POLITECNICO DI TORINO

Master Degree in
SOFTWARE ENGINEERING

Master Thesis

Building a distributed ledger to support a e-commerce website

Supervisori

Prof. LUCA ARDITO

Prof. TOMMASO FULCINI

Candidato

HUANG HAIHANG

03/2022

Abstract

In order to effectively solve the trust problem caused by traditional e-commerce platforms storing and maintaining transaction data by themselves, improve the authenticity of commodity information on e-commerce platforms and the ability of customers to protect their rights when shopping, this paper designs and implements a distributed ledger module based on Hyperledger Fabric and deployed on the Digital Managment Events project. Users conduct transactions with merchants through the website, and the transaction data is stored in the blockchain, which is unmodifiable and highly credible. The transaction data is stored on the blockchain, and the information used for business data statistics also comes from the blockchain. Compared with traditional platforms, the implemented system greatly improves the credibility of transaction history records and provides buyers and sellers with more objective transaction credentials. It is highly reliable and easy to implement, and the technology used can also be used in the development of websites for blockchain applications in other fields.

Acknowledgements

Standing at the tail end of my master's study career, too many things have happened in the past two years. I found that my student status gave me too much protection, and there were too many people who took care of me and helped me during this period. There are too many thanks and gratitude.
First of all, I would like to thank Prof. LUCA ARDITO and Prof. TOMMASO FULCINI for giving me the opportunity to participate in this topic. I completed the research and writing of this dissertation under their guidance, and I benefited a lot from their enthusiasm and wealth of knowledge. My learning ability, autonomy and mentality have been greatly improved and improved.
The sudden outbreak of COVID-19 disrupted the original study abroad life, thank you for the people who resisted danger and gave silently during this special period. And thanks to the friends I met in Italy, Huang Chunbiao, Shi Feihong, Cao Peng, Chen Jinzhuo, Zeng Haohang, Stefano Romboni, Cai Siqi, Giacomo and my landlord Cecilia, you guys form a complete story, making me miss the time with you so much, and I look forward to it again goodbye.
Finally, I would like to thank my parents and family for supporting me over the years, you have made me more optimistic about everything in life. Thanks also to my girlfriend Ma Xueli, thank you for always encouraging me, making me happy, full of energy for projects, full of confidence in life, thank you for appearing in my life.

Table of Contents

List of Tables

List of Figures

Acronyms

GUI
: Graphical User Interface

DOM
: Document Object Model

API
: Application Programming Interface

BND
: Business Network Definition

CA
: Certification Authority

CLI
: Command-Line Interface

POC
: Proof Of Concept

Capitolo 1

Introduction

Digital Management of Events is a project funded by the Puglia region, aimed at creating a digital platform, which is initially planned to be used in the Apulian market but with the aim of expanding into the national territory. The digital platform is aimed at supporting users and suppliers who want to organize, manage, communicate and participate in various types of events. Having established the vastness of the project, its partners are many: The Polytechnic of Turin, the University of Bari, DS Graphic Engineering S.r.l, STRADE S.r.l, Linear System. In particular, the role that the Polytechnic of Turin assumes is that of the Research and Development of the platform in close collaboration with the Linear System company. The work of the Politecnico foreseen during the period concerning the thesis work concerns two macro-periods that we can divide into:

- Update and maintain some functions of the front end of the DME website;
- Study, and analysis of the solutions suitable for the project, based on Hyperledger for the implementation of Smart Contracts;;

1.1 Distributed Ledger

With the steady growth of communication and security technologies, blockchains have emerged as digital innovations that would transform many industries and businesses.[5] These are the so-called "distributed" systems, which are characterized by synchronized replication of the register in different points called nodes, which behave as independent systems connected to each other by a communication network, in which each node manages its own copy, kept aligned with the others. What distinguishes DLT from a replica of information accessible locally by the single node in reading but modified through the intervention of a central body that determines which updates are to be distributed to all nodes in the network, is the

capacity of the systems that adopt it to manage changes to the register itself in a distributed manner. Each update of the register is regulated through consent mechanisms that allow the network to converge on a single version of the register itself. Despite the fact that the subjects within the network independently modify the information. The most evident characteristics that allow to distinguish the different distributed ledger systems involve three main aspects:

- The type of network that determines which actors are part of it and their role in the validation process;
- The consent mechanism, which determines the ways in which the network updates the register;
- The structure of the register that defines how information is organized within the register;

Based on the type of network that has been chosen, we can distinguish between different categories of systems:
Public: anyone can access to read transaction data without the need for explicit approval by a person;
Private: on the contrary, there is a central body that manages access to data, limiting only to a set of authorized users.

Systems are distinguished on the basis of the role attributed to the actors in the validation process:
Permissionless: all the actors can access the data, validate the transactions, without particular authorizations;
Permissioned: only certain actors are authorized to carry out operations. The system that has been created can be said to have chosen a private type system category with permissions. Finally, for the sake of completeness, there is a third characteristic of distributed ledger systems which is the structure of the register, or how the information is organized within the register:

- *traditional ledger* in which the information is validated and recorded at the single transaction level;
- *chain of blocks*, a way in which transactions are collected within a block that is validated and recorded at an aggregate level with another.

1.2 Microservices architecture

The microservice architecture is to split the unified system into several relatively independent and autonomous sub-services according to business and data. Each

service is independent and must implement a single business functionality. In monolithic architectures, all processes are closely linked each other and run as a single service. This means that if an application process experiences a peak in demand, the entire architecture needs to be resized. Add or improve an application functionality becomes more complex, as it will be necessary to increase the code base. This complexity limits experimentation and makes it more difficult to implement new ideas. Monolithic architectures represent an additional risk for the availability of the application, since the presence of numerous dependent and closely related processes increases the impact of an error in a single process.

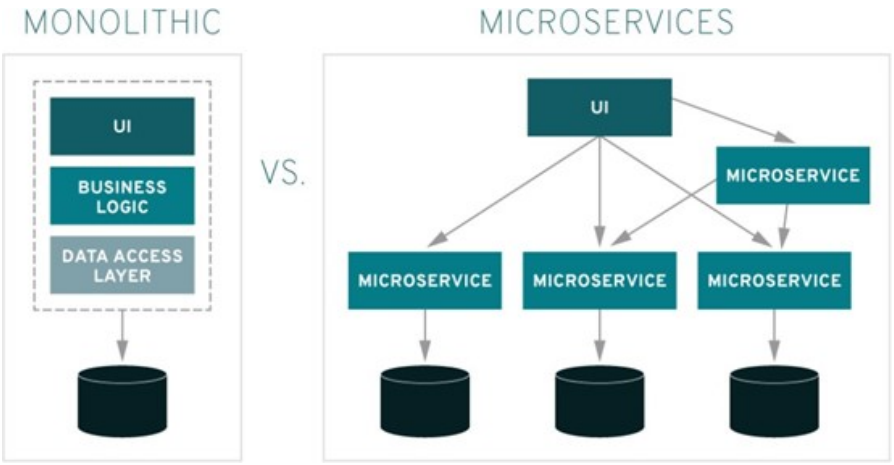

Figura 1.1: comparison between monolithic scheme and microservices scheme

With a microservices-based architecture, an application is built from independent components that run each application process as a service. These services communicate through a well-defined interface that uses lightweight APIs. The services are built for business functions and each service performs only one function. Since run independently, each service can be updated, distributed and resized to respond to the request for specific functions of an application. The size of the microservices is such that they can be written and managed by a single small team of developers. A team can update an existing service without recompiling and redeploying the entire application.[16]

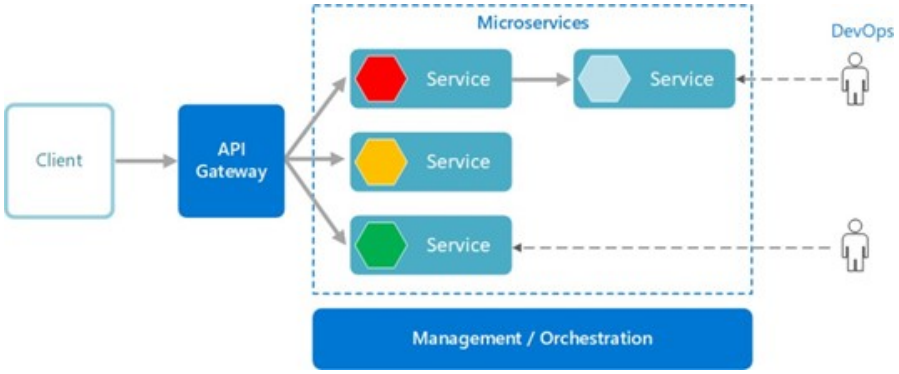

Figura 1.2: microservice scheme

A service is responsible for the persistence of its data or external state. Painless with the traditional way, where data persistence is managed solely by the data layer. Create APIs to make these services accessible. How the service is implemented internally is invisible to other services.The services do not need to share the same frameworks, the same libraries, or the same technology stack. In addition to services, some other components appear in a typical microservices architecture:

- Management/orchestration: This component is responsible for positioning services on nodes, identifying errors, rebalancing services between nodes and so on;
- API Gateway: The API Gateway is the entry point for clients. Instead of calling the services directly, clients call the API gateway, which forwards the call to the appropriate services on the backend.
  The benefits associated with using an API gateway include the separation of clients from services, which can be refactored or versioned without the need to update all clients, another benefit is the use by the services of non-web-compatible messaging protocols, such as AMQP, finally the API gateway can perform other cross functions, such as authentication, registration, SSL termination and load balancing

Splitting an app into its basic functions and avoiding the pitfalls of the monolithic approach may seem familiar concepts, because the architectural style of microservices is similar to that of SOA (Service-Oriented Architecture), a well-established style of software design.

In the early days of application development, even the slightest change to an existing app required a complete update and a quality assurance (QA) cycle of its own, which risked slowing down the work of various secondary teams. This approach is often referred to as "monolithic", because the source code of the entire app was compiled into a single deployment unit (for example with the extension

.war or .ear). If updates to a part of the app caused errors, it was necessary to disconnect everything, take a step back and correct. This approach is still applicable to small applications, but growing businesses cannot afford downtime.

This is where the Service-Oriented Architecture (SOA) comes into play, in which apps are structured into reusable services that communicate with each other via an Enterprise Service Bus (ESB). In this architecture, each service has its own specific business process and follows a communication protocol, including Apache Thrift, ActiveMQ, or SOAP, to be shared through the Enterprise Service Bus. Overall, this suite of services integrated through an ESB constitutes an application.[19]

In addition, this method allows you to build, test and modify multiple services at the same time, freeing IT teams from monolithic development cycles. However, since the ESB represents a single point of failure for the entire system, it could pose an obstacle to the entire organization.

The difference between SOA architecture and microservices is that microservices can communicate with each other, usually in a stateless manner, allowing apps to be built with greater fault tolerance and less dependent on a single enterprise service bus (ESB). In addition, they communicate via programming interfaces language-independent applications (APIs), which allows each development team to choose their own tools.

Considering the evolution of SOA[3], microservices are not an absolute novelty, but lately they have become more attractive thanks to advances in containerization technologies. Today, containers allow you to run multiple parts of an app independently, on the same hardware, with much greater control over individual components and life cycles. Containerized microservices form the foundation of cloud-native applications[8].

1.3 Responsive Web Development

In the chapter we will introduce responsive web development, the main technologies used for the development of the latter will be illustrated. Recently, new languages and frameworks have been created that are able to respond to users' requests for interactivity, as well as the greater diffusion of mobile devices with web connections. The main teams are Microsoft and Google who have contributed to the creation of tools capable of helping developers in the creation of increasingly advanced applications.

