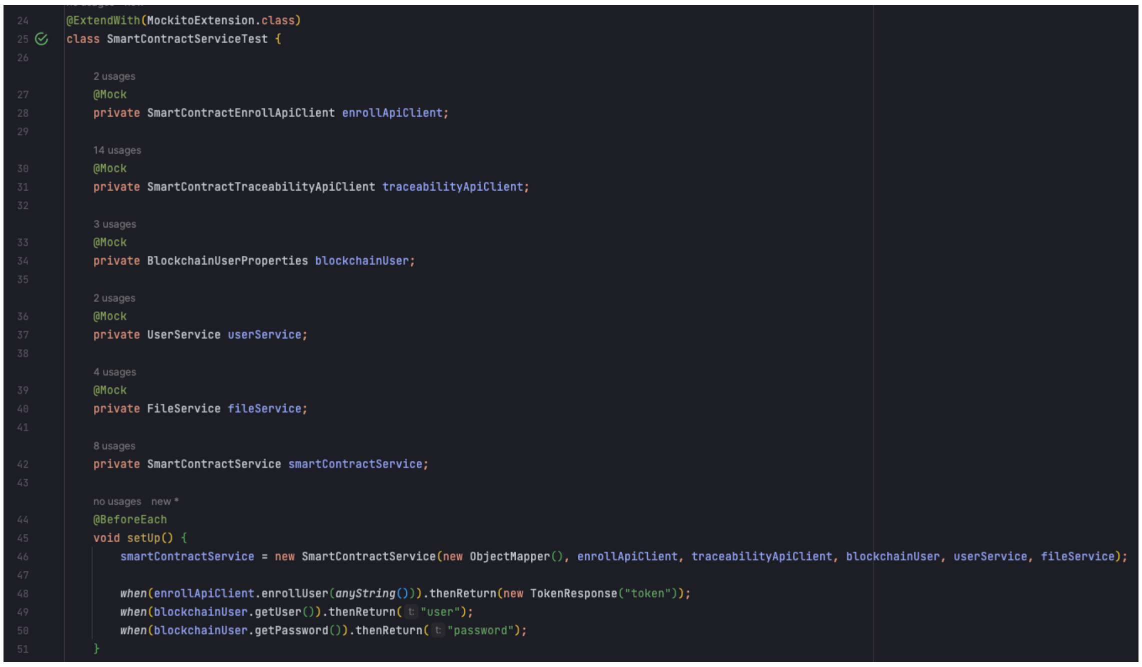
Figure C.37: Results of the operation “CREATE Activity“

C.10 Unit tests on the most important methods of the General REST API

The following sections contain figures of the unit tests and results.

C.10.1 Setup for SmartContractServiceTest class

This class contains the tests for the methods that interact with the Smart Contract API.

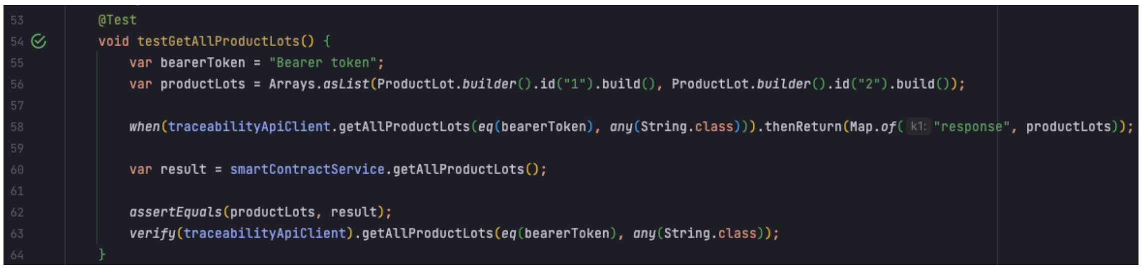


```java
@ExtendWith(MockitoExtension.class)
class SmartContractServiceTest {

    2 usages
    @Mock
    private SmartContractEnrollApiClient enrollApiClient;

    14 usages
    @Mock
    private SmartContractTraceabilityApiClient traceabilityApiClient;

    3 usages
    @Mock
    private BlockchainUserProperties blockchainUser;

    2 usages
    @Mock
    private UserService userService;

    4 usages
    @Mock
    private FileService fileService;

    8 usages
    private SmartContractService smartContractService;

    no usages   new *
    @BeforeEach
    void setUp() {
        smartContractService = new SmartContractService(new ObjectMapper(), enrollApiClient, traceabilityApiClient, blockchainUser, userService, fileService);

        when(enrollApiClient.enrollUser(anyString())).thenReturn(new TokenResponse("token"));
        when(blockchainUser.getUser()).thenReturn( t: "user");
        when(blockchainUser.getPassword()).thenReturn( t: "password");
    }
```

