INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

MESTRADO EM ENGENHARIA INFORMÁTICA




Pedro Pinto

PEDRO MIGUEL DA SILVA PINTO

Outubro de 2020

POLITÉCNICO
DO PORTO

The Solid Ecosystem: Ready for Mainstream Web Development?

Pedro Pinto

Supervisor:
Professor Nuno Bettencourt

Mestrado em Engenharia Informática
Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática
Rua Dr. António Bernardino de Almeida, 431, P-4200-072 Porto
October 2020

Dedicatory

I would like to dedicate this dissertation to all of those who have supported me unconditionally during its development, as well as everyone who keeps pushing me to do more and to do better.

Abstract

Companies have been collecting data from its users over the years. This data it is often grouped in places called data silos and may then be used for profit in many ways: building data models to predict or enforce user behaviour, selling their data to other companies, among others.

Moreover, the centralisation of data makes it appealing for people with malicious intentions to attack data silos. Security breaches violate users' privacy, by exposing its sensitive data such as passwords, credit card information, and personal details. One solution to this problem is to separate data from these systems, demanding a shift in the way companies create web applications.

This dissertation explores different solutions and compares them, focusing on a particular project named Solid. Created by the inventor of the World Wide Web, Tim Berners-Lee, Solid is a solution that takes advantage of the power of RDF in order to create a web of Linked Data, introducing decentralisation on software architecture in different layers.

In order to achieve mainstream adoption, various aspects such as the impact of the introduction of this technology have on the user experience and development experience need to be considered.

This dissertation documents the development of a prototype web application built with Solid at its core and compares it with the same application developed using a more traditional stack of technologies.

An analysis was conducted under two perspectives: developer and final user. While in the former it is considered aspects such as development time and documentation diversity and quality, the latter is focused on the user experience.

Resorting to a questionnaire presented to real users, it was concluded that the user experience of some the features of these applications, such as the user's registration and the login process is affected by introducing this type of decentralisation. Moreover, it was also considered the lack of documentation this technology has at the moment, though it has improved throughout the development of this dissertation.

Keywords: Data, Security, Privacy, Decentralisation, Solid

Resumo

As empresas têm coletado dados dos seus utilizadores ao longo dos anos. Esses dados são frequentemente agrupados em locais denominados de *data silos* e podem ser usados para fins lucrativos através de várias formas: construção de modelos de dados para prever ou impor comportamentos nos seus utlizadores, venda dos seus dados a outras empresas, entre outras.

Para além disso, a centralização desses dados capta a atenção de pessoas com intenções maliciosas, que possuem interesse em atacar esses agrupamentos de dados. Falhas de segurança violam a privacidade dos utilizadores, expondo dados confidenciais, como passwords, informações de cartões de crédito e outros detalhes pessoais. Uma solução para este problema passa por separar os dados das aplicação, exigindo uma mudança na forma como as empresas criam aplicações.

Esta dissertação explora diferentes soluções e efetua uma comparação entre elas, com foco num projecto específico denominado de Solid. Desenvolvido pelo criador da *World Wide Web*, Tim Berners-Lee, Solid é uma tecnologia que aproveita o poder de RDF para criar uma rede de informação interligada, introduzindo descentralização nas arquitetures de software em diferentes camadas.

Por forma a conseguir uma adoção massiva, vários aspetos, como o impacto que esta tecnologia tem na experiência de utilizador e no desenvolvimento de *software*, necessitam de ser considerados. Esta dissertação documenta o desenvolvimento de uma aplicação que utiliza Solid no seu núcleo e compara-a com uma outra aplicação desenvolvida com uma pilha de tecnologias mais tradicional.

Foi conduzida uma análise através de duas perspectivas: desenvolvedores e utilizador final. Enquanto que na primeira os aspetos considerados estão relacionados com tempo de desenvolvimento assim como qualidade e diversidade de documentação, a última está mais focada na experiência de utilizador.

Recorrendo a um questionário apresentado a utilizadores que tiveram a oportunidade de experimentar ambas as aplicações, concluiu-se que a experiência do utilizador em algumas funcionalidades, como o registo de utilizador e o processo de *login*, é afetada pela introdução deste tipo de descentralização, ainda que em muitas outras a diferença seja impercetível. Além disso, também foi considerada a falta de documentação que esta tecnologia possui no momento, embora tenha melhorado ao longo do desenvolvimento desta dissertação.

Palavras-chave: Dados, Segurança, Privacidade, Descentralização, Solid

Acknowledgement

I would like to express my gratitude to my supervisor who has been supporting me during this entire process. I would also like to thank the Solid Community for all the help and great insights during the development of this dissertation.

Contents

List of Figures **xv**

List of Tables **xvii**

List of Source Code **xix**

Glossary **xxi**

1 Thesis Structure **1**
1.1 Problem . . . 1
1.2 Objectives . . . 2
1.3 Research Questions . . . 3
1.4 Document Structure . . . 3

2 State Of The Art **5**
2.1 Solid . . . 5
2.1.1 Storage Architecture . . . 5
2.1.2 Resource Description Framework . . . 6
2.1.3 Authentication . . . 7
2.2 Blockstack . . . 9
2.2.1 Storage Architecture . . . 9
2.2.2 Authentication and Authorisation . . . 10
2.3 Elastos . . . 10
2.3.1 Storage Architecture . . . 11
2.3.2 IPFS . . . 11
2.3.3 Decentralised Authentication . . . 11
2.4 Solutions Comparison . . . 12
2.4.1 Storage . . . 12
2.4.2 Authentication . . . 12
2.5 Testing . . . 13
2.5.1 Unit Testing . . . 13
2.5.2 End-to-End Testing . . . 14
2.6 Summary . . . 14

3 Value Analysis **15**
3.1 Function Analysis System Technique . . . 15
3.1.1 Business Perspective . . . 15
3.1.2 User Perspective . . . 16
3.1.3 Value Proposition Canvas . . . 17
3.2 Customer Perceived Value . . . 20
3.3 Summary . . . 20

4 Business Analysis **21**
4.1 Functional Requirements . . . 21
4.2 Non-functional Requirements . . . 21
4.3 Domain Model . . . 22
4.4 Use Cases . . . 22
4.5 Summary . . . 23

5 Design **25**
5.1 Architecture . . . 25
5.1.1 Centralised Approach . . . 25
5.1.2 Decentralised Approach . . . 26
5.1.3 Architecture Comparison . . . 26
5.2 Sign Up . . . 26
5.2.1 Centralised Approach . . . 27
5.2.2 Decentralised Approach . . . 27
5.2.3 Approaches Comparison . . . 28
5.3 Authentication . . . 28
5.3.1 Centralised Approach . . . 28
5.3.2 Decentralised Approach . . . 29
5.3.3 Approaches Comparison . . . 29
5.4 Consult Medical Notes . . . 30
5.4.1 Centralised Approach . . . 30
5.4.2 Decentralised Approach . . . 30
5.4.3 Approaches Comparison . . . 31
5.5 Submit Medical Note . . . 31
5.5.1 Centralised Approach . . . 31
5.5.2 Decentralised Approach . . . 32
5.5.3 Approaches Comparison . . . 32
5.6 Share Medical Exam . . . 33
5.6.1 Centralised Approach . . . 33
5.6.2 Decentralised Approach . . . 33
5.6.3 Approaches Comparison . . . 34
5.7 Solid Applications Interoperability . . . 34
5.8 Summary . . . 35

6 Implementation **37**
6.1 Technology Stack . . . 37
6.1.1 Centralised Approach . . . 37
6.1.2 Decentralised Approach . . . 38
6.2 Communication Standards . . . 38
6.3 Use Cases . . . 38
6.3.1 Sign Up . . . 39
6.3.2 Authentication . . . 40
6.3.3 Consult Medical Notes . . . 40
6.3.4 Submit Medical Note . . . 41
6.3.5 Share Medical Exam . . . 42
6.4 Solid Applications Interoperability . . . 43
6.5 Testing . . . 44
6.5.1 Unit Testing . . . 44

6.5.2 Frontend . . . 44
6.5.3 Backend . . . 44
6.5.4 End-to-End Testing . . . 44
6.6 Summary . . . 45

7 Evaluation **47**
7.1 Indicators and Information Sources . . . 47
7.2 Assessment Methodology . . . 47
7.3 Results Analysis . . . 48
7.3.1 Documentation and Libraries Availability . . . 48
7.3.2 User Experience . . . 48
7.4 Summary . . . 51

8 Conclusions **53**
8.1 Research Questions . . . 53
8.2 Contributions . . . 54
8.3 Results . . . 54
8.4 Limitations . . . 55
8.5 Future Work . . . 55
8.6 Final Remarks . . . 55

Bibliography **57**

A End-to-End Testing **59**

List of Figures

2.1 Solid Architecture (Source [7]) 5
2.2 RDF Triples (Based on [9]) 6
2.3 Authentication Flow with WebID-TLS [11] 8

3.1 FAST applied in a Business Perspective 16
3.2 FAST applied in a User Perspective 17
3.3 Value Proposition Canvas 18

4.1 Domain Model 22
4.2 Use Case Diagram 23

5.1 Centralised Approach Architecture 26
5.2 Decentralised Approach Architecture 26
5.3 User Registration - Centralised Approach 27
5.4 User Registration - Decentralised Approach 28
5.5 User Authentication - Centralised Approach 29
5.6 User Authentication - Decentralised Approach 29
5.7 Consult User's Medical Notes - Centralised Approach 30
5.8 Consult User's Medical Notes - Decentralised Approach 31
5.9 Submit Medical Note - Centralised Approach 32
5.10 Submit Medical Note - Decentralised Approach 32
5.11 Share Medical Exam - Centralised Approach Approach 33
5.12 Share Medical Exam - Decentralised Approach Approach 34
5.13 Consulting Patient's Medical Exam through another Solid application . . . 35

List of Tables

2.1 Comparison of storage characteristics between solutions 12
2.2 Comparison of authentication characteristics between solutions 13

6.1 Technologies used in the centralised approach development 37
6.2 Technologies used in the decentralised approach development 38
6.3 Consult Medical Notes API - Centralised Approach 41
6.4 Consult Medical Notes API - Decentralised Approach 41
6.5 Submit Medical Note API - Centralised Approach 42
6.6 Submit Medical Note API - Decentralised Approach 42
6.7 Share Medical Exam API - Centralised Approach 43
6.8 Share Medical Exam API - Decentralised Approach 43

7.1 User Experience Survey Result 49
7.2 User Experience Survey Result Analysis 50

List of Source Code

2.1 RDF Syntax Example 6
6.1 Excerpt of the returned WebId data representation 39
6.2 Example of a Public Type Index response 41
6.3 Saving a new medical note through a SPARQL Query 42
6.4 Creating an ACL file for the medical exam 43
A.1 System Authentication End-to-End Test 59

Glossary

Backend Component(s) of a system located on the Server side.

Frontend Component(s) of a system located on the Client side.

Jest JavaScript testing framework.

JHipster FullStack framework that allows to bootstrap systems.

JUnit Java testing framework.

NodeJS Backend JavaScript framework.

PostgreSQL SQL Database.

RDF Resource Description Framework.

React Frontend JavaScript framework.

Solid Pod Server that stores the user information and exposes an API so that other components can interact with it.

Spring Boot Java framework.

UI User Interface.

Web Id URI that uniquely identifies someone or something.

Chapter 1

Thesis Structure

Nowadays we live in an interconnected world, where internet may be considered humanity most valuable resource. It is using the internet that we do business, we entertain ourselves, we talk and see our loved ones from all over the world, we educate ourselves, among other activities.

Since the mainstream adoption of the Internet, some companies have acquired a big share of the overall internet users within its products usage, offering, in most cases, a huge variety of services so powerful that some of these users have even built their own social life and businesses on it (e.g., companies like Amazon and Google are the main information vehicle for a considerable amount of the global Internet users). On its Small Business Impact Report of 2008 [1], Amazon has reported that nine hundred thousand jobs have been created by small and medium companies selling on Amazon.

1.1 Problem

With all the power over the users data, big corporations have created ways within its platforms in order to boost their profits. One example is by gathering users' data and storing it in "data silos". This data can then be used to predict or force behaviour on users (e.g. marketing, political influence) or even to simply sell it to other companies. All of this is mostly executed without user consent, or at least without users understanding the consequences implied when they accept the terms and conditions of the different platforms, in order to use them. Not accepting the company's terms means, for the most part of the available products and services, disallows their usage.

Companies are able to profit from data in multiple ways, making it one of the most valuable assets. Some services are free just because of how valuable it is to collect the data the users produce by using it. Recently, some attention has been redirected to data scandals like the Cambridge-Analytica[2], where data belonging to millions of Facebook users was harvested and exploited. These kind of events highlight the importance of privacy and security on Web Applications. Ideally, users would have the option to move away from this "data silos", empowered by the ownership of their own data and having access to fine-grained authorisation layers.

Moreover, often companies rely on third-party services which integrate within their own products. The increased coupling between services, often reduce the overall security of a product. According to the authors in [3], a study conducted with more than thousand Chief Information Security Officer in the United Kingdom and United States of America, revealed

that around sixty percent of the companies have already experienced a data breach caused by their vendor or third parties.

As of May 2018, the General Data Protection Regulation (GDPR) [4] was enforced in the European Union and the European Economic Area, with the main goal of improving transparency over how personal data is handled by services [5]. However, companies developed strategies to overlook this regulation, applying dark patterns that, for instance, utilise implicit consent inferred via non-affirmative actions on the website (e.g. scrolling the website or closing the consent pop-up without providing a response)[6]. Strategies like the previously mentioned prove that GDPR, by itself, may not be enough to protect users and give them back the control over their personal data and that significant architectural changes to applications may be needed.

Tim-Berners Lee, known for creating the World Wide Web, has been working on a new web development framework with privacy at its core, named Solid. This framework enables Web Artisans to build products with the data decoupled from the application itself empowering users with the control over their data and enabling a set of use cases, such as data reutilization[7].

1.2 Objectives

Although being a promising technology given its privacy-focused characteristics, Solid-based architectures represent a major shift on how applications are developed and possibly used, when compared to what users are invited to use at the moment of writing and if the development approach is not well-defined and has the right tools or if, the user experience provided by the architecture itself is not good enough for the users, it will most certainly struggle to achieve massive adoption.