1.4 The current problems of e-commerce

1.4.1 The lack of integrity of sellers leads to a crisis of platform reputation

Under the rapid development of e-commerce, the competition in the industry is becoming more and more fierce. Improving the reputation of sellers has become an indispensable step in healthy development. For example, some merchants use the opacity of online transaction information to meet their own interests and reduce commodity prices, giving away non-for-sale products and other promotional methods to grab the attention of consumers. However, the sold products are manufactured using low-cost defective products or pirated fakes. Some merchants use gifts or post-purchase discounts to get favorable comments that go against the buyer's original intention, others will unscrupulously register fake customers or steal consumer information for evaluation.

In addition, some consumers are worried about receiving disturbing information from merchants after giving unfavorable comments, therefore, in order to avoid trouble, they generally give good reviews. These methods have made the sellers gain huge profits in a short period of time, but also to a crisis of reputation. On the one hand, it is for their own brand image, and on the other hand, for the entire e-commerce platform, they are trapped in public opinion. The brand image has been seriously affected, and we have to pay attention to the supervision of the merchant's transaction process within the platform, and to crack down on the sale of fake and shoddy products.[6]

1.4.2 Consumer privacy cannot be guaranteed

At present, the vast majority of e-commerce transactions are conducted through third-party platforms, and the entire process is conducted in an opaque and undisclosed environment. There is a possibility that personal information may be exposed to the internet by criminals, which has aroused public concern to a certain extent.

Although third-party platforms can protect consumers' personal information somehow, in recent years, incidents of platforms leaking or stealing consumers' personal information have also caused them to be nervous and panic, and even worse. The increased incidence of surveillance and security breach incidents in recent investigations has led to questions about the security of users' privacy, which calls into question the current model. How to better protect consumption The privacy of consumers, the protection of consumers' rights and interests, and the realization of worry-free online shopping have attracted the attention of e-commerce platforms and consumers.[1]

1.4.3 Supply chain enterprise information sharing is not immediate and opaque

The entire supply chain will revolve around a core enterprise, through the control of information flow, logistics, and capital flow transmission, through purchasing materials, to intermediate products and final products, and finally delivering products to consumers through terminal channels. Factory suppliers, distributors, logistics and warehousing enterprises, consumers and after-sales service providers are the main roles in the supply chain system.

Members in the supply chain need to strengthen communication, cooperation and information sharing, so as to improve the overall collaboration efficiency. Due to the dispersion of information, opacity and lack of communication, The process of supply chain integration is seriously hindered. Insufficient and delayed information communication between different organizations of the enterprise and between supplier and manufacturer enterprises makes it impossible for enterprises to respond quickly to the dynamic needs of consumers, and there will also be procurement and logistics.[11]

At the same time, since the information is not completely accurate in the transmission, it is very difficult to establish a mutual trust mechanism between enterprises. If a dispute occurs between the supply chain member enterprises, how to effectively produce evidence and pursue accountability will be a problem. In the 21st century, the global division of labor has become a major trend, the supply chain of modern enterprises has become more complex, and there are more and more cross-border supply chain management, and the collaboration and response time of the supply chain have also improved.[2]

Capitolo 2

Project DME

2.1 Reference scenario

Digital Management of Events is a project funded by the Puglia region, aimed at creating a digital platform, which is initially planned to be used in the Apulian market but with the aim of expanding into the national territory. The digital platform is aimed at supporting users and suppliers who want to organize, manage, communicate and participate in various types of events. Having established the vastness of the project, its partners are many: The Politecnico di Torino, the University of Bari, DS Graphic Engineering S.r.l, STRADE S.r.l, Linear System.

The DME project concerns the organization of events, which in the last ten years has had a strong development, giving an important input also in the field of marketing and interaction with companies. This has allowed an increase in the number of professionals required in the sector, and has encouraged their collaboration on the one hand and an improvement in productivity on the other.

In particular, the role that the Politecnico di Torino assumes is that of the Research and Development of the platform in close collaboration with the Linear System company.
The development process has adhered as much as possible to those reference rules of the Agile methodology, in order to carry out the work purely remotely in the most effective way possible both among us undergraduates of the Polytechnic and with the Linear System company. The work of the Politecnico foreseen during the period concerning the thesis work concerns two macro-periods that we can divide into:

- A very substantial part on the development of the Front End and connection to the Back End functions implemented at the same time by Linear;

- Study and analysis of the solutions suitable for the project, based on Hyperledger for the implementation of Smart Contracts.

The sector has seen great growth in the last decade: the organization of events has established itself more and more, not only as an end in itself, but also as a decisive lever in the field of marketing and corporate communication. For example, the demand for professionals in the field of events by companies is constantly increasing, as well as to promote products and services, also to organize corporate team building events, aimed at motivating and uniting their staff. This is not only valid for corporate events (for conventions, team building, incentives, product launches, etc.) but also for private parties (weddings, birthdays, graduations ...) or for events organized by public administrations (concerts, demonstrations, social events, fundraising ...)

2.2 DME Architecture

Digital Management Event is a platform visible to the user as a set of services accessible through the use of the Web. Its architecture can be described as in the following image:

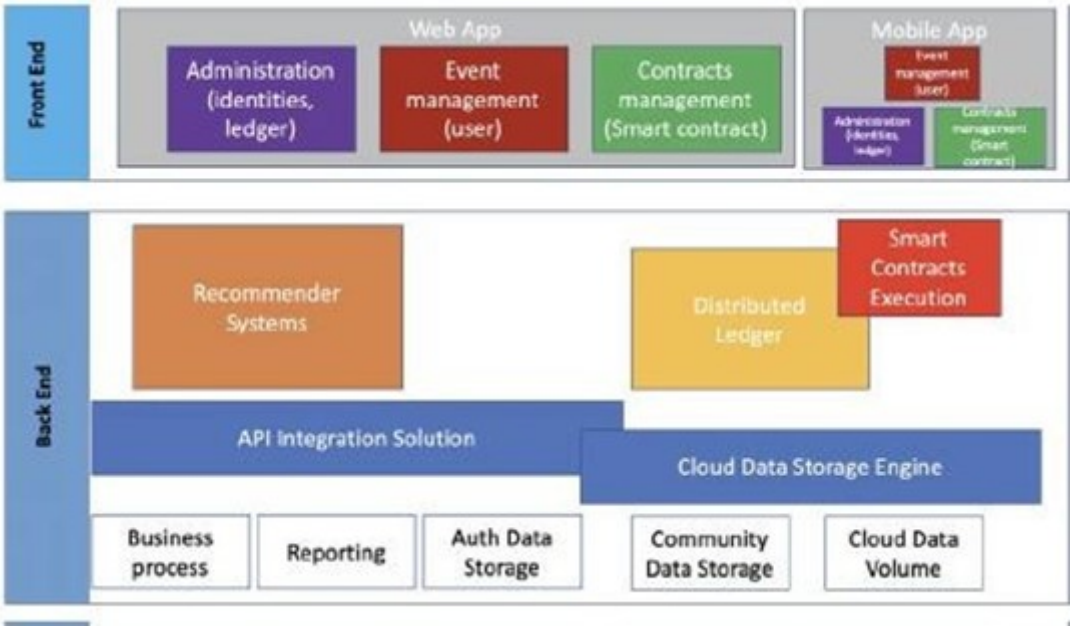

Figura 2.1: Block diagram of the platform DME

As can be seen from the image, there are two main layers:

- Front End Layer: responsible for providing a user-friendly user interface for accessing the various services of the platform. The interface is as responsive as possible to the various types of devices: from the desktop version, to the tablet, to the common smartphone.
  This level simultaneously has the task of communicating with the Back End microservices to retrieve the necessary data and show them to the user user;

- Back End Layer: responsible for communicating with the persistence layer and providing answers to the Front End using a suitable and optimized business logic. The Back End will allow all this by providing the REST APIs that can be contacted by the Front End layer, in a secure manner through the use of the OAuth2 authentication protocol.
  Within the same layer there is a distributed ledger module to implement smart contracts which is made available through a service that exposes special APIs, in order to process transactions and payments between supplier and user;
  Each module offers microservices, which communicate with each other with synchronous or asynchronous protocols.

  The reference elements for the design are as follows:

- Creation of a communication system between microservices, through the components of Business Logic, using a communication protocol between processes. In general, the HTTP / HTTPS synchronous protocol was used for calls to Web API services while the communication between Business Logic components (downstream of the invocation of one or more microservices) takes place asynchronously to ensure the autonomy of the individual microservices;

- Implementation of APIs that allow collaboration between internal and external resources on the platform, taking into account what will be shown to the user;

- Design of a distributed ledger that allows the creation and execution of smart contracts between customers and suppliers;

- Creation of appropriate graphic mockups to establish the aspects of relevance to the overall visual experience.

2.2.1 Objectives and constraints

The organization of events of all kinds (concerts, fairs, weddings, business meetings, birthdays, etc.) is a rapidly growing market, which involves many players and in turn stimulates various economic support activities.
The DME project is located in this context, and aims to create a digital platform to support the organization, management, communication and participation in events. To achieve the results described above, the project sets these objectives:

- Create a platform, based on microservice architecture (MA), of reusable elementary functions to support the definition of events and types of events, creation of marketplaces for the definition of goods and services, definition of users and profiles, horizontal authorization and authentication services, backup and recovery.
- Create a distributed ledger for the definition and management of smart contracts as a core for the management of the customer-supplier relationship throughout the duration of the event.
- Create services for the collection, analysis and modeling of data on events, goods, services, customers. The intelligent analysis of these big data with a variety of Machine learning algorithms to build recurring customer and event profiles is preparatory to the following point.
- Create recommender system to provide old and new customers with proposals for typical events (including organizational model, packages of goods and services, methods of collaboration and interaction) according to the customer profile capable of simplifying the complexity and difficulties inherent in the " organization and management of events, both for private and professional users.
- Creating components for the multi-channel supply (web, mobile, social networks) of the services and processes seen above, with particular emphasis on a high-level user experience, in line with the best offers currently on the market;

2.3 Back-End Layer

The core of the web application as well as the infrastructural layer which is created through cloud infrastructure with proprietary hardware resources of Linear System. In addition to the data saving part and the API mechanism to power the front end, two fundamental modules were initially envisaged for the DME platform:

- The module "Recommend Systems";
- The module "Distributed Ledger / Smart Contract".

The recommender system recommending supplier to organizer and recommending service when creating a new even. This have taken care of automatically and punctually approaching the demand and supply of services relating to the events. Generally, recommendation systems are applied in different sectors, and their purpose is to help people make choices based on different aspects.
Given a person, these aspects can be for example their own history, or the purchases

already made or the positive votes already given, or the preferences of people similar to them. Based on these aspects and on how the recommendations are produced, the recommendation systems are divided into various types.
The recommendation systems can be grouped into three main categories, based on the approach used to obtain the recommendations, plus a fourth which arose following the growing spread of social networks, and which is enunciated last:

- Content-based approaches: the person will be recommended objects similar to those she has preferred in the past;
- Collaborative approaches: the person will be recommended items that people with similar preferences to her have preferred in the past;
- Hybrid approaches: this typology includes the recommendation systems that combine techniques used in the two previous typologies;
- Semantic-social approaches: we consider a set of people and one of objects, where people are connected by a social network and both the latter and the objects are described by a taxonomy.

Depending on the function used, the content-based and collaborative recommendation systems can be:

- Heuristic-based: they usually take various data as inputs and calculate recommendations on the entire dataset of people using similarity metrics;
- Model-based: they build a model using training set-based techniques and validate it on test set data. This model is then used to calculate the score for items not yet recommended to people. Unlike the heuristic-based methods, only a subset of active people of the dataset is given as input to the model, which will produce the forecasts from it.