Figure C.38: Setup of the SmartContractServiceTest class

C.10.2 Unit test on the method “getAllProductLots“

This method returns all ProductLots.

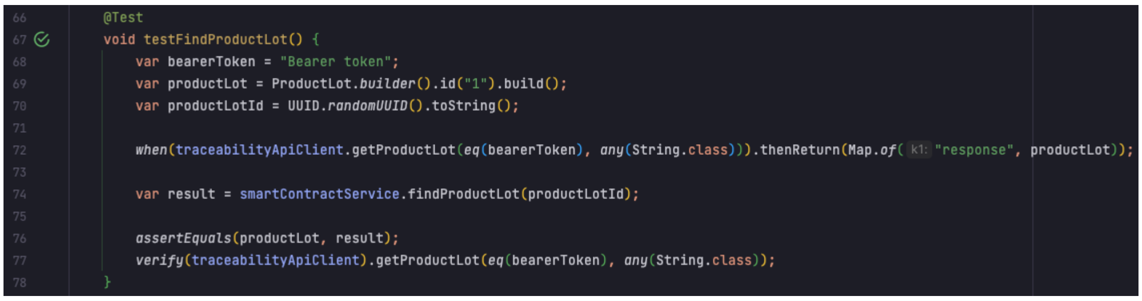


```java
    @Test
    void testGetAllProductLots() {
        var bearerToken = "Bearer token";
        var productLots = Arrays.asList(ProductLot.builder().id("1").build(), ProductLot.builder().id("2").build());

        when(traceabilityApiClient.getAllProductLots(eq(bearerToken), any(String.class))).thenReturn(Map.of( k1: "response", productLots));

        var result = smartContractService.getAllProductLots();

        assertEquals(productLots, result);
        verify(traceabilityApiClient).getAllProductLots(eq(bearerToken), any(String.class));
    }
```

Figure C.39: “getAllProductLots“ unit test

C.10.3 Unit test on the method “findProductLot“

This method returns a ProductLot by its ID.

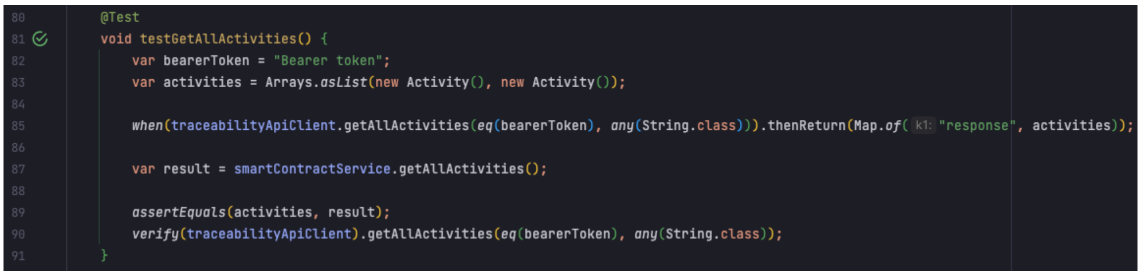


```java
@Test
void testFindProductLot() {
    var bearerToken = "Bearer token";
    var productLot = ProductLot.builder().id("1").build();
    var productLotId = UUID.randomUUID().toString();

    when(traceabilityApiClient.getProductLot(eq(bearerToken), any(String.class))).thenReturn(Map.of( k1: "response", productLot));

    var result = smartContractService.findProductLot(productLotId);

    assertEquals(productLot, result);
    verify(traceabilityApiClient).getProductLot(eq(bearerToken), any(String.class));
}
```

Figure C.40: “findProductLot“ unit test

C.10.4 Unit test on the method “getAllActivities“

This method returns all Activities.