In order to truly achieve privacy and security, decentralisation should be implemented on the different web layers, namely authentication, authorisation and data storage. A proof-of-concept regarding decentralisation within the previously mentioned layers should be designed and implemented using Solid, with the end goal of evaluating the solution according to different metrics related to the entire development process, considering the following set of metrics:

1. Non-empirical study comparing existing approaches;
2. How (architecturally) different from a more traditional development approach it is to build this kind of applications;
3. The development time (man-hours) for this kind of applications when compared to the more traditional and centralised approaches;
4. How is the on-boarding of the user affected;
5. Documentation availability from a developer perspective;
6. If the designed approach is a better solution for privacy and consent problems affecting our society at the time of writing.

For the proof-of-concept, there will be developed two approaches: one that utilises Solid at its core and another built using a current web stack. The domain of these approaches is the Healthcare area, and the users will be able to submit, share and control information with

their doctor. Each of the doctors will, therefore, only have access to the information shared with them and the user should have full control to revoke the authorisation whenever they want to.

1.3 Research Questions

Considering the proposed objectives, a set of research questions were defined which are analysed and answered during this dissertation. The questions are the following:

1. How architecturally different are applications built with Solid at its core compared to those built with a more traditional web stack?
2. Can web social applications built with Solid at its core have the same user experience as others currently built with a more traditional web stack?
3. Do Solid applications require more man-hours to be developed than those built with a more traditional web stack?
4. Is Solid ready to achieve mainstream adoption?

1.4 Document Structure

This dissertation is composed by 7 chapters.

In Chapter 1 (Introduction) it is provided some background that led to the development of this dissertation, as well as explained the problem and the objectives.

In Chapter 2 (State of the Art) it is studied the solutions that currently exist for the present problem, as well as the technologies that support the development of this dissertation.

In Chapter 3 (Value Analysis) it is present the value that Solid can bring from a business and user perspectives.

In Chapter 4 (Requirements Engineering) it is presented the domain of the prototype that will be implemented, as well as the functional and non functional requirements.

In Chapter 5 (Design) it is presented the design of the implemented prototype, where it is included an analysis on each of the use cases.

In Chapter 6 (Implementation) it is presented the technologies utilised as well as the implementation process for each of the use cases.

In Chapter 7 (Experimentation and Assessment) it is presented the indicators and methodologies used to analyse them, as well as the analysis results for each of them.

In Chapter 8 (Conclusions) is finally presented the conclusions of the dissertation, analysed the limitations found as well as the future work.

Chapter 2

State Of The Art

In this chapter we explore different solutions that, similarly to Solid, are working towards returning the control of the data back to the users by offering decentralisation in many of the layers of web applications, such as storage and authentication.

2.1 Solid

Solid is a decentralised platform for social Web applications. Nowadays, most of the popular applications that store users' data (either implicitly or explicitly produced) take ownership over it. Solid is built relying on existing W3C standards, based on the Resource Description Framework (RDF) and semantic Web Technologies, and provides data independence and simple, yet powerful, data management mechanisms[7].

In the Figure 2.1 it is possible to depict Solid's architecture, representing authentication and communications between pod-to-pod and application-to-pod[7].

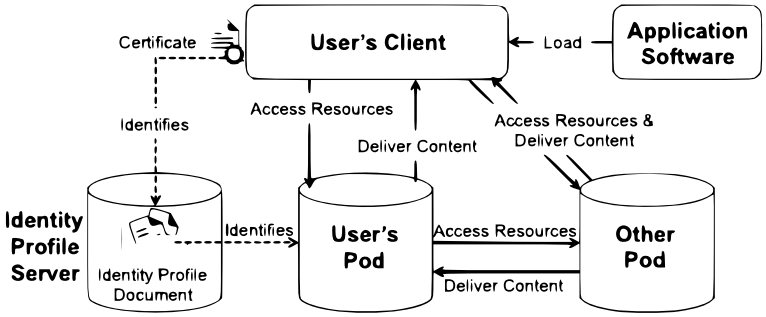

Figure 2.1: Solid Architecture (Source [7])

2.1.1 Storage Architecture

Solid's architecture allows the development of applications that enable users to choose a Pod of their choice (either their own or rely on a Pod provider) in order to grant the applications authorisation to store or read data from it. This fact alone allows users to run from the traditional "data silos" that big companies have been building throughout all these years. Additionally, these pods can offer different granularity for privacy, reliability, availability, legal protection, among others[7].

An important feature achievable with these Pods is data reutilization, which is not a recurrent practice in the current State of the Web. This allows different applications to use data

available on the user's pod - Solid applications are decoupled from the data they need to operate by design[7].

2.1.2 Resource Description Framework

Solid is based on RDF, a standard model that takes part of a family of World Wide Web Consortium (W3C) specifications for data interchange on the Web[8].

This model provides interoperability between applications that exchange information on the Web[9]. It names not only the relationship between things but also the two ends of the link, creating triples (subject, predicate and object).

An RDF model consists of three object types:

- **Resources:** Described by RDF expressions, resources can differ substantially in terms of granularity, e.g. it can be a set of Web Pages, only one Web Page or even a XML element of a page. Resources are named by URIs, optionally ending with anchor IDs;
- **Properties:** It holds a specific meaning, permitted values, types of resources it can describe and even its relationships with other resources. These can be specific aspects, characteristics, attributes or anything else that can describe a resource;
- **Statements:** A statement is composed of a resource coupled with a named property with a value of that property. These three values are named, respectively, the subject, the predicate, and the object. The object can be another resource or simply a string or other primitive datatype defined by XML[9].

These models are used to represent named properties and property values. Its properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. Additionally, these properties may also represent relationships between resources, so a RDF model can therefore resemble an entity-relationship diagram[9].

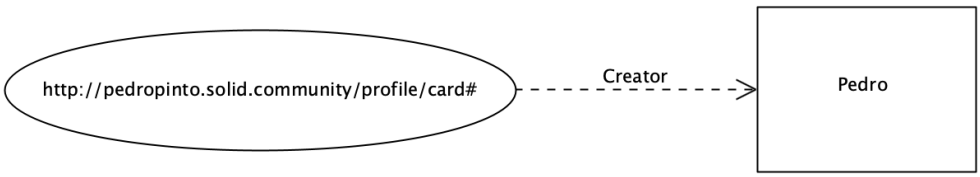

Figure 2.2: RDF Triples (Based on [9])

The Figure 2.3 represents a triple, illustrating that the given subject has Pedro as its creator. With a circle, we have the subject which is Pedro's WebId Profile URI, pointing to his Solid Community Pod. Moreover, the rectangle is representing the object as a literal, whereas the arrow is characterising the predicate. In RDF/XML, this would be represented as showed in the Listing 2.1.

```
1 <rdf:RDF>
2   <rdf:Description about="https://pedropinto.solid.community/profile/
      card#me">
3     <s:Creator>Pedro</s:Creator>
4   </rdf:Description>
```

```
5 </rdf:RDF>
```

Listing 2.1: RDF Syntax Example

Instead of having a literal as the object of this relationship, it is possible to point to other URIs, which will then have their own relationships, creating a web of linked data.

2.1.3 Authentication

The decentralised aspect of this technology has a set of requirements for its authentication mechanisms that are not commonly encountered on traditional platforms. Its main authentication mechanism relies on WebID-TLS, using WebIDs as unique identifiers with the help of cryptographic certificates which, in its turn, allows the users to prove that they are who they say they are[10].

At the moment of writing, Solid's development team is implementing support for WebID-OIDC as another primary authentication mechanism. Additionally, other authentication mechanisms are being investigated, such as combinations of WebID Delegation and the traditional username-password mechanism[10].

WebID-TLS

The WebID-TLS protocol enables secure, efficient authentication on the Web. It enables users to authenticate on any website by choosing one of the certificates presented to them by their browser [1]. WebIDs are URIs with an HTTP or HTTPs scheme which denotes an agent (i.e., Person, Organisation, Group, Device, among others)[11].

In order to achieve a truly distributed social web, we need to ensure that:

- Each person has control of their own identity;
- The Identity is linkable across sites (taking apart of a Web of relationships);
- It is possible to globally authenticate such identities[11].

The Figure 2.3 illustrates how the authentication process works using this protocol. The flow starts with the client opening a TLS connection with the server which authenticates using the standard TLS protocol.

Afterwards, the server may intercept the resource and, if there are authorisation and authentication needs, it may request the client to authenticate itself using public key cryptography by signing a token with its private key and having the Client send its certificate.

The server must then ask the Verification Agent to verify that the WebIds in the WebId certificate do identify the agent who knows the given public key.

Finally, the server can check if one of the verified WebIDs is authorised to request the intended resource and, if granted to do so, may allow the access[11].

[1] https://github.com/solid/solid-spec/blob/master/authn-webid-tls.md

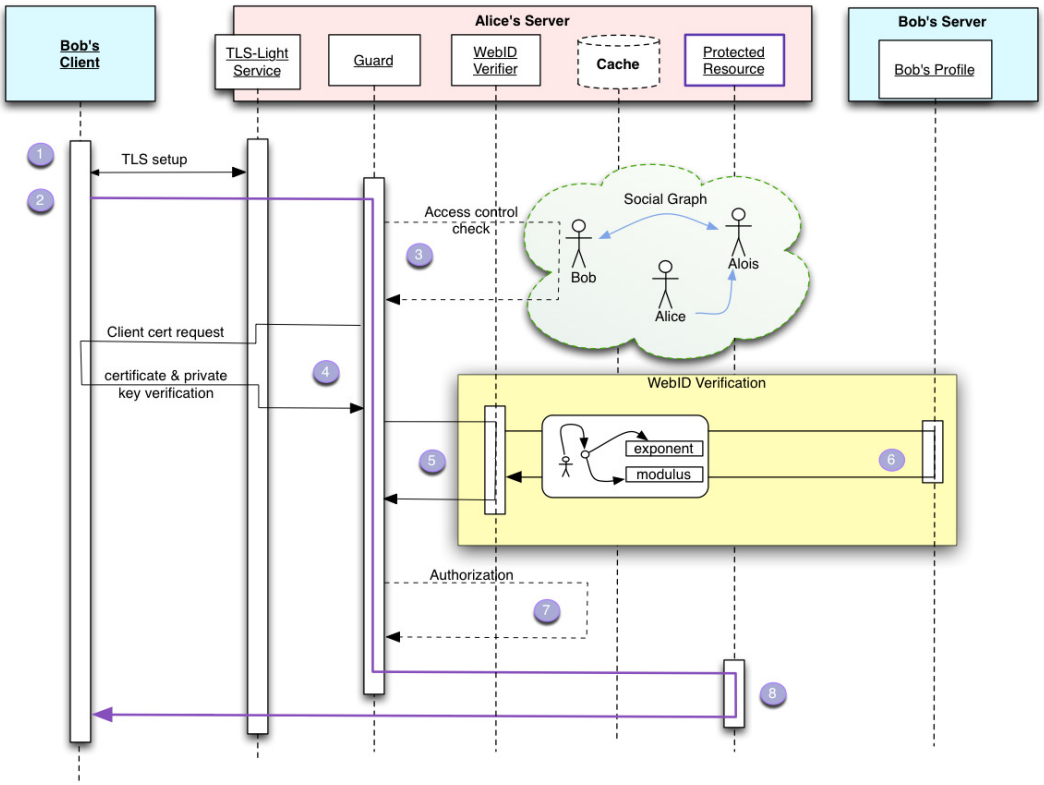

Figure 2.3: Authentication Flow with WebID-TLS [11]

WebID-OIDC

This authentication delegation protocol is suitable for WebID-based decentralised systems such as solid as well as most LDP-based systems. It is built on top of OpenID Connect (OIDC), which in its turn is built on top of OAuth2.

The end result of any WebID-based authentication workflow is a verified WebID URI. While the WebID-TLS derives the WebID URI from a TLS certificate and matches it against the public key, in an agent's WebID Profile, the end result of OpenID Connect (OIDC) workflows is a verified ID Token[12]. The WebID-OIDC protocol takes advantage of the proven security of OpenID Connect while still benefiting the decentralised flexibility of WebID, specifying a mechanism in order to get a WebID URI from an OIDC ID Token.

As presented by Solid's WebID-OIDC Authentication Specification[12], the workflow for this delegation protocol is designed as follows:

1. The user makes a request to a resource;
2. After receiving a HTTP 401 Unauthorised response code, users can select their WebId provider;
3. A local authentication process takes place within the service provider, where users authenticate with their preferred method (e.g., WebID-TLS, password, among others);
4. Optionally, the users are presented with a consent screen in order to verify if they really want to proceed with the login;

5. If the login is successful, the user is redirected to the originally requested resource and the server receives a signed ID Token that was returned on step 3;
6. The server validates the token and extracts the user's WebID URI from it;
7. Finally, by matching the provider URI on the user's WebID profile with the Issuer URI in the ID Token the server can confirm that the service provider is the user's authorised OIDC provider.

2.2 Blockstack

Blockstack is an open-source project that aims to build a decentralised computer network that provides a full-stack alternative to traditional computing using the existing Internet transport layer and underlying communication protocols while removing points of centralisation in the application layer[13]. It follows an end-to-end design principle[13] pushing complexity into the edges (e.g, user devices and user-controlled storages) while keeping its core simple[14]. In order to achieve this, Blockstack designed its architecture with a set of layers, including:

- **Stacks Blockchain:** enables users to control and register digital assets such as usernames and also execute smart contracts and awards developers with tokens for publishing high-quality applications in the ecosystem;
- **Gaia:** Decentralised Storage for applications;
- **Blockstack Authentication:** protocol responsible for enabling authentication decentralisation within applications;
- **Libraries and Software Development Kits (SDKs):** user for developers to build Blockstack applications and interact with the different architecture components easily [13].

2.2.1 Storage Architecture

Blockstack uses a set of components in order to achieve storage decentralisation, namely its Sacks Blockchain and Gaia Hubs. Even though Blockchains would guarantee data privacy and security (among other properties), they can be slow because they require consensus amongst a number of participants in the network. Thus, storing data on a Blockchain would slow down applications besides being an expensive process[15].

In order to solve the performance problem, Blockstack has built a layered architecture that takes advantage of the immutability and security of Blockchain to store essential information, while storing user data in another container. At the base of the architecture there is a blockchain and the Blockstack Naming System (BNS) and the former governs information such as domain names, usernames and application names [16].

On the second layer there is the Atlas Peer Network, which stores all the names in Blockstack (which correspond to routing data in the traditional OSI model). Every node that joins the blockchain obtains an entire copy of the routing data, which is then used associate names with a particular storage location [16].