The Distributed Ledger / smart contract module takes care of saving transactions between customers and suppliers within a distributed ledger and managing contracts digitally by dematerializing them completely.
A distributed ledger manages a log that is nearly impossible to change in a distributed network of nodes. Each node maintains a copy of the register by applying transactions that have been validated by the consensus protocol, transactions are divided into blocks, and each block is calculated as a hash of the previous block.
For business use, the following requirements are generally considered:

1. Participants must be identified / identifiable;
2. Networks must be licensed;

3. High transaction throughput performance;

4. Low latency of transaction confirmation;

5. Privacy and confidentiality of transactions and data relating to commercial transactions;

The Smart Contract is a contract written in a general purpose programming language written in order to automatically determine the execution of the contractual clauses upon the fulfillment of certain conditions included in the contract itself. The application layer chaincode is an application running in a Docker container environment independent of the endorsement node process. The application layer chaincodes are independent of each other and do not affect each other.
The self-execution of the Smart Contract is based on a program capable of:

- verify the clauses that have been agreed;

- check the agreed conditions;

- automatically execute the contractual provisions when the factual information is reached and verified and corresponds to the computer data referring to the conditions and clauses provided for in the contract;

2.4 Front-End Layer

This layer involves the aspects of the interface to users of the Web App that implements DME. The Front End layer provides registration and access services for both the customer and the supplier, in classic registration and accounting forms. It automatically recognizes whether it is a customer or supplier type user, offering a customized page for the different types.
It allows access to the features provided in the analysis phase such as sections for creating events, for managing events, a personalized agenda linked to all events but also to the single event, sections to manage the budget and the contract. Operates both the supplier search functions appropriate to the type of event to be organized and the contract management functions directly within the platform.
As far as the supplier is concerned, the front end layer makes available not only the accounting functions but also those for the creation of a service, allowing media to be loaded by associating it with its own showcase.
The UI was designed through the use in the design and analysis phase of appropriate mock-ups to study the visual composition of the various components within the screen where the interface is displayed.
The principles on which the realization of the User Interface was based are:

- **Simplicity**: since this platform is aimed at any type of user, of any age, the view of the sections has been made as simple as possible;
- **Multi-device access**: since it makes tools available for the daily organization of the event, there has been a lot of focus on making the Web Application responsive to any type of device. Particular attention has been paid to screen formats such as tablets or formats such as iPhones;

During the analysis phase, a possible creation of a mobile application was also envisaged in order to make the same services more usable.
At present it has not yet been realized, however in addition of course to the realization of the application structure of the same, therefore obviously to the creation of an interface that is very close to what is the UI of the Web Application which is visible through a mobile browser , it will be very easy to use the data as the services are made available as described above by the RESTful API.

Capitolo 3

Responsive Web Development

3.1 Introduction

The English adjective "responsive" designates everything that reacts or responds dynamically and appropriately to a stimulus.

The definition applied to the web world consists in adapting, by means of rules, a website based on various factors, such as the size of the screen, platform or orientation of the device. Responsive Web Design is one of the main trends in web development, it is essential for creating web applications that are capable of satisfying the diversity of devices used for web browsing. The first to write about Responsive Web Design was Ethan Marcotte, in 2010, in *"A list Apart"*:

Designers have craved printing because of its precise layouts, complaining about the various contexts of use of their web users that compromise their designs. Ethan Marcotte argues that we need to change the way we think about our designs to overcome these limitations: using flex grids, flexible images and media queries, he shows us how to adopt the ebb and flow of things with responsive web design. [18]
In the last ten years it has become the *de facto* standard used to adapt web pages to a large number of devices, but although it has improved the usability of web pages it is not without limitations and drawbacks, in fact, developers must manually design web layouts for multiple dimensions and implement fitting rules. [12] The RWD is made up of many specific rules, one of these consists in considering the measures in a relative sense, and not fixed measures in pixels, which, on the other hand, do not ensure flexibility of the layout, as the HTML element will always keep the same size. switching from a view on large displays to mobile devices, or tablets. A responsive website is an approach to web design that makes the contents

adapt to mobile devices such as tablets and smartphones and to more traditional devices such as computers and desktops, in this way a user-friendly experience is offered to users, which they can use the website regardless of "where" and "when" the customer uses the site. [17] Web Responsive Design is very important for accessibility, which takes into account numerous factors centered on the difficulties of the users as well as the characteristics of the devices.

3.2 Creating a Responsive Site

To create a Responsive site HTML and CSS is needed. As complexity increases, however, the need to involve languages such as javascript increases, whose latest technologies are Angular, React etc, these allow a more precise front-end both in terms of performance and ease of use. [14]

3.2.1 HTML

HTML is a markup language, which allows to arrange the elements within a page. It is considered an ever-changing language that adapts according to the needs of time and technology. In October 2014, the HTML5 version [21] was published as the W3C Recommendation, since that date this version has become the most widely used version of the language. The World Wide Web Consortium (W3C) is a voluntary standard association that decides what is to be presented in the official version of HTML. Voluntary means that despite accepting the rules, it is not necessary to follow them. Before W3C, browsers created their own rules, which made pages very different depending on the browser used. Following W3C's rules ensures that the site is more likely to be compatible with future devices and technologies. HTML 5, which is the version that was created specifically to leave the Web environment alone and become a platform for creating applications, including desktop and mobile. The HTML5 specification in fact consists of the definition of:

- Syntax for markup, adapted to the most modern needs, with specific controls for forms;
- APIs that allow to manage networks, multimedia, and device hardware;

The language is composed of elements, which are components that serve to describe the elements within the page. The indications are contained in an HTML document, called "HTML page", a text file with the extension *.HTML*. In order to create an HTML document, it is necessary first to declare the dipo declaration (doctype) at the top of the file which tells the browser the version of HTML used, then inside there are the **tag**, some labels, which have the characteristic of being inside angled

brackets. Through the tags, therefore, it is indicated which elements should appear and how they will be arranged. [**HTML**].

```
<! doctype HTML>
<HTML lang = "en">
<head> <title> Hello World! </title> </head>
<body>
<h1> Hello World! </h1>
<p> This is our first HTML page! </p>
</body>
</HTML>
```

3.2.2 CSS

CSS, whose acronym represents the initials of *Cascading Style Sheets* represent a complement to the markup language, providing instructions on how web pages should be represented. CSS models the entire layout of a website, from the formatting of the text to the positioning of the elements, to the arrangement that the elements will have with respect to other elements. The term "cascading" determines the order in which styles are applied, from the broader rules to the more precise, with a decreasing importance.

A rule determines the style applied to an HTML element, each rule consists of two elements, the selector and the declaration. The selector is the HTML element, group of elements or their state, whereas the declaration is the quality of the property to be changed, such as color, size, etc. Each property name is followed by ":" and its value, when it is necessary to declare more than one property, this is followed by the ";".

```
    p {
    color: red;
    font-size: 1.5em;
    }
```

Styles can also be applied to a subset of elements, using ids or classes.

```
.nomeClasse{
    color: blue;
    }
#id{
    color: green;
}
```

this particular code will affect all HTML elements with class = "classname" or with id = "id", class and id work similarly.

Like HTML, there are also different versions of CSS, the most recent is CSS3, each version of CSS includes new stylizable components, but the biggest novelty of CSS3 is the introduction of media queries, which allow the page to adapt to different device sizes. Furthermore, as also happens for HTML, not all pages support CSS in the same way, so there may be different renderings depending on the browser used.

The CSS rules can be applied to the web page in various ways: they can be in a separate file, or included in the web page file: in the first case they will be applied to all files, while in the second case they will be applied only to the related page. However, it is a good idea to write the CSS code in a separate file, even if these relate to a single page. To have an even more specific declaration, it is possible to use the "inline" CSS, which are located in the HTML element, however it is a technique not recommended, as it is easy to lose track of the CSS declaration, but it can be used in test cases.

CSS Cascade

CSS follows very detailed rules for applying styles to an element, in particular, if an element has styles that conflict, the cascades determine the order in which the style sheets will be executed. Browsers follow a very specific hierarchy to decide which property to apply to the element:

1. Rules marked important: property followed by! important;
2. CSS online;
3. Rules containing id;
4. Rules containing classes, attributes and pseudo-classes;
5. Rules containing elements and pseudo-elements;
6. Inherited rules;
7. Default values;

The order is called *CSS specificy* and goes from the most specific to the least specific rules, so it will check that the rules considered important are present before those containing id. For each element on the page the browser goes through a hierarchy, in order until it finds a style for each property. In the case of two rules with the same properties but with different values and applied to the same element, the one that is written last in the order of the source code will have priority:

```
h1{
color: blue;
}
h2{
color: red;
}
```

In this case the rule that will be applied will be the second one because it is the last one to appear in the code.

Box Model

An important concept for CSS is Box modeling, which describes the space taken by the HTML element. In the box-model, each model consists of four types of boxes:

- content;
- padding;
- border;
- margin.

Each of these can be modified via CSS.

Figura 3.1: Box model in CSS

In the image in the figure 3.1 starting from the inside we have the area where the actual **content** is located, which can be text, image etc, the horizontal dimensions can be modified with the property `width` and the vertical ones with `height`. The **padding** is an empty space that can be created between the content and the border of the element, by setting a background color for the content that extends from the content area up to the padding. The **margin** is the space that separates the data from adjacent elements, also modifiable in size. Finally, the border is a line of variable size, style and color that surrounds the padding and content area. [**CSSboxmodel**]

The default HTML code, even without CSS, will set default settings, introducing, for example, a certain margin between the two elements. The most common units of measurement are:

- %: The percentage used in comparison to the containing item;
- em: they are relative to the font size of the element (`font-size`);
- rem: they are relative to the font size of the document (`font-size`), this is by far the most common unit of measurement.
- px: pixels are absolute measures, but they are not necessarily consistent between different devices;
- in, cm, mm: physical units of measurements map to pixels, but are not consistent between devices;
- pt: dots, equal to 1/72 of an inch, are very common in printing.

In the box model each element has height and width that can be changed using the CSS. The elements of a block are, by default, 100 % of the width of the containing element, even if the content does not fill the space, while the height is high enough to contain the element. Each element has margins which provide the outer space for the element, and padding which provides its interior space. For Responsive Web Design, the percentage is generally used to set the left and right margins, and padding, in this way the layout can automatically resize to screens of different sizes. The vertical margins (top and bottom) can instead be set by pixels, since they do not need to correspond to the size of the visible area. The 4 sides of the element can have different values both for the margin and for the padding, the dimensions are defined clockwise starting from `top, right, bottom, left`, they can also be defined individually with for example `margin-top`. The auto value for left and right can be used to center an element inside the container.
Borders are placed between the padding and the margin and provide an outline for the element, the declaration must include the line width, style and color. A new property inserted in CSS3 is the border-box, this property can take on two values:

- content-box: the width refers to the content area;
- border-box: the width refers to the box in its entirety, so it includes padding and border.

3.2.3 Bootstrap

Bootstrap is a toolkit, a library that contains a series of components that allow the creation of responsive and mobile first applications. [4]
Bootstrap natively supports responsive design, thus allowing the use of the application on different devices. Bootstrap was born in 2011 with the aim of developing

an internal library that managed the layouts entirely, today it represents the main resource for designing the cabinet of a site, moreover it is possible to easily integrate the HTML5 and CSS3 components. One of the characteristics of this front-end framework is the responsive design, in fact it helps developers in the creation of applications that automatically support different resolutions.

3.2.4 Angular

Angular is a framework used for single-page client applications; it implements basic and optional functionality always as TypeScript libraries. To build a responsive UI, there are several ways, from using media queries and CSS, to using libraries like Angular Material. An useful component for this purpose is the `BreakpointObserver` from the Angular Material CDK library, through this component the elements will be rendered differently based on the target device.