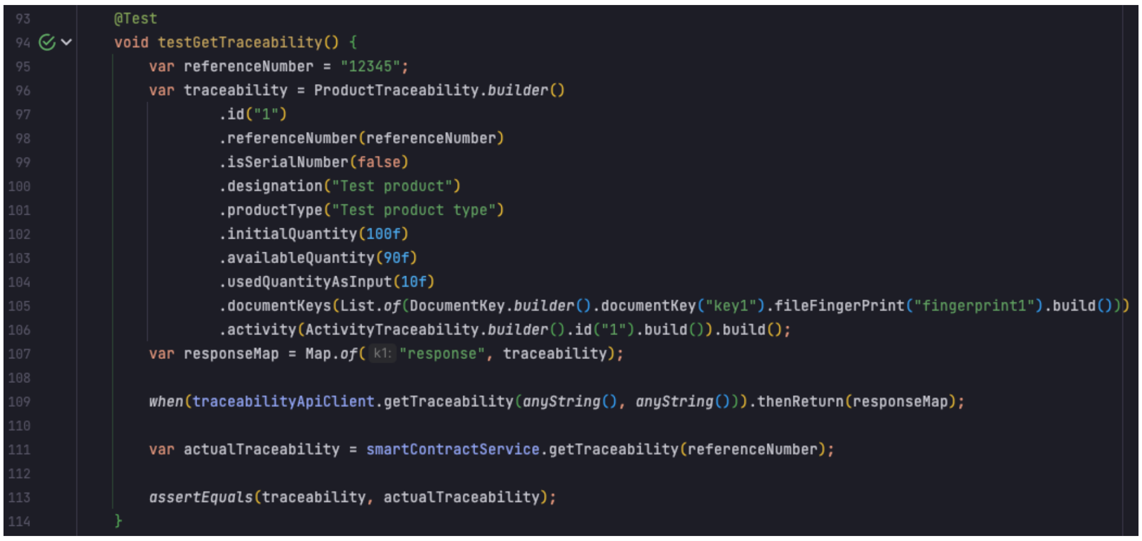


```java
@Test
void testGetAllActivities() {
    var bearerToken = "Bearer token";
    var activities = Arrays.asList(new Activity(), new Activity());

    when(traceabilityApiClient.getAllActivities(eq(bearerToken), any(String.class))).thenReturn(Map.of( k1: "response", activities));

    var result = smartContractService.getAllActivities();

    assertEquals(activities, result);
    verify(traceabilityApiClient).getAllActivities(eq(bearerToken), any(String.class));
}
```

Figure C.41: “getAllActivities“ unit test

C.10.5 Unit test on the method “getTraceability“

This method returns the traceability of a ProductLot by its reference number.