Blockstack applications can achieve storage decentralisation by taking advantage of the Gaia Storage System (Gaia Hubs). It corresponds to the third and final layer, and it is a system

consisting of hubs that work as a storage resource for decentralised applications. These hubs can be installed locally or in software provider (e.g., Azure, Amazon EC2). Users' data as well as any data needed by the applications in order to work properly are stored in these containers so they achieve much better performance and cost-efficiency than applications built entirely on Blockchains. Moreover, users get to choose where their data is stored, and the system has an API in order for applications to be able to retrieve and store information consistently[16].

2.2.2 Authentication and Authorisation

Blockstack provides users with a universal username that works across all the applications in its ecosystem without the need for passwords, recurring to public-key cryptography and using a locally-running software responsible for handling sign-in requests and signing authentication requests[13]. Moreover, its login is designed to be very similar to third-party authentication techniques that they are familiar with, such as OAuth, and it happens totally on client-side [15]. Each application has a specific private key, which is securely exchanged with the application on each authentication. This key serves mainly three purposes:

- it is involved in the credentials, for that application, that give access to the user's Gaia Hub;
- is used for encryption of the files stored by the application on the user's hub;
- has other cryptography functions that applications are able to use[15].

2.3 Elastos

Elastos aims to create a new kind of internet powered by blockchain technology, where users are able to generate digital assets and generate wealth from them.

The organisation believes that property rights pave the way for wealth creation and their main goal is to allow users to access articles, movies, games or any other digital assets without going through intermediaries, making digital assets scarce, tradable and identifiable [17].

Elastos is an open source software, whose development is backed by relevant industry leaders such as Tsinghua Science Park, the TD-SCDMA Industrial Alliance and the Foxconn Group. It wants to create a platform in which applications and services are not allowed to access the internet directly, in order to avoid malware to steal user data, as well as other types of attacks to online services. This vision culminated in a lightweight operating system for virtual machines, where decentralised applications communicate through a peer-to-peer network and are accessible via the users' mobile phones or personal computers, without making them change operating systems, i.e. applications run on Elastos' own operating system.

The Elastos Smart Web is mainly composed of four components:

- Elastos Blockchain: Elastos wants to build a decentralised Web where each digital assets has its own trustworthy ID, i.e. this is enabled by default because of the main properties that Blockchains provide;

- Elastos Runtime: lightweight operating system that prevents applications from connecting directly to the internet, available for on different processor architectures which enables applications to run either on the users' personal computer or mobile devices;
- Elastos Carrier: completely decentralised peer-to-peer-platform, which takes over the network traffic between the different virtual machines and exchanges information on the behalf of the applications;
- Elastos Software Development Kit: applications use the Elastos SDK in order to access their IDs and Elastos Carrier.

2.3.1 Storage Architecture

Elastos Hive is a service infrastructure that provides decentralised storage capabilities to Elastos Decentralised Applications. This technology takes advantage of IPFS and IPFS Cluster as the base infrastructure to save data, with the added benefit of compatibility with the rest of the services that are part of the ecosystem [2].

2.3.2 IPFS

IPFS is a peer-to-peer hypermedia protocol designed to make the web faster, safer and open, aiming to surpass HTTP in order to build a better web for the world.

Developed to be a peer-to-peer distributed filesystem that seeks to connect all computing devices in the same system of files, it provides high throughput content-addressable hyperlinks.

It combines a distributed hashtable, an incentivised block exchange, and a self-certifying namespace. Moreover, it has no single points of failure and the nodes taking part of the network do not need to trust each other[18].

This technology, when compared to the traditional web protocol, is more efficient distributing high volumes of data without duplication and stays true with the original vision of an open, flat web[3]. Additionally, IPFS is able to keep every version of the files and makes it simple to set up resilient networks for mirroring data.

2.3.3 Decentralised Authentication

Applications built within the Elastos Ecosystem use Decentralised IDs (DIDs), enable users to be secured with the power of blockchain, but with the easy and convenience like the existent protocols (e.g., Google, Facebook)[19].

At the moment of writing, users have to manage multiple IDs for the different applications they use, even with some developments that have been approaching this problem (e.g. Google OAuth - users have multiple IDs issued by multiple platforms). This is, by itself, a security problem, as the users have to trust these central authorities, responsible for issuing IDs and storing their information, to secure their infrastructure. On Elastos, DID Sidechains are responsible for issuing DIDs, which act as a proof of ownership of digital identity. While in traditional systems, a centralised entity is needed in order to avoid conflicts when issuing IDs, blockchain-based systems use wallet addresses as user's ID in order to carry the different

[2] https://elastos.academy/hive/
[3] https://ipfs.io/

Table 2.1: Comparison of storage characteristics between solutions

Characteristic	Solid	Blockstack	Elastos
Storage Decentralisation	X	X	X
Data Reutilisation	X		X

transactions. In these decentralised systems, public-key cryptography is used i.e. each public key is linked to a private key used to sign different assets) which eliminates the need for third party to confirm identities [4].

2.4 Solutions Comparison

Solid aims to empower users and organisations with privacy and control over their data, separating it from the applications they use[5]. Other projects also intend to offer this level of privacy to the users.

During this section, we will understand how comparable at this degree are the solutions offered by Solid, Blockstack and Elastos, mainly focusing on two key aspects: Storage and Authorisation/Authentication.

2.4.1 Storage

Solid relies on its pods in order to achieve storage decentralisation, exposing an HTTP interface for applications to communicate and operate with. Blockstack also offers similar functionalities through its Gaia hubs, even though not offering, as of the moment of writing, the possibility for applications to reuse the user data as Solid does. Elastos, unlike the other two solutions, uses IPFS for storage purposes, through Elastos Hive - making it also possible to reuse data.

In the Table 2.1 we expose the characteristics in comparison related to what the storage of the three solutions provides.

2.4.2 Authentication

Solid takes advantage of WebID-TLS as its main authentication mechanism, using cryptography certificates in order to verify that the users are, in fact, who they say they are. These certificates are owned by the user and no centralised authority is necessary for the authentication to proceed.

Blockstack also achieves authentication decentralisation by utilising public-key cryptography. Since there is no need for a password, the user owning the private key can directly authenticate and start using the applications.

Similarly to Solid and Blockstack, Elastos also enables decentralised authentication, this time taking advantage of Blockchain technology which issues decentralised IDs to the users which in turn also capitalises on a public-private key authentication-based method.

[4] https://elastos.academy/decentralized-identifier-sidechain-2/DIDSpotlightStart
[5] https://solid.inrupt.com/

Table 2.2: Comparison of authentication characteristics between solutions

Characteristic	Solid	Blockstack	Elastos
Authentication Decentralisation	X	X	X
Multiple Authentication Mechanisms	X		

As mentioned in the Table 2.2, all the compared solutions offer a decentralised authentication mechanism, with Solid currently implementing multiple options, enriching its ecosystem by doing so.

2.5 Testing

Testing is a fundamental process in Software Development. A variety of tests such as Unit and End-to-End testing ensure that the software was implemented as it was specified/designed. It also increases the confidence level during the deploy process and facilitates the maintenance of the system, by ensuring that previously implemented behaviours were not altered.

2.5.1 Unit Testing

This type of testing focuses on testing small pieces of code that can be logically isolated in the system[6]. This helps to ensure that whatever changes are introduced in the system, the blocks of code covered by the unit tests can maintain the expected behaviour if the tests are still passing.

Jest

Jest is a JavaScript testing framework that works with projects built with a variety of technologies, such as Babel, TypeScript, Node, React and Angular[7].

This framework allows the parallelisation of the tests by running them in their own process in order to maximise performance. Additionally, allows the mocking of objects outside of the test scope, tracks the coverage of the code and presents a report with the failed tests[8].

JUnit

This open-source framework is used to write unit tests on Java applications. By offering features such as assertion instructions, basic built-in template and test runners, it helps developers to write independent, testable modules[9].

Additionally, it provides an error report that allows developers to know exactly which test and which assertions failed in each of the developed test suites[10].

[6] https://smartbear.com/learn/automated-testing/what-is-unit-testing/
[7] https://jestjs.io/
[8] https://jestjs.io/
[9] https://www.softwaretestinghelp.com/junit-tutorial/
[10] https://www.softwaretestinghelp.com/junit-tutorial/

2.5.2 End-to-End Testing

This type of testing helps to validate multiple scenarios across the layers of the built software, being able to replicate entire scenarios as if it was the user executing real tasks on the system by interacting with the user interfaces. It is also useful to ensure accurate interaction and experience and to ensure that error situations are handled correctly [11].

Cypress

Cypress is an end to end open-source testing framework, that offers time travelling, debugging and real-time reloads whenever the tests are changed[12].

Unlike other frameworks, Cypress works with every frontend framework and every test is written with JavaScript. This framework sets itself apart from others for not using Selenium while being an all-in-one framework, assertion library, with mocking and stubbing[13].

Protractor

Protractor is an end-to-end framework that enables developers to mock user behaviour, capable of supporting Angular-specific locator strategies[14].

Based on Node.js, this framework utilises Jasmine for its testing interface and needs a Selenium Server in order to control the browser that is running the tests[15].

2.6 Summary

In this chapter we have studied the state of the art, including Solid and technologies with similar objectives and their own architectures, exposing different details for each of them. Finally, we have compared the solutions on two key characteristics: storage and authentication.

[11] https://www.testing-whiz.com/blog/5-reasons-to-perform-end-to-end-testing
[12] https://www.cypress.io/
[13] https://www.cypress.io/how-it-works
[14] https://www.protractortest.org/
[15] https://www.protractortest.org//tutorial

Chapter 3

Value Analysis

Since the main goal of the present work is to study how and if Solid can achieve mainstream development and adoption, experimentation and unpredictable results are expected.

This technology disrupts how traditionally we build social applications, so new processes, techniques, and good practices will, most probably, emerge in order to help developers and companies to with the development and distribution pipelines.

Moreover, it should consider that there is no mainstream development if there is no customer demand for it, so it is very important to emphasise why would users create demand for this shift to happen.

For this analysis, it was chosen The Fuzzy Front End Process (FFE), since it is ideal for projects focused on research. Unlike other processes like the New Product Development (NPD) where it is mainly applied to projects with a high degree of certainty and, possibly, predictable results, the FFE is more oriented to experimental and projects with high risk and where potential should be optimised[20].

3.1 Function Analysis System Technique

Keeping in mind that a proof of concept for Healthcare will be developed, and given that user adoption is ultimately the main drive for mainstream web development with Solid, we can find in this section an analysis using the Function Analysis System Technique (FAST) in two perspectives: business and users oriented. While on the former we will analyze why companies should adopt a technology such as Solid, on the latter it is more evident the benefits that the users can directly have with such experience.

3.1.1 Business Perspective

The ultimate goal of most companies is profitability, and frameworks such as Solid that have privacy at its core will not be adopted by these organizations if there is no monetary advantages of doing so. The Figure below applies the FAST technique from a Business perspective.

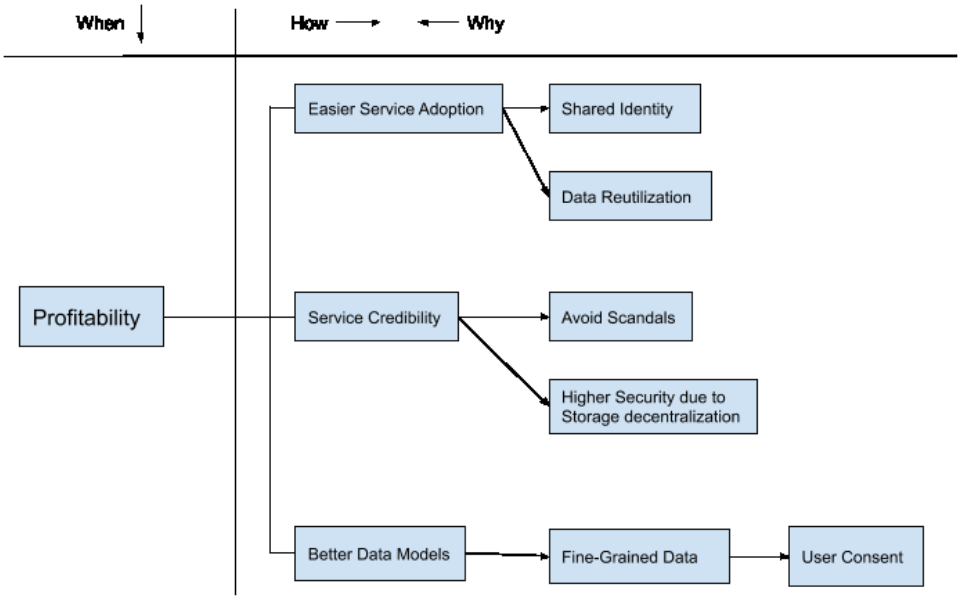

Figure 3.1: FAST applied in a Business Perspective

Organizations will most probably not want to build applications with Solid if the properties it provides do not bring them enough opportunities to achieve profitability.

At times, the process of registering into a service or product is a little bit cumbersome and that can be solved with a shared identity across services/products - which is achievable with decentralized authentication processes. Additionally, shared identities mean that services, if authorized to do so, might reuse existing data on the users profile which, by itself, is an opportunity to deliver a personalized service fast.

Nowadays we have seen some data scandals on the media such as the Cambridge Analytica scandal where millions of user had its data harvested by a third party without consent[2] and Facebook got not only its credibility affected, but also got legally fined [21]. Security breaches can also be avoided by adopting an architecture where the data storage is decentralized, giving more confidence to the users when using the service.

Finally, by asking for users' consent, companies can pay the users for a more fine-grained, controlled and accurate data for different purposes, such as data models that can be used to offer better experiences within a set of services or even research purposes.

3.1.2 User Perspective

In order to achieve what Solid offers at its core, there are a set of requirements that must be met. In the Figure 3.2, we can follow the FAST technique applied to the users perspective.