The BreakpointObserver component allows you to define different types of screens:

```
    export declare const Breakpoints: {
    Small: string;
    Large: string;
};
```

After that the screen type can be subscribed like this: Once the BreakpointObserver service has been injected into the component, we can subscribe to the desired screen type:

```
    this.cardsLayout = merge(
      this.breakpointObserver.observe([Breakpoints.Small]).pipe(
        map(({ matches }) => {
          if (matches) {
            return this.getHandsetLayout();
          }
          return this.getDefaultLayout();
        })),
    );
```

In the example we see the definition of the small screen only, but inside it it is also possible to define other dimensions.
To then view the UI intended for mobile devices:

```
    getSmallLayout(): Layout {
    return {
```

```
      name: 'Handset',
      gridColumns: 1,
      layoutItem: [
        { title: 'Card 1', cols: 1, rows: 1 },
        { title: 'Card 2', cols: 1, rows: 1 },
        { title: 'Card 3', cols: 1, rows: 1 }
        ]
    };
  }
  getDefaultLayout(): Layout {
    return {
      name: 'Web',
      gridColumns: 2,
      layoutItem: [
        { title: 'Card 1', cols: 2, rows: 1 },
        { title: 'Card 2', cols: 2, rows: 1 },
        { title: 'Card 3', cols: 2, rows: 1 }
      ]
    };
  }
  }
```

Basically, the functions allow you to change the number of rows and columns dedicated to the card component, depending on whether the device is of the Small type, or of any other type (Default). In order to view the cards through HTML, the colspan and rowspan properties are used, which allow you to asynchronously update the number of columns and rows dedicated to the element.

```
<div>
    ...
  <mat-grid-list [cols]="(cardsLayout | async)?.gridColumns"
  rowHeight="350px">
    <mat-grid-tile *ngFor="let card of
    (cardsLayout | async)?.layoutItem"
    [colspan]="card.cols" [rowspan]="card.rows">
      <mat-card class="dashboard-card">
       ...
      </mat-card>
    </mat-grid-tile>
  </mat-grid-list>
  ...
</div>
```

3.2.5 React

React, a JavaScript library developed by Facebook, aims to simplify the development of visual interfaces by dividing the UI into a collection of components. For the realization of responsive web applications an npm package called react-responsive is used, it combines media queries and breakpoints to define the elements in the DOM that you want to show or hide depending on the device. [10] The package can be installed using the `npm install react-responsive` command, and subsequently imported via `import useMediaQuery from 'react-responsive'`, in this way it will be possible to use the brakpoints in the components as in example:

```
import { useMediaQuery } from 'react-responsive'

export default function Layout({ children }) {
  const isScreenBig = useQueryMedia({
  query: '(min-device-width: 1224px)' })
  const isScreenSmall = useQueryMedia({
  query: '(max-width: 1224px)' })

  return (
    <div>
        {isSmallScreen ? (
          <LeftSidebar small={true} />
        ) : (
          <LeftSidebar />
        )}
    </div>
  )
}
```

Capitolo 4

Distributed Ledger

4.1 Blockchain

Blockchain can be considered as a complete set of solutions to reconstruct trust problems using technical means such as timestamps, asymmetric encryption, point-to-point transmission, consensus algorithms, smart contracts, chained data structures and distributed storage. It is essentially an encrypted distributed instant ledger system.

All participating nodes of the ledger system are connected to each other through the Internet and can communicate directly. Any participating node saves all the information of the ledger. The account book only allows adding new transaction content, and does not allow any deletion and modification. All participating nodes use asymmetric encryption algorithms to ensure data security in the process of transaction information transmission, and use consensus algorithms to confirm new transaction content. Participants link transactions in chronological order into a chained data structure. Any node unilaterally tampering with the transaction records in the blockchain does not affect the contents of the account books maintained by other nodes, so such unilateral tampering is invalid. In addition, any programmable transaction logic after the blockchain 2.0 [13] can be automatically executed when the preset conditions are met in the blockchain system, ensuring the fairness and justice of the transaction rules.

Therefore, the most significant connotation of blockchain technology lies in the ingenious combination and application of a series of mathematical algorithms and information technology to ensure that data cannot be tampered with, the whole process can be traced back, rules can be correctly implemented, and the trust mechanism endorsed by third-party institutions in social transactions can be changed. To form a social trust mechanism that is guaranteed by strict mathematical algorithms and supported by high-tech information technology, everyone

endorses me, and I endorse everyone, so as to quickly and efficiently complete the fair allocation of resources and ensure that all participants have the same goals.

In the case of blockchain systems, two further elements are added in order to make it possible to distinguish the different types: transfers, which determine the characteristics of the transactions managed through the distributed ledger and the assets, which determine the object of the transfer within the system.

4.1.1 Key elements of a blockchain

:

- **Record immutability**: Participants in the blockchain will not be able to modify any transaction that has been recorded in the shared ledger, no attacker will be able to tamper with the transitions. In order to handle an error or to modify a transaction or just a record of it, it will be necessary to register a new transaction. The mechanism by which the register is immutable is that being a chain of blocks, each block is inserted a cryptographic fingerprint of the previous block. It follows that to change a single piece of information it is necessary to change the rest of the chain as well.
- **Distributed ledger**: A database that can be shared, replicated, and synchronized among network members. A distributed ledger records transactions between network participants, such as the exchange of assets or data. The technology can remove extremely inefficient and costly parts of the current market infrastructure, increasing productivity through a wide range of application scenarios.
- **Smart Contracts**: The "smart" contract is stored as a set of rules that will be executed automatically. This can define the conditions for transfers of corporate bonds, include the conditions for travel insurance to be paid and much more.

4.1.2 Advantages

The intrinsic characteristics of a blockchain technology, in particular the peculiarities concerning the security in the operations that can be carried out within it, has led to many companies in the market to invest in this technology. Among these is the IBM which reports: *"Operations are often designed to prevent single points of failure and systemic risks, requiring layers of audits to control risks, but this also results in high internal costs. Data storage systems or third parties may be vulnerable to attack and tampering. Limited transparency can slow down data verification. And with the arrival of the IoT, transaction volumes have exploded. All of this slows down the business and reduces revenue and makes us understand that we need to adopt new methods.*

- **Greater trust**: Due to the characteristics of decentralization, the nodes joining the blockchain can safely transmit point-to-point information without trusting each other. And each transaction on the chain contains the sender's signature, which requires consensus of the entire network and multiple mechanisms. Guarantees that the data cannot be tampered with and cannot be denied.[20]
- **Greater security**: Distributed ledger technology is based on computer algorithms. When a certain part is modified, nodes in the network can be quickly identified by mathematical algorithms. If the system finds that the information of the two ledgers does not match, it considers that the version with more nodes in the same ledger is the real ledger, and the system will automatically discard those few inconsistent node ledgers, which means that if one wants to tamper with the content of data in a distributed ledger, the only way to do it is by controlling the majority of the nodes in the entire system.
- **High availability**: The traditional distributed database uses the active-standby mode. The main database has high requirements and generally runs on a high-end server; the backup database synchronizes the information of the main database in real time. When the primary database is used, the backup database needs to be switched to the primary database. This mode requires complex configuration and expensive construction and maintenance. In the blockchain system, all nodes have equal status, and there is no master-slave distinction. Even if a few nodes fail or is attacked, it will not affect other nodes, and after the problematic node returns to a normal state, all data of the entire network can be restored.
- **Traceability**: Part of the block records the timestamp of the block and all transaction data since the beginning is stored on the entire chain. Based on the immutability of the data and the existence of the timestamp, each node and regulatory agency can easily trace and supervise any transaction.

This makes us understand the current importance of a blockchain, for which giants like IBM are investing heavily in terms of financial resources, as they recognize its advantages.

4.2 Smart Contract

4.2.1 Historical background and definition

In 1994, computer scientist and cryptographer Nick Szabo first proposed the concept of "smart contracts". It predates the birth of the blockchain concept. Szabo describes what is "a series of commitments specified in digital form, including an agreement by the parties to fulfill those commitments". Despite its benefits, the

idea of smart contracts has not progressed — mainly because of the lack of a blockchain that could make it work.

It wasn't until 2008 that the first cryptocurrency, Bitcoin, appeared, with the introduction of modern blockchain technology. The blockchain originally appeared as the underlying technology of Bitcoin, and various blockchain forks have led to great changes. Smart contracts still couldn't be integrated into the Bitcoin blockchain network in 2008, but five years later, Ethereum brought it to the surface. Since then, various forms of smart contracts have emerged, of which Ethereum smart contracts are the most widely used.

The concept of smart contracts dates back to 1994 and was proposed by Nick Szabo, but the blockchain technology required to adopt smart contracts did not appear until 2008, and finally in 2013, as part of the Ethereum smart contract system, smart contracts were first introduced appear.

Smart Contracts are defined as contractual clauses incorporated into the hardware and software in such a way as to make violation almost impossible.

4.2.2 Operating characteristics

A smart contract is essentially a written piece of code that can automatically execute the code content without user intervention when predetermined conditions are met. Smart contracts are usually linked to business activities, and can be deployed to all nodes in the entire network after the logic is written. In Ethereum, the contract is stored on the blockchain, and is loaded and run by the Ethereum Virtual Machine (EVM) when the contract is triggered when the conditions are met; in Hyperledger Fabric, the contract is packaged into a Docker image, and all nodes in the network will build their own based on this A Docker container that initializes the contract at the same time, and then waits to be called. External applications implement various transactions by calling smart contracts. To modify the smart contract, consensus needs to be reached in the entire network, and all modifications will also be recorded in the blockchain. In addition, the state database will also save the modified results for easy and well-documented investigation (for example, the transfer amount of the transfer transaction will be recorded in the blockchain, and the increase or decrease of the account balance will be applied to the state database). Only the query does not involve Consensus and records are not required for modification.

4.2.3 Advantages and disadvantages of smart contracts

Just like any other new system protocol, smart contracts are not perfect. There are several advantages and disadvantages to using smart contracts, including greater efficiency and lack of regulation. Specifically: Some of the main advantages of

using smart contracts include greater efficiency when processing documents. This is thanks to its ability to employ a fully automated process that does not require any human involvement, as long as the requirements outlined in the smart contract code are met. As a result, time is saved, costs are reduced, transactions are more accurate and cannot be changed.

Additionally, smart contracts remove any third-party interference, further enhancing the decentralization of the network.

On the other hand, the use of smart contracts can also cause many problems. Some disadvantages include: human error, difficulty in full implementation, uncertain legal status. While many see the irreversible nature of smart contracts as its main benefit, others see it as immutable if something goes wrong. Because humans make mistakes, even when creating smart contracts, some binding protocols may contain errors that cannot be reversed.

In addition, smart contracts can only use digital assets, and there will be problems when connecting real assets and the digital world. Last but not least, smart contracts lack legal supervision and are only subject to the obligations agreed upon in the code. The lack of legal oversight may lead some users to be wary of transactions on the web, especially if it matters.

The advantages of using smart contracts are greater efficiency in processing transactions, irreversible, secure transactions, and fully automated processes. On the other hand, the disadvantage is the lack of legal supervision, human error and implementation difficulties.

4.3 State of the art and tools used

In the following section we will give an overview of the existing tools used for the development of the web application

4.3.1 Hyperledger Fabric

One of the most popular permissioned blockchain platforms [7], [9] is Hyperledger Fabric. Born from a project of the Linux foundation in 2015, Hyperledger is an open source technology implemented via Blockchain. Hyperledger's purpose is to implement a Business Network Model through its frameworks and tools. Changes in information can be screened based on algorithms to effectively prevent information tampering. For the use we have to make of it, since it will then be aimed at commercial use, we must consider the following requirements:

- identifiable/identifiable participant.
- Licensed network.
- The performance of transaction throughput is higher.

- Low latency ensures transactions.
- guarantees the confidentiality and privacy of transaction data.

Hyperledger Fabric is highly flexible, confidential, resilient and scalable. It has a wide range of applications in education, telecommunications, medical care, and supply chain industries, and has more than 400 projects (including the feasibility study stage) including major customers such as Maersk, Wal-Mart, Lenovo, and Postal Savings Bank.