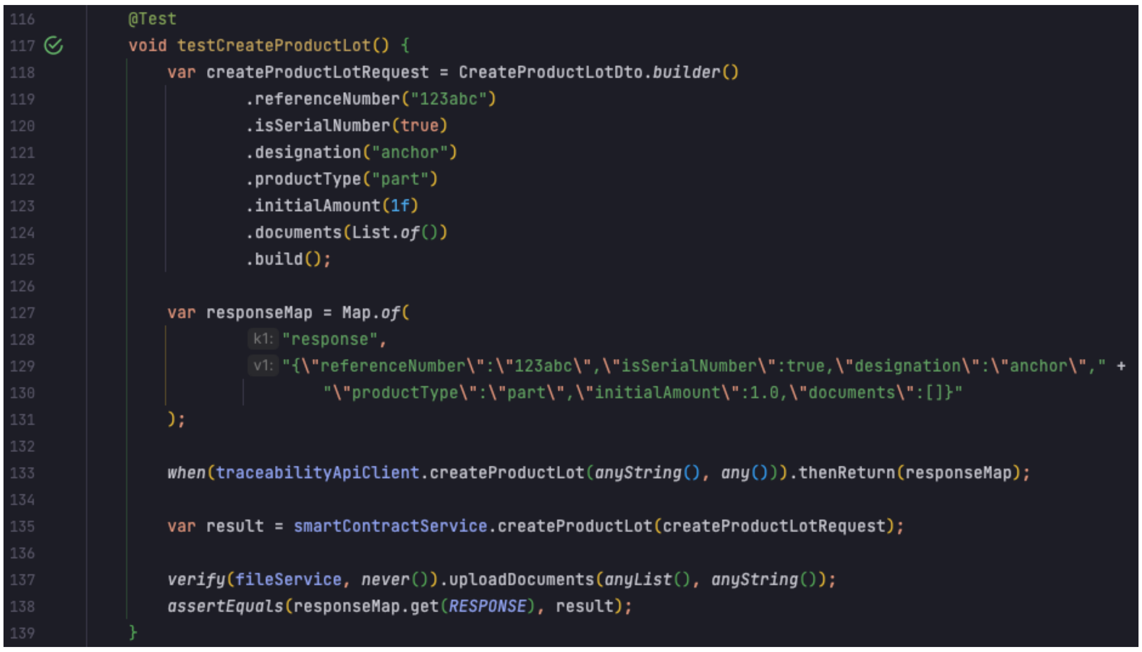


```java
@Test
void testGetTraceability() {
    var referenceNumber = "12345";
    var traceability = ProductTraceability.builder()
            .id("1")
            .referenceNumber(referenceNumber)
            .isSerialNumber(false)
            .designation("Test product")
            .productType("Test product type")
            .initialQuantity(100f)
            .availableQuantity(90f)
            .usedQuantityAsInput(10f)
            .documentKeys(List.of(DocumentKey.builder().documentKey("key1").fileFingerPrint("fingerprint1").build()))
            .activity(ActivityTraceability.builder().id("1").build()).build();
    var responseMap = Map.of( k1: "response", traceability);

    when(traceabilityApiClient.getTraceability(anyString(), anyString())).thenReturn(responseMap);

    var actualTraceability = smartContractService.getTraceability(referenceNumber);

    assertEquals(traceability, actualTraceability);
}
```

Figure C.42: “getTraceability“ unit test

C.10.6 Unit test on the method “createProductLot“

This method creates a new ProductLot and returns it.

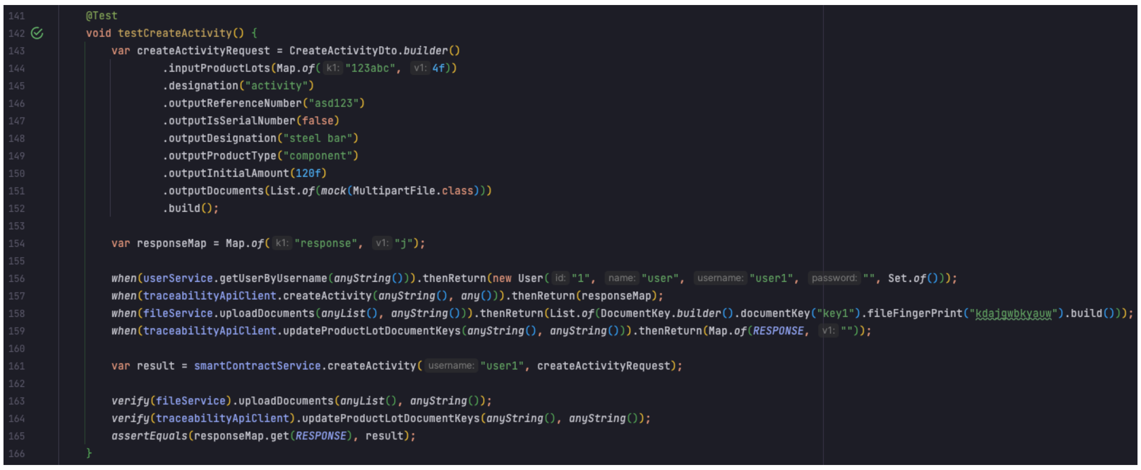


```java
    @Test
    void testCreateProductLot() {
        var createProductLotRequest = CreateProductLotDto.builder()
                .referenceNumber("123abc")
                .isSerialNumber(true)
                .designation("anchor")
                .productType("part")
                .initialAmount(1f)
                .documents(List.of())
                .build();

        var responseMap = Map.of(
                k1: "response",
                v1: "{\"referenceNumber\":\"123abc\",\"isSerialNumber\":true,\"designation\":\"anchor\"," +
                        "\"productType\":\"part\",\"initialAmount\":1.0,\"documents\":[]}"
        );

        when(traceabilityApiClient.createProductLot(anyString(), any())).thenReturn(responseMap);

        var result = smartContractService.createProductLot(createProductLotRequest);

        verify(fileService, never()).uploadDocuments(anyList(), anyString());
        assertEquals(responseMap.get(RESPONSE), result);
    }
```