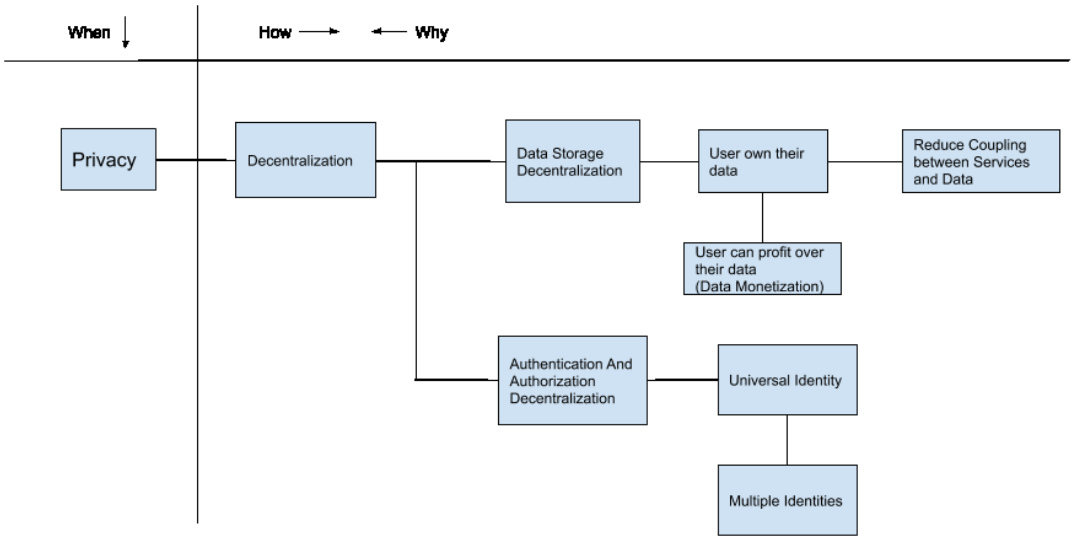

Figure 3.2: FAST applied in a User Perspective

With Privacy as the Higher Order Function, that is, what we ultimately want to achieve, it can only be attainable if we implement decentralisation at, at least, two levels: Data Storage and Authentication/Authorization.

When users find themselves in control of their own data, they can start profiting over it (Data Monetization) in order to help companies to give them better services, build data models or even donate their data for research purposes (e.g. Medical Research).

Moreover, it is essential to implement authentication and authorization decentralisation, so that users can opt in different services using the same identity (e.g., people would sign in on Twitter and Facebook with the same profile) or multiple (e.g., a user might want to remain anonymous in a service for any valid reason).

3.1.3 Value Proposition Canvas

Through the Value Proposition Canvas, it is possible to intersect what a product has to offer with what the customer desires (i.e. what we make and why would people acquire it). The value proposition canvas helps to quickly achieve a "minimum viable clarity" required to start building and testing a product or service[1].

The Value Proposition Canvas adds elements from behavioural psychology and design thinking[2]. It can be divided into two segments: Business and Customer. On the former, we identify Products and Services, Gain Creators and Pain Relievers while on the latter we clarify Gains, Pains and Customer Jobs.

[1]Value Proposition Canvas: https://www.peterjthomson.com/2013/11/value-proposition-canvas/
[2]Value Proposition Canvas: https://www.peterjthomson.com/2013/11/value-proposition-canvas/

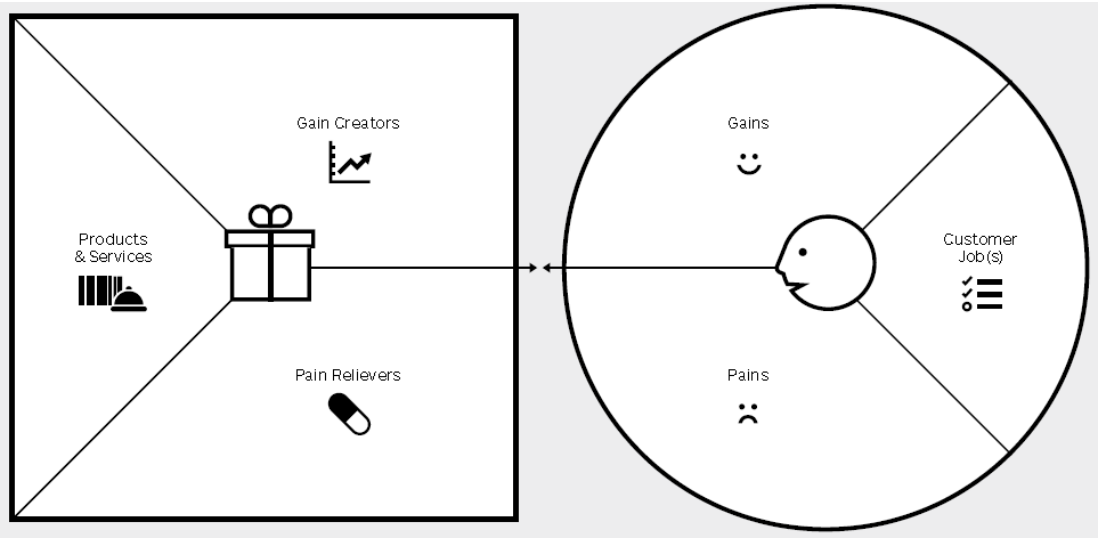

Figure 3.3: Value Proposition Canvas[3]

This analysis takes into consideration the value that services built with Solid at its core can provide for the customer as well as its benefits (Gains and Customer Jobs) and sacrifices (Pains).

Products and Services

In this section, it is listed all the products and services a value proposition is built around, specifying which of them help the customer in a functional, social or emotional way[22].

Since we want to build social web applications with privacy at its core, and given all the previously mentioned technologies, there is not a specific product or service to be listed, rather an architecture that can help customers to be in control of their online data and be exactly aware of whom and how is accessing it.

Gain Creators

This section describes the benefits that can be achieved from a customer perspective, as well as its desires, including social gains, cost savings and positive emotions. It may include not only outcomes the customer expect but also what he would be surprised by[22].

The following characteristics were considered:

- Storage Decentralization;
- Data Reutilization enables interoperability between services;
- Fine grained control over authorizations;
- Shared Identity across services.

Pain Relievers

This category exhibits how a product or service can alleviate, eliminate or reduce negative emotions, costs, risks or overall weaker user experiences[22].

The following characteristics were identified:

- Less cumbersome sign up process, taking advantage of shared identity;
- Permanently disconnect from service or revoke all permissions from it when a customer wants to do so;
- Evident control over resources and storage.

Gains

This area describes the benefits customers expect, desire or would appreciate to have - either functionalities, social gains, cost savings or positive[22].

The following characteristics were identified:

- Data Monetization - customers can get paid to give consent over their data for research purposes, among others;
- Shared identity - different services can be accessed with the same identity, which by itself reduces some of the cumbersome sign up processes;
- Fine-grained authorisation control over their data.

Pains

This category describes negative emotions, situations with considerable risk, undesired costs and negative experiences that the customer might be subjected to[22].

For this category, the following pains were identified:

- Storage Provider Reliability;
- Customers might loose their private keys, thus not being enabled to authenticate again with the same profile;
- Storage Provider maintenance cost - if the customer is using its own self-hosted storage provider then it might need to do its own maintenance, otherwise may need to pay to use an external storage provider if he can not find a reliable free one.

Customer Jobs

This final section describes which customer tasks that the products or service built with Solid might enable, problems they are trying to solve or the needs they are trying to satisfy[22].

The following customer jobs were identified:

- Control over data while sharing it with friends or any other entity (e.g., photos, files) - customers are able to control when and who exactly can see a photo or open a file;
- Provide Storage for other users - this represents a business opportunity for both organisations and regular users, who can provide reliable storage at a price;

- Identity credibility - verified identity across services (also easily proven through public-private key techniques). Nowadays some social services like Twitter manually verify identities of politicians and celebrities[4].

3.2 Customer Perceived Value

Customers buy products or services for the perceived benefit they will gain from it. Customer Perceived Value is created by adding this perceived benefit to the customer's opinion on the product or service[23].

The perceived value is often not related to the price itself, but with the relationship between perceived benefits and perceived costs. If the difference between these two dimensions is positive, it means that the customer's perceived value is high and the customer will most likely want to buy the product/service.

Customer Perceived Value may exist in three levels: Physical, Logical and Emotional[23]. Solid's benefits are more integrated within the Logical and Emotional levels.

Services empowered by Solid technology offer higher security than what users have today within the social web applications they use and may also financially compensate them as described in the Gains section of the Value Proposition Canvas.

Solid's advantages may also bring positive emotional value with its high privacy level. Moreover, by empowering users with universal identities, people may find friends and family more easily in different services, or rest assured that the person they are interacting with is who they say they are - attacking a currently growing problem: Identity Theft [5].

3.3 Summary

In this section, it was presented the analysis of the value of the present work. We have started by applying the FAST models in business and customer perspective, followed by the value proposition canvas that enabled us to study what organisations can offer and which problems they can solve with Solid and how can users capitalise on it. Finally, the chapter ends with an overview of the customer's perceived value.

[4] Twitter on verified accounts: https://help.twitter.com/en/managing-your-account/about-twitter-verified-accounts

[5] https://www.thebalance.com/college-identity-theft-a-growing-problem-1947515

Chapter 4

Business Analysis

Business Analysis is the process responsible for defining and maintaining requirements. In this chapter the functional and non-functional requirements are analysed and documented.

For the prototype that allowed the study of the Solid technology, it was adopted a medical use case that involves security as well as privacy between Patients and their Doctors.

4.1 Functional Requirements

This section describes the features that the systems composing our proof of concept will provide. These systems support the same set of functional requirements, allowing a direct comparison in terms of the software development processes:

FR1 As a Patient, I want to be able to create an account so that I can be able to access the system

FR2 As a Patient, I want to be able to log in on the system, so that I can use the implemented features

FR3 As a Patient, I want to be able to consult my saved medical notes;

FR4 As a Patient, I want to submit notes, so that I can keep track of important medical information

FR5 As a Patient, I want to share my medical exams with my doctor, so that he can assess my health concerns

These features were chosen specifically so that the systems can support a variety of concerns of current web social applications relevant for this study, such as the authentication process and the submission and control of information.

4.2 Non-functional Requirements

This section describes the other attributes other than features that the system should support.

The following Non-functional requirements were chosen:

NFR1 The Look and feel of both systems should be similar, so that the user experience analysis is not impacted as much as possible by the user interface

NFR2 High degree of testability: both systems should be capable of being tested at a unit and end-to-end levels

NFR3 Data should be kept separated from the application

4.3 Domain Model

Considering the domain of the intended functional requirements, we can see an abstraction of the business logic behind the software through domain models.

The domain of our proof of concept (cf. 4.1) consists of a patient storing medical notes and medical exams in the system. The Patient will, therefore, give authorisation to their doctor so that they can visualise the submitted exams.

The medical notes as well as the medical exams submitted by the Patient will be stored in different storage systems depending on the adopted approach.

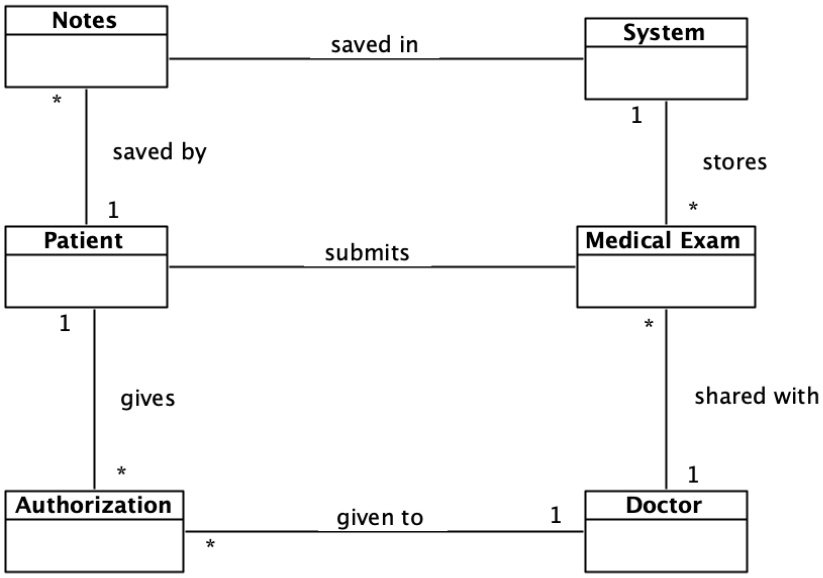

Figure 4.1: Domain Model

4.4 Use Cases

In this section we deep dive on each of the proposed functional requirements (cf. section 4.1), getting an abstraction on the overall design of both of the systems. The figure 4.2 represents the uses cases through an Use Case Diagram (cf. image 4.2.

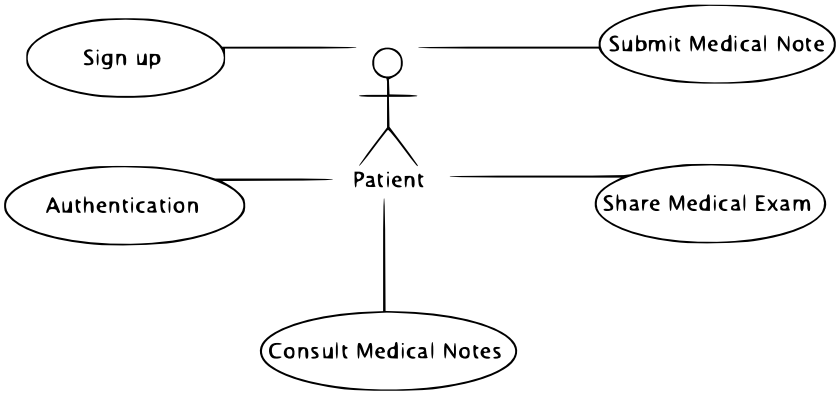

Figure 4.2: Use Case Diagram

4.5 Summary

In this chapter, it was defined, analysed and documented the functional and non-functional requirements for the present work as well as the use cases originated from them.

Chapter 5

Design

Given that it is intended to study if Solid is prepared for mainstream adoption not only from a standpoint of web development but also in terms of user experience, it was designed a proof of concept composed by two systems in order to be possible to better study and deep dive on the subject.

These two systems were designed to support the same functionalities (cf. section 4.1). Even though their frontend implementation will be similar, their backend infrastructure will significantly differ: one of them was designed to use Solid at its core, while the other will be provided with a centralised backend component built with a more traditional web technology stack. In this document, the former is often described in this study as the Decentralised Approach, while the latter is referred to as the Centralised Approach.

Although both of the implemented approaches offer a certain degree of decentralisation, in this study the focus is on how decentralised are the processes that manage data, such as storage, authorisation and data ownership.

These systems will allow us to compare the software development as a whole (e.g., average development time, documentation, testability) as well as the user experience enabled by them.

In this chapter we deep dive into the design of the systems composing our proof of concept, starting by defining the architecture of both approaches and comparing them, as well as design the use cases that will be implemented.

In each of the use cases, it is first analysed each of the approaches and then they are compared against each other. Finally, it is presented how Solid applications are interoperable and can often complement each other.

5.1 Architecture

This section presents the Architecture of the two implemented approaches and compares them.