Fabric has a modular and versatile design that satisfies a wide range of industryuse cases, including finance, banking,health, human resources, insurance and supply chain, and therefore also very suitable for our case study. Where Fabric differs from other blockchain systems is the licensing mechanism. The Bitcoin network is an open, permissionless system that allows users with unknown identities to participate in system transaction verification, that is, bookkeeping. The system employs a proof-of-work (PoW) consensus protocol to verify transactions and secure the network. Members in the Fabric system that can participate in bookkeeping need to register from a trusted Member Service Provider (MSP). Due to the popularity of Bitcoin, Ethereum and some other derivative technologies have grown up, and some innovative companies have begun to pay attention to the underlying technology of blockchain, distributed ledgers and distributed application platforms. However, many enterprises require higher performance that cannot be achieved with permissionless blockchain technology. In addition, in many scenarios, the identity authentication of participants is a core requirement, such as the financial field.

For enterprise use, there are the following comparative requirements: The identity of the participants is clear and identifiable Access to the network must be permissioned High performance, concurrency Low latency of transaction confirmation Suitable for private and confidential transactions in business scenarios. Based on this, Hyperledger Fabric came into being. Hyperledger Fabric is an open source, enterprise-grade permissioned distributed ledger technology platform. Meanwhile, Fabric's smart contracts are the first to support all general-purpose programming languages such as Java, Go and Node.js.

The application client operates the blockchain system services through SDK, API or events, including identity management, ledger management, transaction management, deployment and invocation of smart contracts. The blockchain system includes service components such as member management, consensus service, chain code service, and security encryption. The Fabric system adopts a modular design, and each component is pluggable. For example, the ledger data can be stored in various formats, and the consensus mechanism can replace.

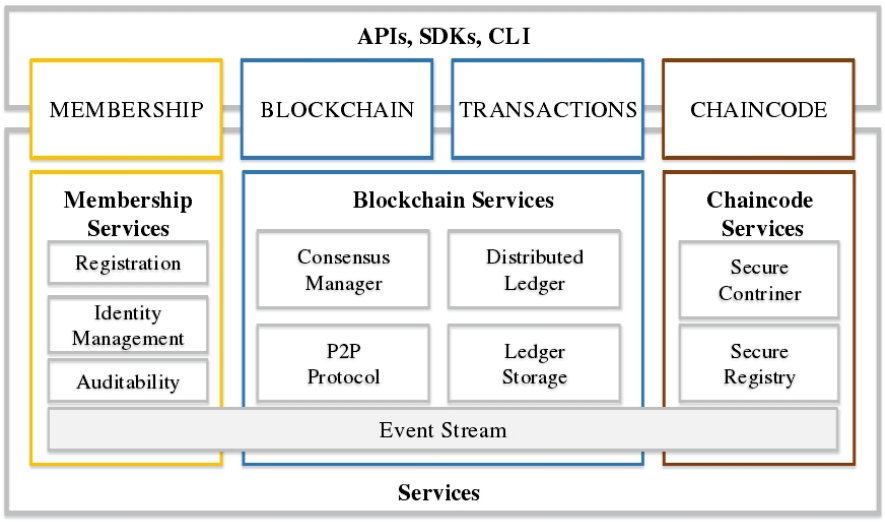

Figura 4.1: Hyperledger Fabric architecture

Hyperledger Fabric architecture consists of three main parts: Membership Servers (MSP), Blockchain Servers, and Chaincode Servers.

Membership Servers(MSP)

Registration, Identity Management, Auditability and other services. The Fabric alliance chain provides member certificate (Certificate) management services through MSP to determine which nodes on the chain are trusted. All types of nodes and clients must be authorized by MSP before they can join the network. The premise of certificate authorization is to configure Public Key Infrastructure (PKI). PKI is a technology and specification of a set of security basic platforms provided by standard public key encryption technology. The Certificate Authority (CA) on Fabric issues digital certificates to users through PKI and assigns keys to realize node identity authentication and data access authorization management in the consortium network. CA certificates are divided into two categories, Root Certificate Authority (RCA) and Intermediate Certificate Authority (ICA), which constitute a certificate chain of trust.

The certificate categories are shown in Table:

Tabella 4.1: Certificate classification

Certificate name	Issuing authority	Describe
ECert	Enrollment CA	User's real-name certificate
TCert	Transaction CA	Authentication during transaction
TLSCert	TLS CA	Communication Identity Information Certificate

Chaincode

Chaincode is an instruction to define assets and modify the state of assets. After the chaincode is executed, a set of key-value pair data for writing will be generated, submitted to the network, and finally recorded in the ledger.

Fabric's smart contracts are written in chaincode. In most cases, the chaincode only interacts with the ledger's world state database, not the transaction log. Chaincode can be implemented in a variety of programming languages and currently supports Go, Node.js, and Java, among others.

BlockChain Service

It includes services such as P2P Protocol, Consensus Manager, Distributed Ledger, and Ledger Storage. The ledger consists of two parts: Blockchain and State. Blockchain is the so-called chain connected by blocks and is used to record historical transactions; State is a database in the form of Key-Value, corresponding to the current network. The latest world state of, for each transaction of the same key, the state needs to be updated.

Asset

They represent the objects at the center of the exchange between two entities. Fabric's ledger records the status of assets. Assets can be tangible assets, such as real estate, equipment, etc., or intangible assets, such as intellectual property rights, claims, etc. In the modules of this article, assets are transaction data. Hyperledger Fabric allows you to modify assets using transactions. They are represented as a collection of key-value pairs, as the state record changes, it is written to the channel's ledger as a transaction.

Ledger

Each asset transaction generates a data record of asset key-value pairs, and these key-value pair data are finally submitted to the ledger in the form of new additions, updates or deletions. The ledger stores these transaction records in blocks and in

an immutable order, and maintains the current asset state in a state database. For each channel there is a register and consequently, each peer keeps a copy of the register for each channel of which it is a member. Some features of the ledger:

- The transactions contain the signatures of each peer for approval.
- Transactions consist of versions of keys / values that have been read in chaincode (read set) and keys / values written in chaincode (write set).
- The validated and executed transaction is immutable. By the very characteristic of the ledger.
- Transactions are sorted into chunks and transmitted to peers in the channel.
- Retrieve and update the ledger using key-based lookups, range searches, and combined key searches.
- Before inserting a new block, a version check is performed to ensure that the states of the resources that have been read have not changed since the chaincode code was run.
- The ledger of a channel contains a configuration block that defines policies, access control lists and other similar information.
- Peers validate transactions with approval policies and enforce established policies.
- Channels contain Membership Service Provider instances.

Privacy

Hyperledger Fabric uses an immutable ledger on each channel, as well as chaincodes that can manipulate and modify the current state of an asset. The ledger exists within the scope of the channel, and if each participant is on the same public channel, the ledger can be shared across the entire network; the ledger can also be privatized, allowing only specific participants to hold a copy of the ledger.

To achieve ledger privatization, the involved participants need to create a separate channel that isolates their transactions and ledger from the rest of the network. Install chaincode only on nodes that need access to asset state to perform read and write operations, if chaincode is not installed on a node, then the node will also not be able to interact with the ledger. Thus, channels keep transactions private from the rest of the network, while collections keep private data within subsets of organizations in the channel. To further hide the data, the values inside the chaincode can be encrypted (partially or totally) using standard cryptographic algorithms, such as AES, before sending the transactions and adding blocks to the ledger. The encrypted data recorded in the ledger can only be decrypted with the corresponding key, and the user uses it to encrypt the data.[15]

Consensus

In distributed ledger technology, transactions must be written to the ledger in the order in which they occur, even though they may be in different sets of participants in the network. To do this, an order of transactions must be established, and a method must be in place to reject illegal transactions that are erroneously or maliciously inserted into the ledger. In short, a consensus mechanism for verifying the correctness of transactions needs to be established.

Consensus is formed when the order and results of all transaction submissions in a block pass the agreed-upon policy checks. Detection is done during the life cycle of a transaction, and endorsement policies are used to specify which members must endorse a transaction class, and these policies are enforced and maintained through the system chaincode. Before committing, peers invoke the system chaincode to ensure that sufficient endorsements exist and that the origin is correct. Each time a version check needs to be performed in advance to see if it is consistent with the current version, then the transaction block is submitted to the ledger, and the current state of the ledger is approved or agreed. This check avoids threats that could compromise data integrity, such as double-spend operations and others, and allows functions to be executed against non-static variables.

In addition to version checking, endorsement and validity, authentication is performed in both directions of the transaction flow. Access control lists are implemented at the hierarchical layers of the network, and when transactions need to pass through different architectural components, the payload must be repeatedly verified, signed, and authenticated. To sum up, consensus is not limited to validating transactions; It is a by-product of the ongoing validation of transactions as they go from proposal to commitment.

Composer

Another framework used by the Hyperledger ecosystem is Composer. The purpose of this tool is to speed up the development process and make it easier to integrate blockchain applications into existing business systems. The Composer allows you to model a business network and integrate existing systems and data into the blockchain application. Hperledger Composer supports existing Hyperledger Fabric blockchain infrastructures, which support consensus protocols to ensure that transactions are validated, according to the policies of the business network participants.

Hyperledger Composer is a programming model that includes a modeling language, a set of APIs for defining and developing business networks and applications that allow participants to send transactions to exchange assets.

Status report of the Blockchain

The blockchain ledger stores transaction data within the network, and the state database stores the current values of participants and assets. There is a state and a ledger on a group of peers, and the ledger of each peer node is always consistent through a consensus mechanism centered on the ordering node.

Connection Profiles

Through the configuration file we can make Hyperledger Composer connect to the Fabric blockchain network and interact. Each type of execution runtime has different configuration options. Connection profiles are referenced by name (in code and on the command line), and connection profile files (in JSON format) are parsed from the user's home directory.

Asset

Assets also in this case are tangible or intangible items, services or properties, and are stored in registers. Assets can potentially represent anything within a business network, documents, real objects or patents. The asset must have a unique identifier, but may contain any properties that need to be defined and can also be linked to other assets or participants.

Participants

Is a member of a business network, can own assets or initiate transactions, must have a unique identifier, and can also contain other optional attributes. A participant can have one or more identities.

Identity

Different actors in a blockchain network include peers, orderers, client applications, administrators, etc. Each of these actors, inside or outside the network, is able to use the service's active element, has a digital identity encapsulated in an X.509 digital certificate, and these identities are really important because they determine the exact rights to the resource and the ability to participate in access rights to the information that the user has in the blockchain network.

Business Network cards

Business Network Card is a feature that appeared in Hyperledger Composer V0.15.0. A Business network card consists of an identity, a connection profile, and metadata; the latter optionally contain the name of the business network to connect to.

Starting from this version, whether to access the Fabric blockchain network or access the business network, the card must be used.This card simplifies the process of connecting to the business network, and extends the concept of identity outside the network into a "container", in which each is associated with a specific business network and a profile for the connection.

Transactions

It is a request, which is used to execute a function interact with assets on the ledger, which is implemented by chaincode, for example the transfer of ownership from one participant to another.

Query

Querying data in the blockchain can be done through the Composer API. As long as the business network and related variables are defined.

Events

Events are defined in a business network in the same way that assets or actors are defined. Once the event is defined, it can be triggered by the transaction handler. Applications can subscribe to these events through the composer-client API.

Access control

Access control in Composer is done through participant and business network ACL files. It provides declarative access control to domain model elements. By defining ACL rules, which users/roles are allowed to create, read, update or delete elements in the domain model of the business network. Access control to a business network is defined by an ordered set of ACL rules. Rules are evaluated in order, and the first rule that matches the condition determines whether access is granted or denied. Access is denied if no rule matches.

Chronology

The history is a specialized register that contains the transactions carried out successfully, including the participants and the identities who carried out them. Transactions are recorded as assets named HistorianRecord, which are defined in the Composer namespace.

Hyperledger Fabric SDK

The SDK offers an abstraction layer used by client applications to interact with the Hyperledger Fabric blockchain network.

In particular, the use of the Java SDK allows Java applications to manage the life cycle of Hyperledger channels and also the user chaincode. In particular, the great utility of this SDK is that it provides many features for the execution of the user chaincode, query blocks, pre-packaged transactions on the channel and monitor for channel events. For these reasons, great use has been made in the creation of the DME ledger layer.