Figure C.43: “createProductLot“ unit test

C.10.7 Unit test on the method “createActivity“

This method creates a new Activity and returns it.

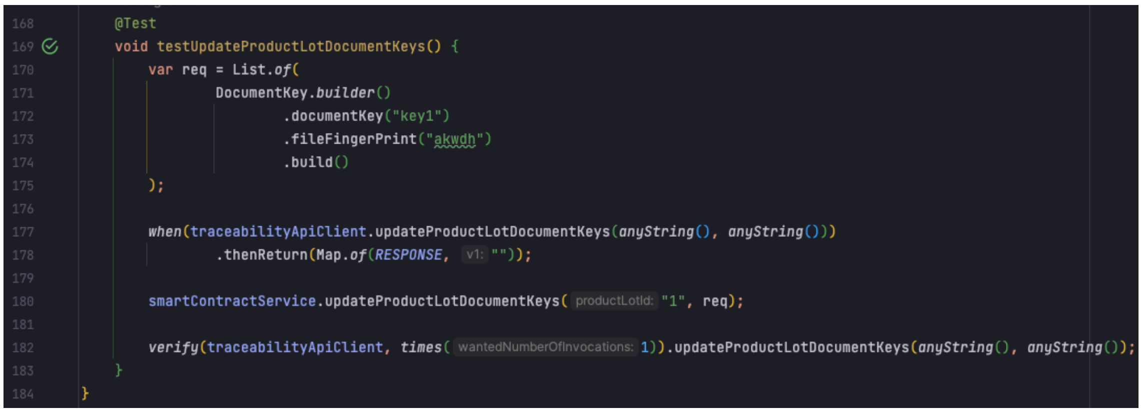


```java
    @Test
    void testCreateActivity() {
        var createActivityRequest = CreateActivityDto.builder()
                .inputProductLots(Map.of( k1: "123abc",  v1: 4f))
                .designation("activity")
                .outputReferenceNumber("asd123")
                .outputIsSerialNumber(false)
                .outputDesignation("steel bar")
                .outputProductType("component")
                .outputInitialAmount(120f)
                .outputDocuments(List.of(mock(MultipartFile.class)))
                .build();

        var responseMap = Map.of( k1: "response",  v1: "j");

        when(userService.getUserByUsername(anyString())).thenReturn(new User( id: "1",  name: "user",  username: "user1",  password: "", Set.of()));
        when(traceabilityApiClient.createActivity(anyString(), any())).thenReturn(responseMap);
        when(fileService.uploadDocuments(anyList(), anyString())).thenReturn(List.of(DocumentKey.builder().documentKey("key1").fileFingerPrint("kdajgwbkyauw").build()));
        when(traceabilityApiClient.updateProductLotDocumentKeys(anyString(), anyString())).thenReturn(Map.of(RESPONSE,  v1: ""));

        var result = smartContractService.createActivity( username: "user1", createActivityRequest);

        verify(fileService).uploadDocuments(anyList(), anyString());
        verify(traceabilityApiClient).updateProductLotDocumentKeys(anyString(), anyString());
        assertEquals(responseMap.get(RESPONSE), result);
    }
```

Figure C.44: “createActivity“ unit test

C.10.8 Unit test on the method “updateProductLotDocumentKeys“

This method updates the documents of a ProductLot.