5.1.1 Centralised Approach

This system will be provided by a traditional Client-Server Architecture as illustrated in figure 5.1.

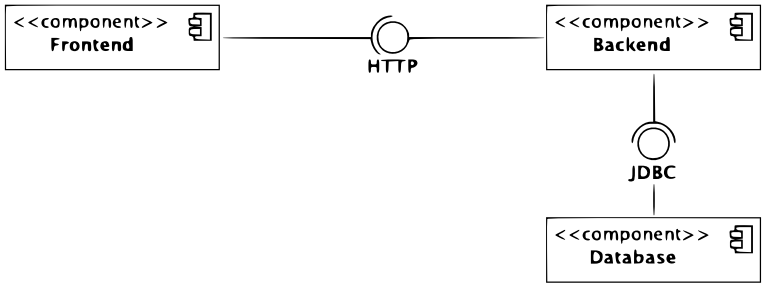

Figure 5.1: Centralised Approach Architecture

5.1.2 Decentralised Approach

This approach will be provided by a Client-Server Architecture, even though there can be as many Servers as the users want to, taking in consideration that they are the ones providing it (cf. figure 5.2).

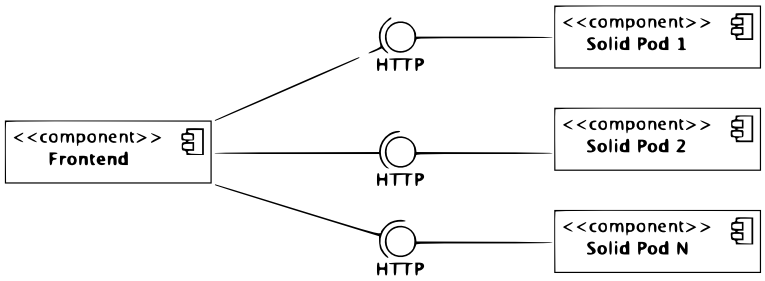

Figure 5.2: Decentralised Approach Architecture

5.1.3 Architecture Comparison

Although both approaches have apparently a similar Architecture, there are some key details that makes the decentralised approach stand out.

From a frontend perspective, both systems will be similar (cf. requirement NFR1 on section 4.2), communicating with their respective backend through an endpoint it exposes.

Its mostly on the backend that the Architecture differs significantly. While in the centralised approach we have a backend in a certain location, awaiting requests from the frontend, in the decentralised approach the location of the backend component (i.e., Solid Pod) is provided in the moment of login.

Each user can connect the frontend application to their own storage system (cf. figure 5.2). Ultimately, this shifts the ownership of the data from the companies to the users, enabling them to decide when, for how long and which data applications can access and control.

5.2 Sign Up

Creating user accounts is one of the most basic features of social web applications, but given the decentralised context of the project is crucial to understand how that can be done in a system that uses Solid Pods as its base layer. In this section we analyse the implementation of this feature in both systems, starting with the centralised implementation and moving towards the decentralised one, highlighting the main differences with a direct comparison.

5.2.1 Centralised Approach

In centralised systems, user registration is very similar across different web social applications. As illustrated in figure 5.3, user access the registration page on the frontend that then performs a request to the API exposed by the backend in order to create the user account.

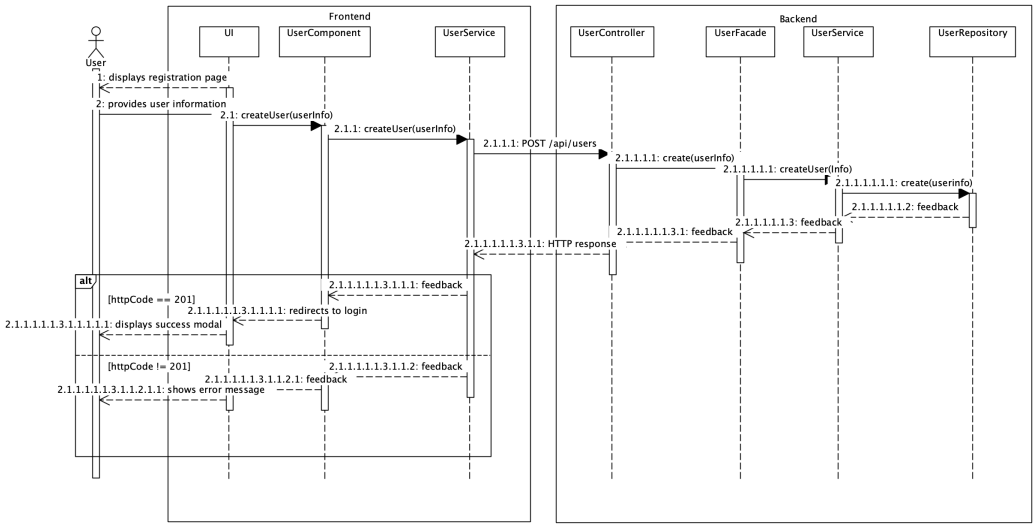

Figure 5.3: User Registration - Centralised Approach

On the backend each process might differ slightly even within centralised solutions depending, for instance, if the backend is a monolithic system or composed by multiple microservices. In this case, the designed approach is only composed by one server and the data flows through the different layers and the account is finally created in a SQL database.

5.2.2 Decentralised Approach

The process to create an account on the system that utilises Solid at its core, as illustrated in figure 5.4, starts by having the Patient indicate its storage provider (i.e., Solid Pod). This makes it so that the data is not storage in the application, respecting the non functional requirement NFR3. Afterwards, the process is completely controlled by the provided instance, which displays a form in its own UI and creates the user account if all the requirements are met.

One notable characteristic is the fact that the user, after completing the registration process, is not redirected back to the application that triggered the process, but rather to its WebID profile page, generated by the chosen Solid Pod instance itself.

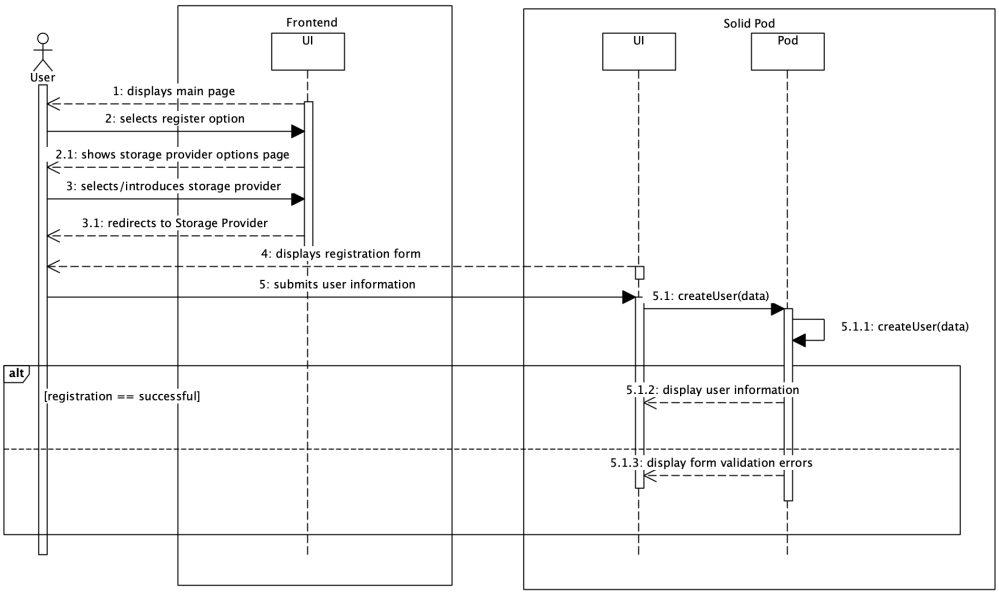

Figure 5.4: User Registration - Decentralised Approach

5.2.3 Approaches Comparison

These approaches differ significantly both at a frontend and backend levels. In the centralised solution, the user is directly requested by the application to fill a form which is then validated by the backend and the frontend knows when the process ends.

Conversely, in the decentralised approach, the frontend delegates the entire process to another component chosen by the user and has no information if the process was completed or not. This application will also not keep a list of all its users, nor will be responsible for storing credentials, thus avoiding data leaks, which can happen in the centralised approach.

5.3 Authentication

One important concern when introducing decentralisation into systems is how the authentication process will be implemented and how much complexity may be introduced as a consequence of this specific characteristic. In this section we deep dive in the design of the Authentication process in the centralised and decentralised approaches.

5.3.1 Centralised Approach

The centralised approach follows a common user/password sign-in process. The frontend is responsible for displaying the authentication form and the submitted information is validated by the backend (cf. figure 5.5).

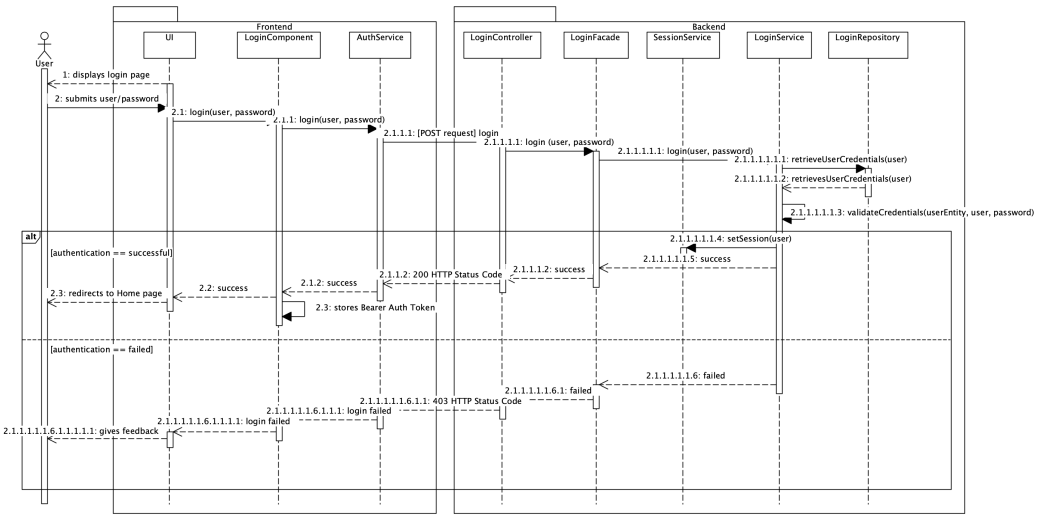

Figure 5.5: User Authentication - Centralised Approach

5.3.2 Decentralised Approach

Although Solid allows different authentication processes (cf. section 2.1.3, for this proof of concept and in accordance with the section 4.2, the one chosen was the WebID-OIDC mechanism (cf. section 2.1.3). This allows a familiar experience to the user as this is a common authentication method for social web applications at the moment of writing.

In this scenario, the user must first provide the Solid Pod location, so that the authentication itself can be performed on it. After that, they should fill the form and authenticate successfully if the information is correct (cf. figure 5.6).

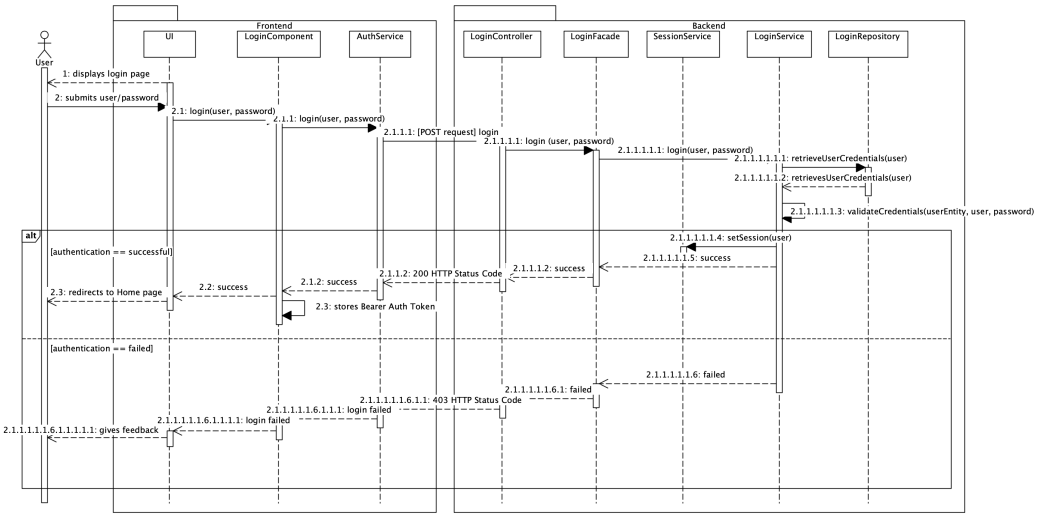

Figure 5.6: User Authentication - Decentralised Approach

5.3.3 Approaches Comparison

In this use case, both approaches have a very similar approach in terms of the steps they need to perform in order to authenticate in the applications. The main difference is the fact

that in the decentralised approach, the user provides the location where its authentication information is stored, thus having total control over it.

5.4 Consult Medical Notes

This section explores the visualisation of the user notes and the process in order to retrieve it from the backend. Firstly, it is presented the centralised approach, followed by the decentralised approach and finishing with the main differences between both designed systems.

5.4.1 Centralised Approach

In order to retrieve the user medical notes, the frontend of this approach retrieves the notes directly through an API, parsing each of them and finally displaying them on the UI (cf. figure 5.7).

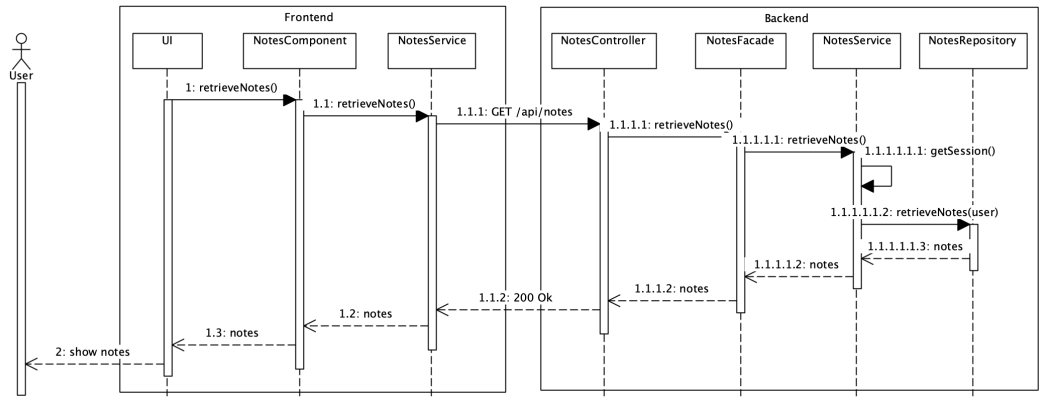

Figure 5.7: Consult User's Medical Notes - Centralised Approach

5.4.2 Decentralised Approach

The decentralised approach retrieves the user's medical notes in two steps (cf. figure 5.8): firstly, it starts by retrieving the user's Public Type Index file, which contains the different public files present on the user's pod and, finally, it queries the file to retrieve the location of the medical notes and performs request to the Solid Pod in order to fetch them.