However, the SDK does not provide any means of persistence for applications, but this is by choice as it leaves the management to the creation of channels that can be serialized through Java serialization. Therefore, the application that uses these SDKs requires the implementation of management policies for data migration.

Certificate authority

Of considerable importance is the fact that the SDK provides a certificate authority, so that the client can communicate securely with the ledger layer. However, the SDK is not dependent on a particular implementation of a certificate authority, in fact other certificate authority implementations can be used. This was really important for the creation of the user registration interface within the blockchain.

SDK Gateway

The Fabric Gateway SDK has also been widely used. This made it possible to connect to the blockchain network and allowed the application to interact with it. All this thanks to simple APIs that allow you to submit transactions to the ledger or to query the contents of the ledger with a truly minimal code. The main elements used in the Fabric Gateway were.

- **Wallet** which defines the interface for storing and managing the identity of users in the Fabric network.
- **Gateway** using the createBuilder() method, it configures the connection used to access the gateway by setting the wallet and network configuration.

Dependencies of the SDK

The SDK depends on third-party libraries which for its proper functioning must be included in the classpath when using JARs.

In particular to build the project you have to add:

- A version of JDK 1.8 or higher.

- a version of Apache Maven 3.5.0.

Then to launch the Fabric CA or for Unit tests that have not been implemented in DME, it must be integrated:

- Docker version 18.03.
- Docker Compose 1.21.2.

4.3.2 Spring

Spring Framework provides a comprehensive programming and configuration model for modern Java-based business applications, on any type of deployment platform.

A key element of Spring is application-level infrastructure support: Spring focuses on "plumbing" business applications so that teams, just like in our case, can focus on application-level business logic, without unnecessary ties. with specific distribution environments.

[6] In particular, the goal was to make the distributed Ledger available within a RESTful Web service like other applications and Spring was just the framework suitable for our purpose.

Spring Boot Among the various choices that Spring makes available, the particular Spring distribution used was the Spring Boot, whose main features are:

- Directly embed Tomcat, Jetty or Undertow (no need to distribute WAR files);
- Provide "initial" dependencies to simplify build configuration;
- Automatically configure Spring and third-party libraries whenever possible;
- Provide production-ready capabilities such as metrics, health checks, and outsourced configuration;
- No code generation and no XML configuration requirements;

Capitolo 5

Experimental phase and development of the Smart Contract

5.1 Preliminary stages

5.1.1 Choice of technology

The choice to use Fabric, in its pure version, was not immediate. In the firstinstance, the use of the Fabric Composer was evaluated, as it guaranteed the advantages in terms of development, asreported by the documentation. As previously mentioned, Composer allows for faster development as it makes the integration and implementation of a blockchain easier. And also Fabric is the most active platform project among the current 15 Hyperledger projects **Composer installation prerequisites** For the implementation of the blockchain, the decision was made to develop it on a Linux environment. Therefore, to install the Composer prerequisites, all the tools necessary to develop in this operating system had to be installed, which specifically is the Ubuntu 18.04 LTS version.

The tools that have been installed are the following:

- Node v10.16.0;
- Npm 6.9.0;
- Docker-Compose: Version 1.8;
- Docker Engine: Version 17.03;
- Python: 3.5.2;

The installation first undertaken individually for each tool could give rise to some problems, so if you have chosen the Linux Ubuntu 18.04 LTS version, there is a script made available by the official Hyperledger Fabric Composer documentation that allows the automatic installation of all these tools.
For the execution of the same script, the commands useful for executing it are shown here:

-curl -O https://hyperledger.github.io/composer/v0.19/prereqs-ubuntu.sh

-chmod u + x prereqs-ubuntu.sh

Composer installation and configuration Below we have installed a set of client tools that are useful for developing through Composer and useful tools for running the Rest server on your machine as a RESTful API.

The essential CLI tool is installed with the following command:

-npm install -g composer-cli@0.20

While to install the Rest server:

-npm install -g composer-rest-server@0.2 Very useful in the composer environment is a browser application that is used to modify, manage and modify the business network, this is called Playground and can be run locally during development.

To install it:

-npm install -g composer-playground@0.20 **Fabric installation** Together with the installation of these components, the installation of the Hyperledger Fabric is essential. This provides a set of commands that are used to install a local Fabric runtime.

The official Fabric documentation provides a script for downloading it. To download and run it, we report the commands that have been launched:

curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz

- tar -xvf fabric-dev-servers.tar.gz You have to go back to the directory where you are downloaded the script and execute it with:

- /.downloadFabric.sh

Experimentation of use of a Composer solution Before a possible implementation of our solution for DME, the use of Composer was evaluated and tested by exploiting the demos made available by Hyperledger itself.

The script available after downloading is very useful as it not only allows you to do this type of evaluation in the experimental phase but is also very useful in the development phase as we will see later.

The ./startFabric.sh script takes care of the installation and configuration not integrated into the network itself. It obscures the basics of the network at that level and that's fine for most application developers. They don't necessarily have to know how network components actually work in detail to build their app. In particular in its default configuration:

- creates a channel and joins the peer to the channel;
- installs the fabcar smart contract on the peer's file system and instantiates it on the channel(the instance starts a container);
- call the initLedger function to populate the channel register with 10 unique cars;

That is, it is initialized and consists of a single peer node configured to use CouchDB as a state database, a single node, a certification authority (CA) and a CLI container for executing commands.

As a second step, the script is run to create a PeerAdminCard that contains a single identity and is mapped to a participant, in this case the admin, and allows the owner of the business network card to perform transactions in the business network.

Subsequently, a tool was used to help us in the generation of the application, the Yeoman (npm install -g yo), which in particular was used to generate the structure of the Business Network Definition(BND), which defines the data model, the transaction logic and the rules for access control for the blockchain. To do this, you must specify in the yo commands the name of the business network, the name and e-mail of the author, the type of server, the namespace and specifying that you do not want to generate an empty BND we will have a skeleton composed of assets, participants , transaction and the rules of access control, query and event.

The model is represented in a ".cto" file which contains the class definitions for all assets, participant and transaction in the network. It also contains an access control document (permissions.acl) with basic rules, a script (logic.js) that contains the functions to perform transactions, and the package.json file that contains the business network metadata.

For the purposes of testing the platform, the model was therefore modified by adding a "Trader" participant, a "Commodity" asset and a "Trade" transaction that indicates the relationship between Trader and Commodity.

The logic.js file describes the transaction functions contains the JavaScript logic to execute the transactions defined in the model file. The "Trade" transaction has been defined in such a way that it accepts the identifier of the "Commodity" asset to be exchanged, and the identifier of the "Trader" participant to set up a new owner.

The next step was to create a business network archive to make it

```
/**
* Track the trade of a commodity from one trader to another
* @param {org.example.mynetwork.Trade} trade - the trade to be process
* @transaction
*/
async function tradeCommodity(trade) {
    trade.commodity.owner = trade.newOwner;
    let assetRegistry = await getAssetRegistry('org.example.mynetwork.
    await assetRegistry.update(trade.commodity);
}
```

Figura 5.1: Code related to the function of the transaction

contain the BDN and finally deploy the business network. Administrator information is normally required to create a PeerAdmin identity, with the privileges to install the chaincode on the peer and launch it on the channel. The PeerAdmin identity was created during the installation phase of the development environment. The deployment of the business network in Hyperledger Fabric requires it to be installed on the peer, after which it can be launched, and a new participant, the identity and its card must be created by the network administrator. Finally, the network administrator's business network card must be imported which can be pinged to verify that the network is active.

Reasons why it was decided to abandon Composer Although the advantages of using Composer are truly remarkable, as can also be seen from this experimental phase, a careful analysis was made before proceeding with developments with this technology.

In general according to the current state of the art, Composer has been regarded as deprecated. But why is it choose such if the advantages are many and varied? Basically it is that the major multinationals such as IBM, which is investing heavily in blockchains, have abandoned it. Consequently, if the multinationals abandon this track, the whole market moves accordingly. IBM's reasons for this choice can be summarized as follows:

- Composer is designed to support multiple blockchain platforms, plus Fabric. There are two programming models - Fabric programming model (chaincode) and Composer programming model (business network). This requires the user to choose one of two programming models, which are almost completely different. In this particular case, selection is a bad thing, and many users choose not to use the "optional" part after the initial exploration or POC phase.
- This design has also made it much more difficult for IBM to adopt and expose the latest Fabric capabilities. It is extremely difficult for IBM to keep up with

and align with the latest Fabric features. This meant that users understandably stopped using Composer and went back to developing with Fabric instead.

Finally, the comparison with Linear further pushed us to take the choice of starting development simply by using a solution that involves the use of only Hyperledger Fabric.

5.2 Smart Contracts in DME

The innovation of Digital Management of Event does not only concern the digitization of a system that allows to have various functionalities at an organizational level, but also concerns the possibility for the user to manage the entire process: that is, to choose a location, contact and purchase a service from the various suppliers who will be registered on the platform. All this will then be digitized and consequently the possible constraints relating to privacy and digital contracts will be implemented with Smart Contracts. The closing of contracts directly within the platform through "smart contracts" is destined, as amply demonstrated by the market analysis, to increase strongly in relation to the needs of manufacturing companies to implement solutions for the digital transformation of companies, on the one hand, and for the multidevice and mobile ubiquity of the services to be provided.

The use of Smart Contracts is supported by the use of a distributed ledger to allow communication between the various players in the supply chain.

The "Distributed Ledger / smart contract" module takes care of saving transactions between customers and suppliers within a distributed ledger and managing contracts digitally by completely dematerializing them. The smart contract contains pre-defined status information, conversion rules, contract execution conditions, and contractually agreed operations. After the node is verified, it is recorded in the distributed ledger. The blockchain can monitor the status of the entire smart contract in real time, and start and execute the operations specified in the contract after confirming that the specific contract execution conditions are met.

To make the experience complete and innovative, the use of this technology is essential. However, although it exists and is used in some branches, there is still not a very wide use of this ecosystem based on ledgers and smart contracts, so it can still be considered an implementation in an experimental phase that will certainly be increasingly used over the years. Currently, in fact, there are no academic documentation about it so everything that has been implemented was based on official documentation of the various technologies used and on the study and analysis of what could be more suitable as solutions for our DME platform.

5.2.1 Network initialization

As for the initialization of the network, our DME application does not require a particular configuration of the blockchain network, we have therefore chosen to use the default initialization script of Fabric: /startFabric.sh. This command will create a blockchain network consisting of two peers and a certificate authority. It will also install and instantiate a TypeScript version of our DME smart contract that will be used by our application to access the ledger. So we have only configured the start within the network of our particular chaincode. The scheme of the network according to which DME should operate can be summarized with an outline reported by the Hyperledger documentation.

5.2.2 Drafting of the Smart Contract relating to DME

First of all it was necessary to choose in which language to develop the chaincode. Hyperledger offers the possibility to choose between different types of languages for writing the Smart Contract, including: GO, JavaScript, TypeScript, C ++. Since in terms of performance or difficulty there is no advantage of one over the other, the choice fell on TypeScript only for reasons of confidence with the language itself.

Figura 5.2: DME blockchain network

The Hyperledger Contract API was used, extending the Contract class with our DMEContract class. The asynchronous methods that are part of the logic of the contract have been implemented within, these will be listed below:

- **initLedger:** takes Context, another Hyperledger Contract API that makes the transactional context available for the execution of transactions. As can be seen from the name itself, the initLedger initializes the register, in particular by reconstructing the Reservations (which we will see later, will be the element that will be treated and recorded within the ledger) within our ledge.

- **createReservations:** this method is used, passing all the parameters necessary for the Reservation, to create a new reservation and insert it into the ledger, always using the context method: ctx.stub.putState (prNumber, Buffer.from (JSON.stringify (pr))).
- **updateReservations:** method similar to the previous one, only it updates an existing reservation within the ledger. Remember that the transaction actually updates the reservation in this case, but the ledger keeps track of all transactions.
- **getReservationsHistory:**this method returns all the transactions linked to a reservation to reconstruct a historical one, taking advantage of the characteristic of the ledger that keeps transaction memory by transaction.
- **queryAllReservations:** utility method in case you need to return all ledger transactions.