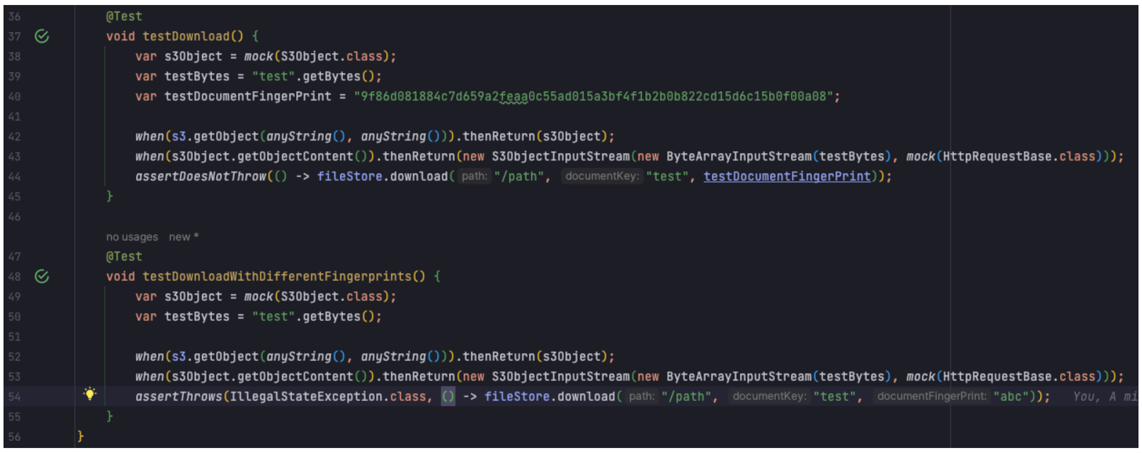


```java
    @Test
    void testUpdateProductLotDocumentKeys() {
        var req = List.of(
                DocumentKey.builder()
                        .documentKey("key1")
                        .fileFingerPrint("akwdh")
                        .build()
        );

        when(traceabilityApiClient.updateProductLotDocumentKeys(anyString(), anyString()))
                .thenReturn(Map.of(RESPONSE,  v1: ""));

        smartContractService.updateProductLotDocumentKeys( productLotId: "1", req);

        verify(traceabilityApiClient, times( wantedNumberOfInvocations: 1)).updateProductLotDocumentKeys(anyString(), anyString());
    }
}
```