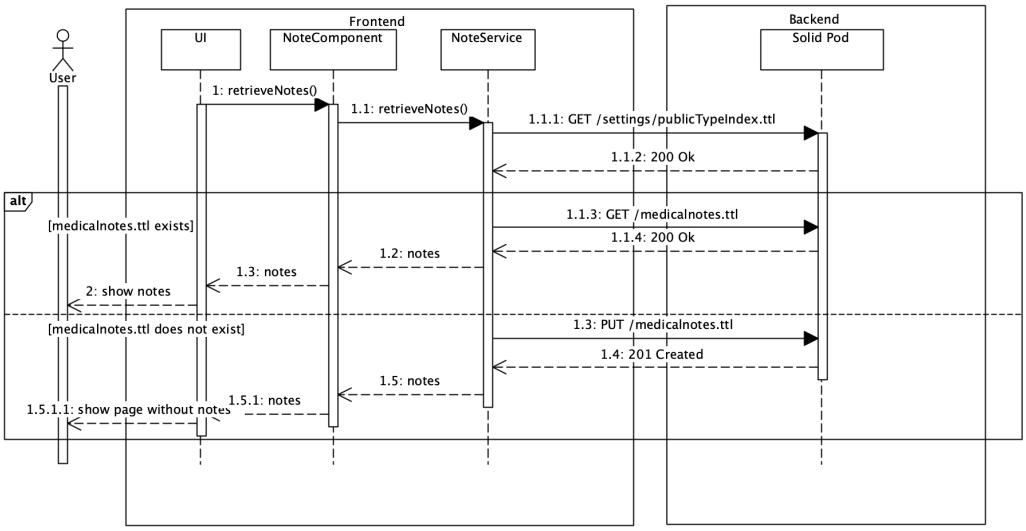

Figure 5.8: Consult User's Medical Notes - Decentralised Approach

5.4.3 Approaches Comparison

From a frontend perspective, both approaches follow a similar structure in what concerns components and services. The main differences rely on the APIs needed to retrieve the information regarding the user's medical notes.

In the centralised approach, the frontend requests the medical notes in one request through one endpoint exposed by its backend component. Conversely, the decentralised approach needs three endpoints to do the same, as described in section 5.4.2, retrieving the medical notes location through the Public Type Index file and creating a new one in case it does not exist already.

5.5 Submit Medical Note

In the designed systems, Patients should have the possibility to save medical notes. This section describes two approaches from a frontend and backend perspective and highlights the main differences between them.

5.5.1 Centralised Approach

In this approach, the frontend display a form to the user, who fills and submits the note. This component makes use of an endpoint exposed by the backend component, which allows the creation of this note. The sequence diagram illustrated in figure 5.9 describes the flow with more detail. The backend receives the note through an API and stores it in the database after validating the necessary credentials.

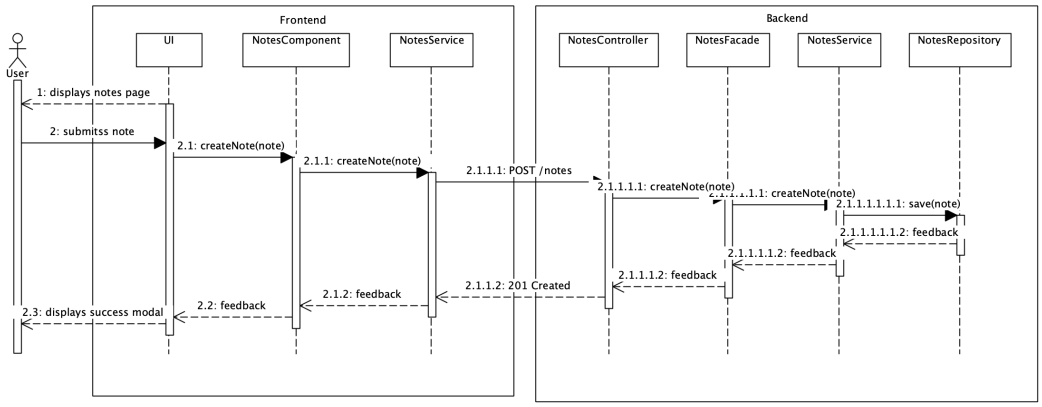

Figure 5.9: Submit Medical Note - Centralised Approach

5.5.2 Decentralised Approach

The user can save medical notes in the decentralised approach by filling a form and the Solid Pod allows the creation of the note through a request to the respective endpoint. The complete process is illustrated in the figure 5.10.

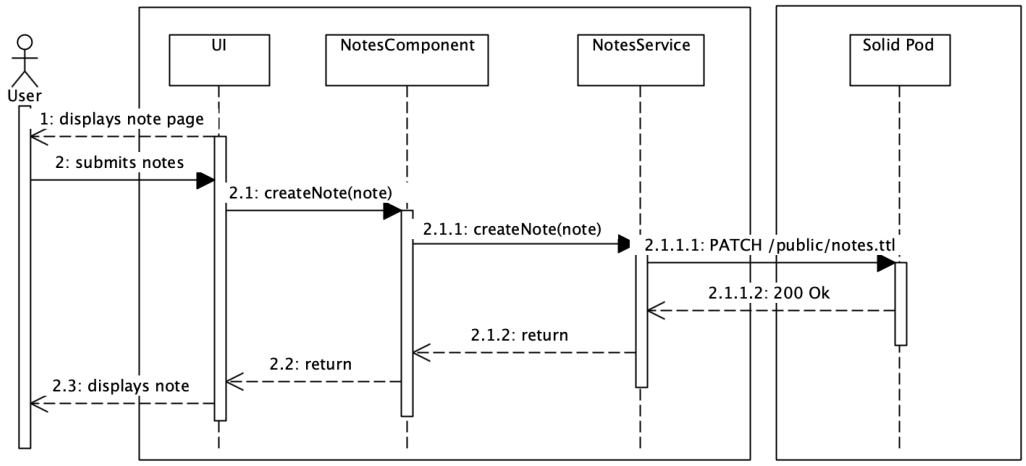

Figure 5.10: Submit Medical Note - Decentralised Approach

5.5.3 Approaches Comparison

From a frontend perspective, both approaches present a very similar design: the user fills a form, which then has its data mapped into a component and, finally, a service handles the data and performs request to the backend.

On the backend, both approaches save the request with the decentralised one acting as a black box, and the centralised one handling the data through its different layers and ultimately saving the note in the database.

5.6 Share Medical Exam

This section presents the design of the feature in which the Patient can share a medical exam with their Doctor. Firstly it is presented the design for the centralised approach, followed by the decentralised one and finishes with a comparison between both of them.

5.6.1 Centralised Approach

In the centralised approach, the user has to select the medical exam they want to share from their file system, and select one of the Doctors available in the system for the exam to be shared with. The backend will then be responsible for setting up the authorisation and store the exam.

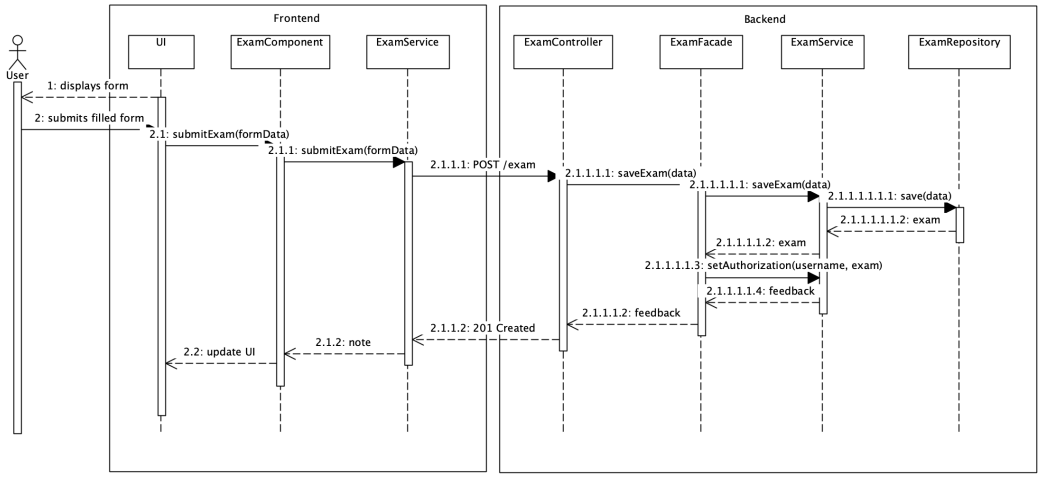

Figure 5.11: Share Medical Exam - Centralised Approach Approach

5.6.2 Decentralised Approach

In order to share a medical exam in the decentralised approach, the Patient submits it through an input and chooses the Doctor with who they want to share the exam. This process has to be done in two steps because of the way the Solid Pod API works: first the frontend performs a request in order to save the exam on the user's Solid Pod and then, if successful, performs another request so that it can authorise the Doctor to see the exam (cf. figure 5.12).

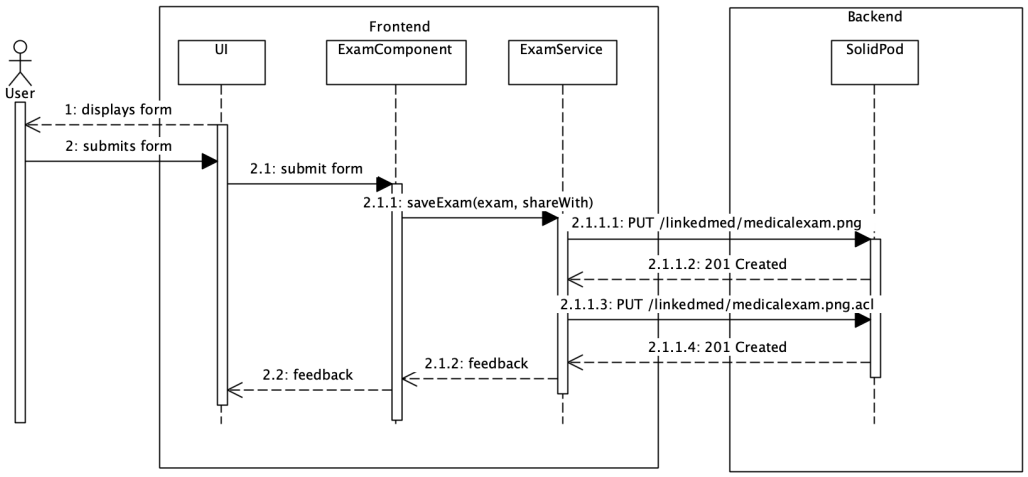

Figure 5.12: Share Medical Exam - Decentralised Approach Approach

5.6.3 Approaches Comparison

In this use case, the flows differ significantly especially on the authorisation flow. While the centralised approach can delegate work to the backend by performing a single request where it sends both the image and the identification of the Doctor with who the exam should be shared, the decentralised one has two perform two separate requests.

5.7 Solid Applications Interoperability

In a context where the Patient can save and share their medical exams, it is desirable to have a requirement in order to allow the Doctors to consult the exams shared by the Patient. Even though not chosen as a functional requirement for this project, as described in section 4.1, applications built with the Solid technology at its core are interoperable by default.

This interoperability allows applications to extend the functionalities of each other. In this case, it is possible to connect another application (in this case, a file explorer) from the existing Solid ecosystem to allow the Doctor to check files in the Patient's Pod as long as they have the authorisation to do so, as illustrated in figure 5.13).

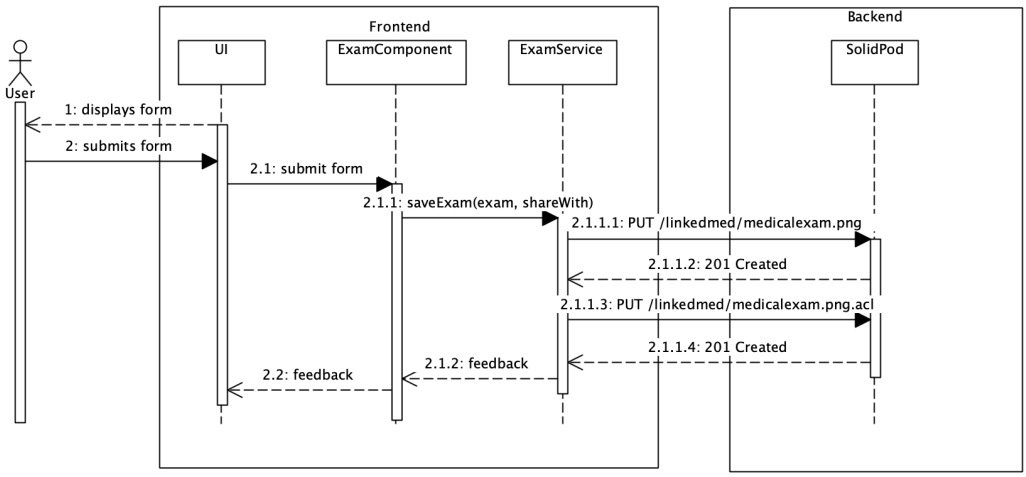

Figure 5.13: Consulting Patient's Medical Exam through another Solid application

5.8 Summary

This chapter presents the design of the systems for this project, starting with the architecture designed for both approaches, followed by each of the use cases' design and finishing with information on how interoperability between Solid applications can be introduced on a solution.

Chapter 6

Implementation

This chapter documents the implementation of the present dissertation. It starts by presenting the technology stack for both implemented approaches, followed by explaining how the approaches communicate within their architecture. Finally, it presents the implementation of the uses cases as well as how the testing was performed.

6.1 Technology Stack

In this section it is presented the technologies used in order to develop the applications used in both the centralised and decentralised approaches. The technologies, libraries and programming languages should be current and facilitate the development of the applications.