Demonstrative example of the code of the createPR method: From the image shown, you can see how

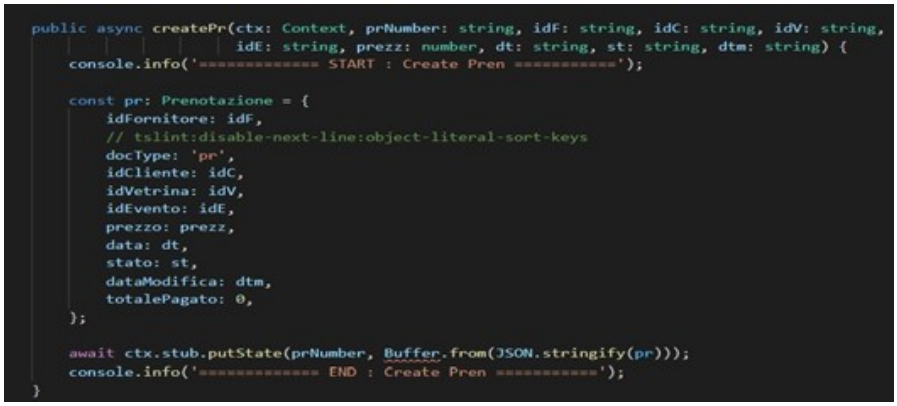

Figura 5.3: Reservation creation method

the Prenotazione class plays a fundamental role. As already mentioned, it is the element that is recorded in the ledger and keeps the information necessary for the operating logic of our DLT. The Reservation structure is defined in an appropriate Prenotazione.ts and imported into the chaincode. The fundamental fields of this structure are:

- **idCliente:** stores the id of the customer who carries out the transaction;
- **idVetrina:** the id of the showcase on which the customer makes the reservation, which is linked to the supplier.
- **idEvento:** links the booking to the customer's event.

- **prezzo:** the total to be paid for the provision of the service to be respected by the customer.
- **data:** delivery date of the supply, to be respected by the supplier.
- **stato:** keeps track of the status of the reservation made in the ledger. A reservation can be in the status of only "booked" or with a balance not fully paid, in the case of payment in installments;
- **totalePagato:** it keeps track of how much the customer has paid. Fundamental for the customer's obligation to fulfill the contract. Only when the total paid coincides with the price the customer has fulfilled the contract..
- **dataModifica:** indicates the last change of status of the existing Contract.

So this is the Smart Contract whose instance will be initialized during the network execution phase when the blockchain is launched within the environment that will host it

5.3 Smart Contracts in Lottery

The traditional model is that lottery players buy lottery tickets, and then transfer the money to the intermediary platform of a third-party payment institution. After the seller issues the order and the buyer confirms that the bet is successful, the buyer notifies the payment institution to transfer the money to the account of the lottery institution. The lottery transaction model supported by blockchain technology is different. Lottery players and institutions can trade directly without going through any intermediary platform.

Here we mocked up a lottery ledger. For the basic data structure of one lottery, we designed like :

- **contractId:** the id of the transaction contract;
- **orderId:** the id of the order which the customer buy the lottery;
- **buyerId:** stores the id of the customer who buy the lottery;
- **sellerId:** the id of the seller which the customer buy the lottery
- **buyerCompanyName:** Purchaser's company name information.
- **sellerCompanyName:** Seller's company name information.
- **buyerTaxPayerIdentificationNumber:** Buyer Taxpayer Identification Number;
- **sellerTaxPayerIdentificationNumber:** Seller Taxpayer Identification Number;
- **address:** indicates the address of the seller;

- **buyerFiscalCode:** stores the Fiscal Code of buyer.
- **buyerAddress:** indicates the address of the buyer;
- **orderTotal:** how much the buyer should paid for this order.
- **note:** note for this order
- **attachments:** it's form withname:String; uri:String, should include the name and the URI.

5.4 Microservice for Smart Contract

To make the smart contract available to the end user so that he can make his transactions, a microservice has been implemented in Spring Boot, which by exposing RESTful API calls expose the Hyperledger Fabric SDKs to interact with the blockchain .

5.4.1 Services

The services that have been implemented to support are:

- **ChainCodeService**
- **RegisterService**

ChainCodeService The service core is implemented in this class. The fundamental method that is called in each method is the Contract connect (String user).

This method uses Hyperledger's fundamental SDK: the Gateway. Gateway allows us to carry out the primary operations to connect to our blockchain. First of all, it checks the presence of the user within the wallet which guarantees the registration not only on the platform, but also within the blockchain. Passing the configurations of our network (the peer organization, the name of the network, the name of the connection) to the gateway builder it initializes it so that in turn, receiving the name of the channel in which the DME smart contract is initialized, it creates a Network object of the SDK. This finally allows with the getContract method to return our smart Contract DME.
Each of the other methods are used to invoke the methods of creating, updating, and returning the history.
Taking as a starting point the service method for creating the reservation:
note that the previously described connect () method is called, and then the transaction is submitted on the contract by creating a new reservation and setting the status to "created". The data is taken from a DTO for the transaction which is managed and passed to the createPR of the contract.

```
Gateway.Builder builder = Gateway.createBuilder();
builder.identity(wallet, user).networkConfig(networkConfigPath).discovery(true);

// create a gateway connection
Gateway gateway = builder.connect();

// get the network and contract
Network network = gateway.getNetwork("mychannel");

return network.getContract("DME");
```

Figura 5.4: Gateway

```
@Override
public void createTransaction(TransactionDTO dto, String user) {

    try {
        Date dtm = new Date();
        Contract contract = connection(user);
        contract.submitTransaction("createPr", dto.getTransactionId(), dto.getIdFornitore(), dto.getIdCliente(),
                dto.getIdVetrina(), dto.getIdEvento(), dto.getPrezzo().toString(), dto.getData(), "creata", dtm.toString());

    } catch (IOException | InterruptedException | TimeoutException e) {
        e.printStackTrace();
        throw new GenericException();
    }catch (ContractException e){
        e.printStackTrace();
        throw new ClientException();
    }
}
```

Figura 5.5: creation of the transaction

RegisterService The registration service is used when creating a new customer / supplier. During the registration phase, the FE in addition to calling the endpoint relating to the saving in the database of the new user, through the appropriate controller, which will be shown later, the registration service within the blockchain will be called.
This service will receive the user's ID and save it in the wallet related to our chaincode. In short, the wallet is a container of the SDK where a key and value pair is recorded, where the key is the User ID while the value is an Identity. This Identity is formed using another key object, the HFCAClient. The steps are as follows:

- A new RegistranRequest is created by passing the user ID;
- Of this new object, the following are set: the affiliation, that is the department of our blockchain, the enrollment ID, that is that of the user;
- We call the register function of the HFCAClient object, passing the registrationRequest and the admin, this will return an enrollment secre;
- Subsequently, the ID and the enrollmentSecret are passed to the HFCAClient enrollment and this will return an Enrollment object;

- Finally, a new X509 Identity is created by passing the enrollment, with a method of the Identities object that contains the admin's certificate;
- Finally it is added to the wallet;

```
RegistrationRequest registrationRequest = new RegistrationRequest(id);
registrationRequest.setAffiliation("org1.department1");
registrationRequest.setEnrollmentID(id);

String enrollmentSecret = caClient.register(registrationRequest, admin);
Enrollment enrollment = caClient.enroll(id, enrollmentSecret);
Identity user = Identities.newX509Identity("Org1MSP", enrollment);
wallet.put(id, user);
```

Figura 5.6: Register service

These two objects mentioned above are initialized as Java Beans when the service is started using always appropriate SDKs. To use them within the service, the two objects are injected.

5.4.2 Controller

Controllers expose endpoints to serve requests from the FE, call services by injecting them.
RegistrationController Endpoint mapped to "/registration", has a single Post createUser method for creating a new user.
The user id is passed through the RequestBody, which will be created when a new user is created. The circumstance that this id will certainly be unique is exploited, as its uniqueness is managed by the DME database. Here, too, the registration service described above is injected, the creation method is thus invoked within the post, and will return a 200 OK if the registration within the blockchain is successful.

TransactionControllerEndpoint mapped to "/transaction". By means of the injection of the chaincodeService, it allows you to use all its services, and therefore allow to expose them as Web services, essential for the whole platform, an importance that will be discussed later.
The Post methods exposed are as follows:

- **createTransaction:** Fundamental for the whole logic of the smart contract. This controller receives as a parameter the userid of the user who is booking a service and a dto with the main parameters of this transaction. The controller uses the chaincode service to create this new booking linked to the DME Smart Contract within the blockchain. Of fundamental importance is the set of the

reservation status, of default, at the time of creation it is set as "created" and this is inextricably linked to the reservation, to the blockchain register and is valid as the status of the contract. To vary this conditioning is possible only through a payment or an interaction with the supplier acting with the blockchain through other calls.

- **transactionHistory:** It receives the userid as a parameter and through this data calls the getAllTransaction of the service passing the userid as an argument. It receives the history of the transactions linked to a single booking of a service made by the customer, ready to be shown in the Front End. In this way the user can analyze all the changes of the states in his contract with the absolute certainty offered by the blockchain and its services, being able to observe the various changes in status by monitoring the time and date
- **getTransaction:** It receives the userid and transaction ID as a parameter. It uses the getTransactionById chaincode method that allows you to view information relating to the single transaction, should it be necessary..
- **getReservationsHistory:**this method returns all the transactions linked to a reservation to reconstruct a historical one, taking advantage of the characteristic of the ledger that keeps transaction memory by transaction.
- **queryAllReservations:** utility method in case you need to return all ledger transactions.

The Get Methods:

- **getAllTransactions:** this method is a helper method. It receives the userid as a parameter and returns all transactions, no longer linked to a reservation, but a list of all transactions linked to the individual user.

The Put methods

- **updateTransaction:** Together with create Transaction, this is the other fundamental endpoint of the chaincode operating architecture.
  This method updates the reservation in the editable fields with a new transaction. It is mainly used for status changes allowed for a given booking.
  Please note that all transactions and therefore changes in the booking status will be visible to both the user and the supplier, thus establishing a solid trusting bond, thanks to the smart contract.

PaymentController EndPoint mapped to "/payment". This controller also uses the chaincodeService, and uses it to record and control payment statuses. At the current stage of development, payments are mocked, but an easy introduction of a module is left that can implement a real payment service, which can be: a banking service, PostePay, PayPal, etc.
The only method is a Post method: pay

It receives as the userid of the user who wants to make the payment and as the body the Data Transfer Object data structure of the Reservation.
The method first of all checks if the payment method is not null, in the negative case a BADREQUEST is returned to the Front End. Then follows the payment management logic and therefore the contractual closing conditions. The last status of the reservation is reconstructed using the user ID and the reservation, from which two fundamental data are processed: the status of the reservation and the total amount paid up to that moment. Therefore, it is verified whether the amount paid in this transaction plus the amount paid up to that moment corresponds to the total amount of the booked service. From here two scenarios open up:

1. If the amount paid plus the sum of the previous amount is equal to the price of the booked service, the status is set as PAID and consequently the date of the last payment and the total paid are updated. At this point, the entire reservation is updated with a new transaction.
2. If the amount paid plus the current payment is not equal to the total amount of the service price, this is considered as a partial sum of the amount due and therefore considered as the payment amount of an installment.
   The status is then set as INSTALLMENT PAYMENT and the total amount paid so far is updated and, as in the previous case, the date of the last payment.