Figure C.45: “updateProductLotDocumentKeys“ unit test

C.10.9 Unit tests on the method “download“

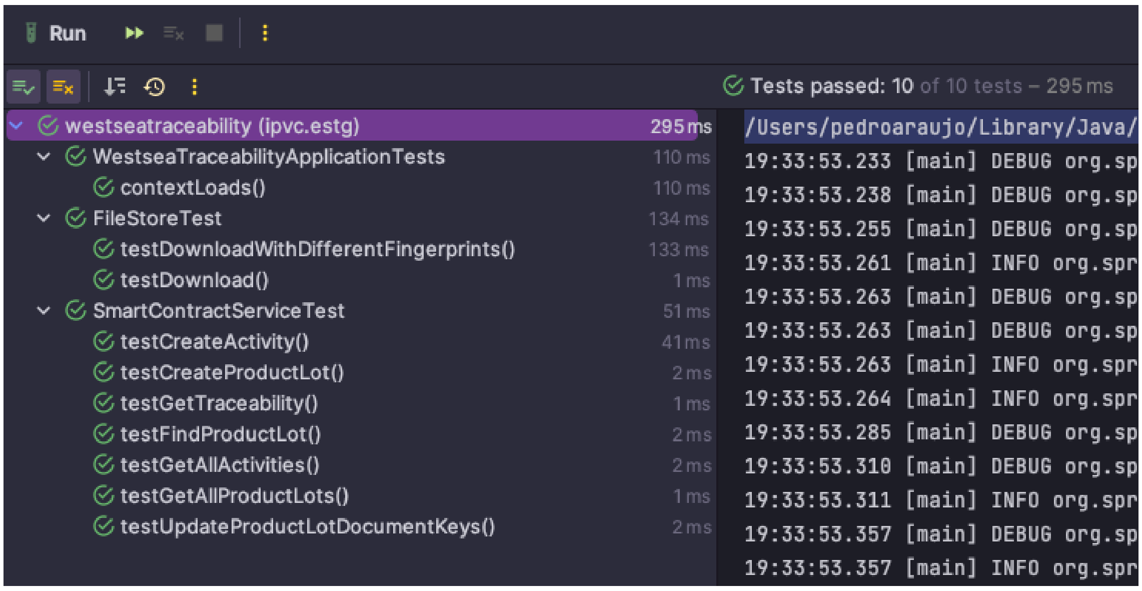
This method downloads a document from the AWS S3 Bucket by its key, it also checks if the document fingerprint of the document to download is the same as the one requests (stored on the blockchain), if it is not, it will throw an error. This method is from the class “FileStore“ and the only setup is to instantiate the class.



```java
    @Test
    void testDownload() {
        var s3Object = mock(S3Object.class);
        var testBytes = "test".getBytes();
        var testDocumentFingerPrint = "9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08";

        when(s3.getObject(anyString(), anyString())).thenReturn(s3Object);
        when(s3Object.getObjectContent()).thenReturn(new S3ObjectInputStream(new ByteArrayInputStream(testBytes), mock(HttpRequestBase.class)));
        assertDoesNotThrow(() -> fileStore.download( path: "/path",  documentKey: "test", testDocumentFingerPrint));
    }

    no usages   new *
    @Test
    void testDownloadWithDifferentFingerprints() {
        var s3Object = mock(S3Object.class);
        var testBytes = "test".getBytes();

        when(s3.getObject(anyString(), anyString())).thenReturn(s3Object);
        when(s3Object.getObjectContent()).thenReturn(new S3ObjectInputStream(new ByteArrayInputStream(testBytes), mock(HttpRequestBase.class)));
        assertThrows(IllegalStateException.class, () -> fileStore.download( path: "/path",  documentKey: "test",  documentFingerPrint: "abc"));   You, A mi
    }
}
```

Figure C.46: “download“ unit tests

C.10.10 Results of the presented unit tests



```
westseatraceability (ipvc.estg)                 295 ms
  WestseaTraceabilityApplicationTests           110 ms
    contextLoads()                              110 ms
  FileStoreTest                                 134 ms
    testDownloadWithDifferentFingerprints()     133 ms
    testDownload()                                1 ms
  SmartContractServiceTest                       51 ms
    testCreateActivity()                         41 ms
    testCreateProductLot()                        2 ms
    testGetTraceability()                         1 ms
    testFindProductLot()                          2 ms
    testGetAllActivities()                        2 ms
    testGetAllProductLots()                       1 ms
    testUpdateProductLotDocumentKeys()            2 ms
```

```
/Users/pedroaraujo/Library/Java/
19:33:53.233 [main] DEBUG org.sp
19:33:53.238 [main] DEBUG org.sp
19:33:53.255 [main] DEBUG org.sp
19:33:53.261 [main] INFO org.spr
19:33:53.263 [main] DEBUG org.sp
19:33:53.263 [main] DEBUG org.sp
19:33:53.263 [main] INFO org.spr
19:33:53.264 [main] INFO org.spr
19:33:53.285 [main] DEBUG org.sp
19:33:53.310 [main] DEBUG org.sp
19:33:53.311 [main] INFO org.spr
19:33:53.357 [main] DEBUG org.sp
19:33:53.357 [main] INFO org.spr
```

Figure C.47: Results of the unit tests