6.1.1 Centralised Approach

In the centralised approach (cf. section 5.1), the application was generated with JHipster. This development platform allowed the creation of the frontend and the backend as well as some base infrastructure, such as Docker files, and boilerplate code, including the base code for the user creation and login both ends. The list of different technologies used as well as a description of each of them is provided on table 6.1.

Table 6.1: Technologies used in the centralised approach development

Technology	**Description**
Docker	Tool that allows the Containerization of software
Docker Compose	Tool that allows the creation and managing of multi-containers software
PostgreSQL	SQL Database
Java	Object-oriented programming language
Spring Boot	Framework that facilitates the creation applications, providing boilerplate code for Web Services
JHipster	Development Platform that accelerates the development process by generating applications with boilerplate code and infrastructure
JavaScript	Interpreted Programming Language mainly used to build Frontend and Backend applications
React	JavaScript library used for building component-based user interfaces

The frontend was developed using React, a JavaScript library, allowing it to be composed by different components that interact with each other. These components kept the code organised and allowed separation of concerns and responsibilities.

The backend component, in its turn, was developed using Spring Boot with Java as the chosen programming language. Similarly to what React provided in the frontend, this framework

allowed the backend to be easily developed using different components/services known as Beans, making it easier to design boundaries within the system and separate concerns as well as enabling an easy integration with the PostgreSQL database.

6.1.2 Decentralised Approach

The decentralised approach (cf. section 5.2) has the Solid Pod as its backend component and storage system, therefore not requiring development on it apart from the infrastructure setup.

The frontend component was developed using JavaScript, embedded in the React framework just as in the centralised approach. The backend component is the Solid Pod that the user has the possibility to connect, which exposes a HTTP API that can be used to manage the Pod and its information. The development was aided by tools like Docker and Docker Compose which facilitated the construction of the different components and elements of the system. In the table 6.2 there is a description of the main tools used.

Table 6.2: Technologies used in the decentralised approach development

Technology	**Description**
Docker	Tool that allows the Containerization of software
Docker Compose	Tool that allows the creation and managing of multi-containers software
Solid Pod	Technology that serves as a storage system to organise data, applications and as well as manage identification
JavaScript	Interpreted Programming Language mainly used to build Frontend and Backend applications
React	JavaScript library used for building component-based user interfaces

6.2 Communication Standards

During the development of both the centralised as well as the decentralised approach, it was adopted some communication standards for both of them.

The centralised approach follows a traditional HTTP API, that mostly consumes and produces JSON, with an exception to the submission of the medical exam, which is accomplished through a multipart request (cf. section 6.7).

In order to explore what the Solid Pods are capable of and how applications can communicate with, the decentralised approach communicates through the Turtle Syntax as well as SPARQL Queries, with the same exception to the centralised approach (cf. section 6.8). This allowed to experiment with different methodologies when interaction with the Solid Pod.

6.3 Use Cases

This section explains the implementation of each of the functional requirements (cf. section 4.1), exposing the main flows and highlighting key features for each of them.

6.3.1 Sign Up

The sign up process is the first feature that Patients encounter when using both systems. This process is as straightforward as possible according to current standards.

For the centralised approach, each Patient will encounter a similar form as many other current social applications such as Facebook or LinkedIn have available. From an architecture point of view, the frontend component carries the information to the backend through an API and provides the feedback to the user accordingly (cf. section 5.3.

The decentralised approach, however, starts the flow by asking the Patient about the location of its storage system. This is where the decentralisation aspect is introduced to the system, allowing the application to perform every application on the user's Solid Pod, which can be either one set up by the user or one opened by a community that is allowing new users. After this step, the application completely delegates the sign up process to the provided component.

As described in 5.4, after the user successfully submits the form, the Solid Pod redirects them to a page where they can manage its WebId profile. One notable characteristic of this flow is that the user is not redirected to the application that originated the sign up process, which can impact the user experience. Upon completion of the registration process, the Solid Pod returns the information of the WebId profile described in the Turtle syntax. An excerpt of the provided profile data can be consulted in the listing 6.1.

```
@prefix : </#>.
@prefix ped: <>.
@prefix ldp: http://www.w3.org/ns/ldp#.
@prefix terms: http://purl.org/dc/terms/.
@prefix XML: http://www.w3.org/2001/XMLSchema#.
@prefix n0: </.well-known/>.
@prefix inbox: </inbox/>.
@prefix priv: </private/>.
@prefix pro: </profile/>.
@prefix pub: </public/>.
@prefix set: </settings/>.
@prefix st: http://www.w3.org/ns/posix/stat#.
@prefix vnd: http://www.w3.org/ns/iana/media-types/image/vnd.microsoft.
    icon#.
@prefix c: </profile/card#>.
@prefix ter: http://www.w3.org/ns/solid/terms#.
@prefix pl: http://www.w3.org/ns/iana/media-types/text/plain#.

pub:
a ldp:BasicContainer, ldp:Container, ldp:Resource;
terms:modified "2020-08-23T22:10:07Z"^^XML:dateTime; st:mtime
    1598220607.5;
st:size 4096.
</robots.txt> a pl:Resource, ldp:Resource;
terms:modified "2020-08-23T22:10:07Z"^^XML:dateTime; st:mtime
    1598220607.5;
st:size 83.

set:
a ldp:BasicContainer, ldp:Container, ldp:Resource;
terms:modified "2020-08-23T22:10:07Z"^^XML:dateTime; st:mtime
    1598220607.5;
```

```
29 st:size 4096.
```

Listing 6.1: Excerpt of the returned WebId data representation

The returned WebId information describes the different elements present on the user's Pod such as the public folder and the files in it, as well as useful metadata such as the description of what type each element has and the last modification date. The line 22, for instance, specifies a file named robots, which was modified on the 23rd of August of 2020 and has a size of 83 bytes. This can be specially useful for applications that may lookup for files of a certain type in order to retrieve information useful for the user experience of the application.

6.3.2 Authentication

An important concern when introducing decentralisation into systems should be how the authentication process will be implemented and how much complexity may be introduced as a consequence of this specific characteristic.

The centralised approach follows a traditional sign in process (cf. section 5.5), with a form through which the user provides its username and password. The backend exposes a HTTP API and validates the information comparing it with what is stored on the database, allowing the frontend to redirect the user to the application main page in case of success and saving the returned Bearer token in the local storage so that it can be used to prove the user is who they say they are in future HTTP requests. An error message is shown in case the authentication process fails.

The decentralised approach does not have a backend component until the Patient provides it as the first step of the authentication flow (cf. section 5.6), introducing the decentralisation in the sense that the provided component (i.e., Solid Pod) is the one proving that the users are who they are saying they are and stores their credentials. This specific characteristic also provides an additional layer of security to the overall ecosystem, since there is no centralised component holding all the credentials for the users that use this application.

Although Solid allows different authentication processes, the one chosen for the proof of concept was the WebID-OIDC mechanism (which can utilise the traditional username/password), allowing the user to have not only a similar experience in both the implemented approaches, but also a similar experience as they would have in a popular social application at the time of writing. From a technical perspective, similarly to the centralised approach, the Solid Pod returns a Bearer token which is stored to be used in future requests to this storage system. Along with the token, the storage system provides a description of the user's WebId to the application, as described in 6.1.

6.3.3 Consult Medical Notes

In both systems, the Patient has the possibility to consult medical notes submitted by them into the system.

The frontend of the centralised approach takes advantage of an API exposed by the backend (cf. section 6.3) which enables the retrieval of this notes returned through the traditional JSON syntax. For each one of the created notes it is created a React component that is rendered onto the display. The process is straightforward and can be done in just one flow.

Table 6.3: Consult Medical Notes API - Centralised Approach

Resource	**HTTP Verb**	**Media Type**
/api/notes	GET	application/json

Conversely, in the decentralised approach, the process is more complex. When the Patients are using the application for the first time, they do not have any medical notes on their storage system. Therefore, the first step is to setup the Solid Pod to have a place so that it can save the submitted notes. Every time the notes page initialise, it is first verified if the file exists in the backend component and it is created if not.

In order to check if the file already exists, the Solid Pod provides a HTTP API (cf. section 6.4) that allows to check if the type of file the application is looking for is already on the system. This can be done by retrieving the Public Type Index file stored in the settings folder, which is a Turtle Language Document. This file is returned as can be seen in the code listing 6.2.

```
@prefix : <#>.
@prefix solid: <http://www.w3.org/ns/solid/terms#>.
@prefix schem: <http://schema.org/>.

<> a solid:ListedDocument, solid:TypeIndex.

:1590350348549705354539393265
    a solid:TypeRegistration;
    solid:forClass schem:TextDigitalDocument;
    solid:instance </public/medicalnotes.ttl>.
```

Listing 6.2: Example of a Public Type Index response

The index allows applications to look for documents inside the Pod as well as to be aware of the schema of these applications, so that they know how to manipulate it. This pattern is common when developing Solid applications: it starts by retrieving the public type index location from the user's WebId and then it is possible to look for the intended information by filtering for its schema type.

If the file where the application wants to read the medical notes from did not exist, it would have to be created. For this, the application can perform a PUT HTTP Request (cf. section 6.4) to the Solid Pod in order to create it.

Table 6.4: Consult Medical Notes API - Decentralised Approach

Resource	**HTTP Verb**	Media Type
/settings/publicTypeIndex.ttl	GET	text/turtle
/linkedmed/medicalnotes.ttl	GET	text/turtle
/linkedmed/medicalnotes.ttl	PUT	text/turtle

6.3.4 Submit Medical Note

In the implemented systems, Patients are able to save medical notes in the system, so that they can keep track of any symptoms or other relevant information on their health.

The centralised approach follows a traditional flow where it exposes a HTTP endpoint (cf. section 6.5) that enables the frontend to perform a request in behalf of the user, with the Bearer token retrieved in the authentication process. It's a straightforward process with just one end-to-end step since the submission of the note to the storage of it in the database.

Table 6.5: Submit Medical Note API - Centralised Approach

Resource	**HTTP Verb**	Media Type
/api/notes	POST	application/json

Similarly, in the decentralised approach, the process is also straightforward by having the frontend performing a PATCH HTTP Request (cf. 6.6) to the Patient's storage system.

Table 6.6: Submit Medical Note API - Decentralised Approach

Resource	**HTTP Verb**	Media Type
/linkedmed/medicalnotes.ttl	PATCH	application/sparql-update

The request is successfully achieved through a SPARQL Update Query to the Turtle Language Document that as retrieved and/or created when the frontend displayed the Medical Notes page (cf. section 6.3.3). The query responsible for the creation of the medical note in the storage system is described in the listing 6.3.

```
1 INSERT DATA {
2     https://pedropinto.solid.community/public/medicalnotes.ttl
    #159839827610655459341776195 49
3         a http://schema.org/TextDigitalDocument;
4         http://schema.org/text "Another headache. This time it lasted 3
    hours.";
5         http://schema.org/dateCreated "2020-08-25T23:31:16Z"^^http://www
    .w3.org/2001/XMLSchema#dateTime.
6 };
```

Listing 6.3: Saving a new medical note through a SPARQL Query

The query starts by explicitly stating that will perform an insert, as well as pointing the file in which it wants to insert data and providing an ID (in this case, the note ID will be 159839827610655459341776195 49). Along with it, it is also provided the creation date of the note as well as its schema which allows application to understand how to manipulate this note.

6.3.5 Share Medical Exam

The last implemented feature on both approaches is the possibility for a Patient to share their medical exams with a Doctor. This feature is particularly interesting given that it explores the authorisation process with the Solid Pod.

In the centralised approach, the backend component does most of the work: it exposes a HTTP API (cf. section 6.7) in which it receives the exam and the identifier of the user that can access the exam.

Table 6.7: Share Medical Exam API - Centralised Approach

Resource	HTTP Verb	Media Type
/exams	POST	multipart/form-data

Conversely, in the decentralised approach, the process can not be as simple as its counterpart approach. With Solid, this process must be done in two steps: firstly the image is created in the Solid Pod and then an ACL file is created for that image, applying the authorisation. In the the table 6.8, it is described the endpoints exposed by the Solid Pod that allow both actions.

Table 6.8: Share Medical Exam API - Decentralised Approach

Resource	HTTP Verb	Media Type
/linkedmed/medicalexam.png	PUT	image/png
/linkedmed/medicalexam.png.acl	PUT	text/turtle

This ACL file includes the authorisation rules for the submitted medical exam, as can be seen in the code listing 6.4.

```
@prefix acl: <http://www.w3.org/ns/auth/acl#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix n: <http://www.w3.org/2006/vcard/ns#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix : <https://pedropinto.solid.community/linkedMed/medicalexam.png.
    acl#>.
@prefix me: <https://pedropinto.solid.community/profile/card#me>.

:ReadWriteControl a acl:Authorization;
acl:accessTo <https://pedropinto.solid.community/linkedMed/medicalexam.
    png>;
acl:default <https://pedropinto.solid.community/linkedMed/medicalexam.
    png>;
acl:agent <https://pedropinto.solid.community/profile/card#me>;
acl:mode acl:Read, acl:Write, acl:Control.

:ReadWrite a acl:Authorization;
acl:accessTo <https://pedropinto.solid.community/linkedMed/medicalexam.
    png>;
acl:default <https://pedropinto.solid.community/linkedMed/medicalexam.
    png>;
acl:agent <https://tmdeitester.solid.community/profile/card#me>;
acl:mode acl:Read, acl:Write.
```

Listing 6.4: Creating an ACL file for the medical exam

In this case, it can be seen that the Doctor with the username TMDEITester is getting both read and write access to the submitted medical exam.

6.4 Solid Applications Interoperability

In the developed applications, even though Patients can submit their medical exams and manage who can see it, it is not possible for the Doctor to authenticate and see the shared

checkups. Using Solid, interoperability can easily be achieved between applications and therefore it is possible for some applications to complement functionalities that others may lack.

As long a user has the necessary permissions to access a resource, that can be done independently of the frontend component being used. Using a File Manager built by a developer of the Solid community [1], the Doctor can login and explore the Patient's storage system, accessing its medical exams.

6.5 Testing

Even though the development systems are only a proof of concept, testing was also considered not only because it helped to ensure that the features were working as expected, but also because it is important to understand if an application built with Solid at its core is as easily tested as other current approaches and more centralised approaches.

This section presents the different developed tests as well as how they are conceived and which technologies or libraries or used for that sense.

6.5.1 Unit Testing

Unit tests are a fundamental part of the development process of any system, ensuring correctness and integrity as well as facilitating changes across the code.