5.5 Overall view of the operation

In conclusion, it can be said that the operating logic of the entire smart contract is distributed among the various services, the controllers, the chaincode and the network.
For our infrastructure, it can be said that the network provides the skeleton of the blockchain, using the standard startup of a Hyperledger network that instantiates our chaincode. This defines the behavior of the smart contract, i.e. the possible transactions on what the asset is, i.e. the Booking.
The services through the SDK communicate with the blockchain and enter the various transactions within the blockchain register.
Finally, the controllers activate the various functions of the services and are activated by the front end as they expose endpoints that are contacted by the web. In addition, it implements validation logics for the various states or payment.
It should be emphasized that any anomalies certainly cannot occur within the blockchain as it is practically unassailable in any way.
But in addition, any insertion anomalies, or attempts to evasion to fulfill the payment will in any case be recorded, therefore at the limit both supplier and customer are protected from this point of view.

Capitolo 6

Future trends

With the development and integration of blockchain technology and e-commerce, a decentralized e-commerce model will emerge. At present, the mainstream e-commerce platforms use a centralized service model. Well-known e-commerce platforms such as Amazon, Ali Baba acts as a trusted third party. At the same time, online payment systems such as Alipay and PayPal also play a third-party role. By integrating blockchain and e-commerce, the centralized mode of e-commerce trading market and online payment system may change. The following will compare the two different models of e-commerce trading platform and payment platform to explore new trends in future development.

6.1 An overview of the e-commerce transaction market

An e-commerce trading platform can be defined as a virtual market for online transactions between buyers and sellers. This virtual market for online transactions can be generally described as having a central management agency and a decentralized management agency (peer-to-peer). For example, third parties such as Amazon and Taobao The e-commerce trading platform is a typical one with the characteristics of a central management structure. The decentralized management structure refers to a market where there is no third-party platform to intervene in free transactions between buyers and sellers. At present, most of them adopt the first and more mature model. The latter has recently been compared by experts and scholars. Attention, its application in e-commerce is also being tried by e-commerce platforms, but there are many disputes.

6.1.1 Third-party e-commerce trading platform

Since the 21st century, e-commerce has ushered in rapid development. Enterprises relying on the third-party e-commerce service platform model, such as Alibaba, Amazon, ebay and other e-commerce companies, have risen rapidly and changed people's lives. The development of the Internet has further optimized the sales process. It improves the response efficiency of the traditional trade supply chain, thereby reducing the cost of commodity circulation, making consumers profitable, and gaining a large number of loyal consumers. At the same time, more and more consumers have changed their identities in the process of experience. In the process of high centralization, the platform brings more value to consumers and terminal manufacturers on the one hand, but also brings many hidden dangers on the other hand. For example: the platform In order to compete for more market share, sellers may spend money on third-party trading platforms to purchase exposure, search rankings, click-through rates, etc. to increase customers, resulting in vicious competition or oligopoly. The need for personnel expansion will increase expenditure costs. In order to alleviate the pressure of expenditure costs, the platform may open more privileges and permissions to sellers to ease the pressure. The continuous occurrence and further fermentation of these situations will inevitably affect the experience of consumers , e-commerce will fall into a vicious circle of development.

6.1.2 Decentralized e-commerce trading market

With the birth and development of mobile social tools, the e-commerce model with a decentralized structure has a new development space. It has the characteristics of low customer acquisition cost and fast promotion speed. For producers, it maximizes economic benefits and maximizes revenue At the same time, due to the lack of reputation guarantee provided by third-party platforms, this idea was difficult to achieve before. However, its concept has also begun to be applied in some aspects, such as on-demand The production mode directly matches the needs of consumers with the machine resources of manufacturers, and has the characteristics of no intermediary, no cost, and high transparency. In the process of transaction, there is no third-party platform intervention to directly conduct transactions. Buyers and sellers in this transaction With both identities in the platform, buyers can become sellers, and sellers can also become buyers. Through constant point-to-point exchange of demand information and completion of transactions. In this decentralized trading platform, the transaction process does not require There is no transaction cost for review, so consumer information is highly confidential, and leakage and abuse will not occur. Manufacturers do not need to worry about the customer base being controlled by third-party platforms, and all customers are in their own hands. Transactions The information of the process is recorded in the

blockchain and traceable to the source, which can eliminate the phenomenon of fake goods, and can detect and punish criminals in the first time.

6.2 Overview of Online Payment Systems

At present, third-party payment monopolizes the vast majority of payments, and central institutions are used to protect online payments between buyers and sellers. The birth of blockchain payment with a decentralized structure has impacted the status of third-party payment, and it will be more efficient in the use process, the customer experience is better.

6.2.1 Centralized online payment system

At this stage, centralized payment systems such as online banking and third-party payment institutions all rely on the bank account system. To perform payment, both parties to the transaction must have bank accounts, and third-party payment platforms such as Paypal must rely on banks for fund settlement. With so much transaction data engraved, it can be said that in order to ensure payment security, the cost of investment is huge, and the process is cumbersome, which leads to low efficiency. Among them, cross-border payment is a typical example. On the other hand, handling fees are generally on the high side, and at the same time, problems such as false transactions, high chargeback rates, and long arrival times have also hindered the development of the cross-border economy.

6.2.2 Blockchain payment

Blockchain payment has changed the original online payment mode. With the research and development of blockchain technology, blockchain payment technology has gradually matured. The digital currency based on blockchain technology has changed the traditional payment mode and is designed to be A trusted digital payment method that does not rely on third parties. It hopes to build a decentralized payment system, and the blockchain payment model has many advantages. First, blockchain payments can bypass banks to reduce unnecessary fees , because of its safe, transparent and low-risk characteristics, it will ensure the security of cross-border payments, improve settlement efficiency, and ensure the authenticity of the transaction process and the integrity of records. Secondly, due to the immutability of blockchain technology All transactions are open and transparent and cannot be changed by both parties, which can solve the problems of false transactions and untraceability that may occur in payment. Finally, a fast foreign exchange settlement platform can be created through blockchain technology. Multiple parties in the joint transaction process conduct real-time settlement and transactions,

thereby greatly improving efficiency and speeding up the turnover of funds. At the same time, through the point-to-point (P2P) network structure, it can achieve instant arrival, convenient withdrawal, all-weather payment and other traditional currencies as payment media. Because of a "centralized" entity (such as a bank), the value transfer of digital currency as a payment medium is point-to-point transfer, which is actually a "decentralized" payment process. This weakens the payment medium function to a certain extent.

Capitolo 7

Conclusions

This paper applies blockchain technology to e-commerce websites, and uses blockchain to store transaction information. Relying on its transparent and non-tamperable characteristics, it meets the needs of both supply and demand for high authenticity of transaction data, and hits the industry's pain points. This article uses common technologies to develop the front end of the website, and uses the Fabric Java SDK to access the blockchain. The technologies used are easy to learn and master, and have high efficiency. The solutions provided in this paper are suitable for different industries, especially applications that require highly authentic and credible information, such as stock securities transactions, product traceability, bank loans, and credit checks.

Blockchain technology is slowly gaining acceptance, and its features such as traceability, tamper resistance, decentralization, and timestamps provide innovative solutions for e-commerce in areas such as reputation, privacy, and supply chain to a certain extent. At the same time, with the establishment of a decentralized e-commerce trading market and the improvement of blockchain technology, a new model of e-commerce will inevitably follow in the future. However, because the current mastery of blockchain technology is not comprehensive enough, the complete ecology of blockchain has not been completed, which also makes the research of this paper in the early stage of analysis, but it is undeniable that blockchain technology will become the future e-commerce. The driving force for development, and some of the problems existing in e-commerce will disappear with the support of blockchain technology. A more open and transparent new e-commerce ecosystem will also be born.

7.1 Future developments

Future research will increase the number of Orderer nodes and reduce the dependence on a single node. Apply Kafka's multi-node mode to achieve high concurrent throughput of the website. In addition, more information will be stored and maintained in the blockchain, to achieve a blockchain system with more optimized functions.

7.2 Final considerations

Based on the requirements and project specifications, the following functions for DME have been implemented and tested:

- Manager / Supplier/ Guests registration
- Authentication
- Creating and editing an event
- Search and book a service
- Viewing and booking of appointments
- Management of guests
- Table management
- Service selection recommendation system
- Creation and management of smart contracts through the use of a blockchain

This project gives an early taste of the analysis and research of the most commonly used techniques in web application development today. The dissertation work is very helpful in applying the knowledge learned in the course, learning new skills and connecting with the real world of modern business.

Bibliografia

[1] Godwin J Udo. «Privacy and security concerns as major barriers for e-commerce: a survey study». In: *Information management & computer security* (2001) (cit. a p. 6).

[2] Ari-Pekka Hameri e Juha Hintsa. «Assessing the drivers of change for cross-border supply chains». In: *International Journal of Physical Distribution & Logistics Management* (2009) (cit. a p. 7).

[3] Michael Rosen, Boris Lublinsky, Kevin T Smith e Marc J Balcer. *Applied SOA: service-oriented architecture and design strategies*. John Wiley & Sons, 2012 (cit. a p. 5).

[4] Jake Spurlock. *Bootstrap: responsive web development*. " O'Reilly Media, Inc.", 2013 (cit. a p. 20).

[5] Tareq Ahram, Arman Sargolzaei, Saman Sargolzaei, Jeff Daniels e Ben Amaba. «Blockchain technology innovations». In: *2017 IEEE technology & engineering management conference (TEMSCON)*. IEEE. 2017, pp. 137–141 (cit. a p. 1).

[6] Yue Hongfei. «National report on e-commerce development in China». In: *Inclusive and Sustainable Industrial Development Working Paper Series WP17* (2017) (cit. a p. 6).

[7] Elli Androulaki et al. «Hyperledger fabric: a distributed operating system for permissioned blockchains». In: *Proceedings of the thirteenth EuroSys conference*. 2018, pp. 1–15 (cit. a p. 28).

[8] Nane Kratzke. «A brief history of cloud application architectures». In: *Applied Sciences* 8.8 (2018), p. 1368 (cit. a p. 5).

[9] Parth Thakkar, Senthil Nathan e Balaji Viswanathan. «Performance benchmarking and optimizing hyperledger fabric blockchain platform». In: *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE. 2018, pp. 264–276 (cit. a p. 28).

[10] Adam Boduch e Roy Derks. *React and React Native: A complete hands-on guide to modern web and mobile development with React. js*. Packt Publishing Ltd, 2020 (cit. a p. 23).

[11] Xiaofang Xue, Junpeng Dou e Yao Shang. «Blockchain-driven supply chain decentralized operations–information sharing perspective». In: *Business Process Management Journal* (2020) (cit. a p. 7).

[12] Laine Zhang Santala Jokinen Oulasvirta A. «Responsive and Personalized Web Layouts with Integer Programming. Proceedings of the ACM on Human-Computer Interaction». In: (2021) (cit. a p. 15).

[13] *ETHEREUM.Ethereum whitepaper[EB/OL].[2020-05-15].* URL: `https://ethereum.org/en/whitepaper/` (cit. a p. 24).

[14] *How to Make your Angular Responsive Web App Design?* URL: `https://aglowiditsolutions.com/blog/angular-responsive/` (cit. a p. 16).

[15] *Hyperledger Fabric Model*. URL: `https://hyperledger-fabric.readthedocs.io/en/release-2.2/fabric_model.html` (cit. a p. 32).

[16] *Microservices$_a$rchitecturestyle*. URL: `https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices` (cit. a p. 3).

[17] *Responsive pixel*. URL: `http://www.responsivepixel.com/` (cit. a p. 16).

[18] *Responsive Web Design di Ethan Marcotte 08 Giugno, 2010*. URL: `https://alistapart.com/it/argomenti/responsive-design/page/3/` (cit. a p. 15).

[19] *What is an application architecture?* URL: `https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture` (cit. a p. 5).

[20] *What is blockchain technology?* URL: `https://www.ibm.com/in-en/topics/what-is-blockchain` (cit. a p. 26).

[21] *wikipedia: HTML 5*. URL: `https://en.wikipedia.org/wiki/HTML5` (cit. a p. 16).