6.5.2 Frontend

In the developed applications, both frontend components utilise React which takes advantage of JavaScript at its core. For that reason, there are no differences between testing the centralised and the decentralised approach. Both approaches communicate with their backend components using HTTP (cf. sections 5.1 and 5.2) and that can be mocked by Jest - the testing library used when testing both approaches.

6.5.3 Backend

In the decentralised approach, the Solid Pod is the only backend component and it acts as a black box, allowing applications to be built on top of it. This component is by the team that develops it and that also tracks issues on its Github project with the help of the community [2].

The centralised approach, developed with SpringBoot, was tested using JUnit 5. The ratio of tested code was not taken as the most important factor given that this is a proof of concept, even though in an ideal scenario the majority of the code should be covered by unit tests.

6.5.4 End-to-End Testing

End-to-end testing allows the developers to automated tests that mock the user behaviour, ensuring the software layers are working together as expected given a set of inputs.

[1] File Manager: https://otto-aa.github.io/solid-filemanager/
[2] Solid Pod Repository: https://github.com/solid/node-solid-server/issues/

During the development of this dissertations, two end-to-end frameworks were studied (cf. section 2.5.2): Cypress and Protractor. Because there was previous knowledge acquired on the development of tests using Cypress, this was the chosen framework.

While developing the tests, it was detected that Cypress does not support, at the moment of the development of the tests, testing across multiple browser windows/tabs. Since the authentication process opens a new window so that the user can introduce their storage system location (cf. section 5.3.2) and no workaround was found for this limitation, the end-to-end tests were developed using the framework Protractor

One of the developed tests can be seen through Appendix A.1.

6.6 Summary

This chapter presents the implementation of the centralised and decentralised approaches.

Firstly, it is introduced the technologies utilised for each of the approaches, followed by how the communication between the frontend and the backend works.

Afterwards, each of the use cases implementations is showcased, highlighting certain aspects between each of them, along with an additional feature introduced by the interoperability that Solid applications are able to have.

Finally, the chapter presents how the testing was performed.

Chapter 7

Evaluation

This chapter explores the experimentation done through the course of this dissertation as well as its assessment. Firstly it is presented the indicators and information sources, followed by the assessment methodology and, finally, the analysis of the results.

7.1 Indicators and Information Sources

Considering the proposed objectives (cf. section 1.2), it was defined a set of metrics that will support the conclusions of the present work. These metrics are as follows:

1. Availability of the development tools/libraries to develop Solid applications
2. The current state of the Solid technology documentation availability
3. Comparison of the approximate man-hours invested developing an application with Solid at its core compared with another with a more traditional web stack
4. How easy is the onboarding of the user compared to a traditional sign-up/sign-in process
5. How the user experience was impacted by adopting a more decentralised approach when compared to its counterpart

7.2 Assessment Methodology

In order to achieve results and draw conclusions for each of the metrics, it was thought a methodology to assess each of them, presented in this section.

The availability of tools and libraries as well as the documentation to develop Solid applications could be researched and studied during the process of developing the Solid application.

As for the onboarding process of the users and the impact that Solid can have in the user experience of applications, it was presented a survey to ten users who could evaluate it so that more precise insight could be collected.

The man-hours invested for each of the applications will also be considered, with the amount spent having been saved when both approaches were developed. In this case, variables such as the previous experience with the experimented technologies, among others, will have a considerable influence.

7.3 Results Analysis

This section deep dives in the presented indicators (cf. section 7.1) and presents the analysis for each of them.

7.3.1 Documentation and Libraries Availability

When developing the decentralised approach, it was researched possible libraries that would make it easier to communicate with the Solid Pod. There were some libraries that significantly helped to develop the decentralised system:

1. Tripledoc - Helps developers to communicate with the Solid Pods without having significant knowledge about RDF. [1]
2. Plandoc - Utilises Tripledoc and helps initialise documents in the Solid Pod if they are not present [2]
3. Solid Auth Client - Allows applications to log in to Solid Pods and manage data [3]
4. Solid ACL Utils - Allows the creation and manipulation of ACL files [4]

As for the documentation itself, since the beginning of this research, the Solid team has been improving and organising their documentation [5] substantially, facilitating the understanding of the project and providing more resources and tutorials to developers.

At the moment of writing, what is available is organised enough for developers to try the technology and learn the basics, even though it lacks deeper knowledge on subjects related with Linked Data such as vocabularies and how applications can take more advantage of them and create interoperability between multiple projects through the creation of new vocabularies.

7.3.2 User Experience

The user experience that can be achieved using Solid as its core technology its fundamental in order to achieve mass adoption from a end user perspective, since no one would want to use an application that advertises itself as more private and secure, but its slow and complicated to use.

In order to study how the user experience on the implemented functionalities are perceived by the user, a questionnaire was presented to ten users who had to experiment with all the features made available for both approaches. As a prerequisite for the selection of who would answer the survey, the user had to be active in any social application at least once a day.

Although the user interface for both approaches is not exactly the same, the simplicity of the flows from a frontend perspective makes it so that it does not have a significant impact on the results.

[1] https://vincenttunru.gitlab.io/tripledoc
[2] https://developer.aliyun.com/mirror/npm/package/plandoc
[3] https://www.npmjs.com/package/solid-auth-client
[4] https://github.com/Otto-AA/solid-acl-utils
[5] https://solidproject.org/

In this section, we deep dive in each of the questions the users had to respond to, with the decentralised approach being referenced as Service 1 and the centralised on as Service 2. Each of the participants performed all the actions in both services and responded to the survey right after.

All the questions were provided by a categorical Likert scale, with the following options:

1. Strongly Disagree
2. Disagree
3. Agree
4. Strongly Agree

The lack of neutral response was intentional because even though it is mostly introduced with the intention of reducing false responses, studies show that it contributes to an increase of the number of respondents that do not have an opinion on the subject when they actually do [24].

In order to simplify the survey analysis, the submission of medical notes and the sharing of medical exams are referenced as Task A and Task B, respectively.

In total, there were 6 questions in the survey, as follows:

1. Do you agree that creating an account on Service 1 is at least as simple than on Service 2?
2. Do you agree that logging into Service 1 is at least as simple as logging into Service 2?
3. Do you agree that executing Task A on Service 1 is at least as easy as on Service 2?
4. Do you agree that finishing the execution of Task A on Service 1 is at least as fast compared to the same task on Service 2?
5. Do you agree that executing Task B on Service 1 is as easy as on Service 2?
6. Do you agree that finishing the execution of Task B on Service 1 is at least as fast compared to the same task on Service 2?

The table 7.1 showcases the results of the survey. In the first column, there are the numbers corresponding to the questions and all the other columns have the number of responses for each of the Likert scale options.

Table 7.1: User Experience Survey Result

Question	Strongly Disagree	Disagree	Agree	Strongly Agree
1	0	8	2	0
2	0	4	6	0
3	0	0	0	10
4	0	0	0	10
5	0	0	0	10
6	0	0	0	10

Results Analysis

The analysis of the survey results was conducted by calculating the average for each of the questions. This average could be achieved by attributing a number to each of the Likert scale options from 1 to 4, with 1 being attributed to Strongly Disagree and 4 to Strongly Agree.

The table 7.2 demonstrates the average of the responses for each of the questions:

Table 7.2: User Experience Survey Result Analysis

Question	Average
1	2.2
2	2.6
3	4
4	4
5	4
6	4

A result above 2.5 or more in each of the questions means that the experience performing an action in the decentralised approach was at least as good as the same action performed in the centralised approach.

An average of 2.2 and 2.6 for the first and second questions, respectively, means that the users do not agree that creating an account in the decentralised approach is as simple as creating it in the centralised approach, unlike the login process.

As described in 5.4, the process that allows the user to sign up in the system has the users choose a storage provider in which their account should be created. Although there were different options from which the user could choose without any additional setup, the user has to perform an additional decision when compared to more traditional social applications.

Conversely, a result of 2.6 for the login process demonstrates that this process has similar user experience in both approaches. Similarly to the registration process, here the user also has to select the provider with which they want to log in, even though they can enter the system with a common username and password form after.

All the other questions that explore the simplicity and velocity of the tasks on the decentralised approach when compared to its counterpart got the maximum result from all the users, from which we can perceive that the user experience was not significantly affected by introducing the decentralised characteristics.

Development Time

This metric is significantly impacted by multiple external variables. The amount of time presented in this analysis includes not only the time spent coding and testing but also the infrastructure setup and the time researching and exploring the different libraries. For the centralised approach, the approximate number of hours spent was 50 hours, while on the decentralised approach the total number was approximately 110 hours.

The centralised approach, on the backend side, was built with tools with which there was a lot of knowledge and experience acquired already, so it is natural that the development of that component was smoother than the others.

Conversely, the backend of the decentralised approach was already built (i.e., Solid Pod) so the time was mainly spent on working out how to effectively communicate with it and how to better take advantage of the libraries in order to do it in the cleanest way possible.

Although it is interesting to consider the development time as a metric in order to compare the development of two approaches that share the same goal, the fact that there was absolutely no knowledge priorly acquired on the technology of one of them indicates that this metric should not have a significant impact in their comparison.

Finally, other metrics such as the availability of the documentation, tutorials and other resources impacts significantly the development productivity.

7.4 Summary

In this chapter, it is presented with the indicators and information sources defined having the objectives of the dissertation taken into account as well as the assessment methodology for each of them. Finally, we have analysed the results through the information gathered throughout the development of the present work.

Chapter 8

Conclusions

This dissertation allowed the exploration of a disruptive technology that proposes a significantly different approach on how social web applications can be built with security and privacy as the main focus.

The present work resulted in a Solid application that could cover different interactions that occur in modern social web applications. From the requirements analysis, to the architecture design and the implementation, it is possible to get an overview on how Solid applications can be built and how interactions with the Solid Pod are possible.

This final chapter walks through the research questions, contributions, results, limitations, future work and final assessment.

8.1 Research Questions

According to the research questions formulated in section 1.3, they are now answered taking into account the analysis performed during this dissertation.

1. How architecturally different are applications built with Solid at its core compared to those built with a more traditional web stack?

As described in section 5, from a frontend perspective the architecture of the applications is quite similar. The main difference is that while in the centralised approach most frontends will be communicating directly with a known backend service, the decentralised one will be communicating with whatever Solid Pod the user logged in with.

From a backend perspective, Solid Pods already have an immutable interface, while the centralised approaches can have a backend service more tailored for its needs.

Considering a high-level perspective, Solid applications and applications built with a Client-Server architecture are quite similar, in which the frontend component of both solutions can communicate with one or more services.

2. Can web social applications build with Solid have the same user experience as others currently built with a more traditional web stack?

In this dissertation, the developed Solid application performed different operations common in social web applications (e.g., submissions, reading and sharing of information) and as long as the frontend orchestrates the requests as Solid allows, it is possible to give the same look and feel to the applications.

However, this dissertation did not deep dive enough in concepts such as performance, which highly impact the user experience (e.g., long time retrieving information), or notifications management. For this reason, it is not possible to achieve a confident conclusion in this item without exploring this technology with more detail.

3. Do Solid applications require more man-hours than those built with a more traditional web stack?

The Solid ecosystem is currently in development, with documentation and tutorials covering the basics but lacking diversity. Information on topics such as performance tuning in multi-pod queries, notification management and clarity on the best practices of when and how new vocabularies should be created are scarce.

For applications that only require basic operations, once the developers get practice with the technology it is possible that the development time difference becomes insignificant when compared with systems built with a more traditional web stack.

4. Is Solid ready to achieve mainstream adoption?

This question has a high correlation with the previous ones as well as this dissertation as a whole. Mainstream adoption of a technology needs to be considered at the development and user levels.

From a developer perspective, the fact that the documentation diversity is still lacking makes it harder for more developers to try out the technology and come out with innovative implementations as well as collaborate in the core of the project.

Additionally, from a user perspective, in order to achieve mass adoption these applications need to support the same functionalities that popular social web applications provide at the current time of writing, offering more security and privacy without sacrificing core popular features.

For the current time, the conclusion is that Solid is still not ready for mainstream adoption and one could argue that, ultimately, it will only be when a Solid application achieves a significant amount of active users. This would shift attention to the technology and possibly accelerate its development.

8.2 Contributions

Along with the present work, a short paper was published [25] in the 1st Edition of Simpósio de Engenharia Informática in the Instituto Superior de Engenharia do Porto.

The implemented Solid application is also available on GitHub, so that other developers can explore and learn how to develop applications with this technology [1].

8.3 Results

The analysis and development achieved during this dissertation indicates that Solid is still not ready to achieve mainstream adoption.

From a developer perspective, the technology needs to have better and more diverse documentation when it comes to common real use cases for social web applications.

[1] https://github.com/pedropinto/linkedmed

Additionally, from a final user perspective, one could argue that Solid will be ready for mainstream adoption when one of its application gathers a significant amount of usage, offering security and privacy while being performant and easy to use.

8.4 Limitations

During the development of this dissertation, it was possible to identify multiple limitations when studying if Solid could achieve mainstream adoption.

At the moment, the user registration process is quite cumbersome (cf. section 5.4), given that a new browser tab is opened and not only no feedback is provided to the application itself when the Pod finishes the process, but also the user is not redirected to the application. This is a limitation in multiple levels, because not only it impacts the overall user experience, but also, for instance, can impact the collection of metrics (e.g., number of people who start the registration process but does not complete it or never log in the system for the first time).

The lack of diversity when it comes to documentation and tutorials is also an important limitation, since a showcase of what the technology can do and how its libraries work would facilitate new developers to experiment with it.

From a frontend development perspective, the Solid Pod can also limit the way the use cases are structured since it is not possible to open new APIs or endpoints in the Pod. This can lead to the frontend having to perform several requests, while in other centralised approaches this could be mitigated by having an API tailored for a specific need.

8.5 Future Work

There are topics that could not be investigated in the scope of the present work and its analysis would be important to better understand the technology potential.

A common feature in social web applications is the notification system, from which the users are notified when an action of their interest is performed in the system. Given the decentralised character of the Solid applications, where each user can connect their own Pod, this is an important topic to work on.

One of the main features of a Solid architecture is that it can utilise a knowledge network provided by the Linked Data. Since a Pod can query another Pod for information, it is important to understand how the performance can be impacted when the depth level of the queries (e.g., the number of Pods it has to query) exponentially grows.

Even though this dissertation focus significantly on the developer and end-user experience, it is also important to consider how the shift can happen from a business perspective. Businesses often rely on metric collection in order to make data-oriented decisions and how (or if) Solid architectures can handle this needs to be analysed.

8.6 Final Remarks

The Solid technology represents a major shift on how applications are built when compared to what is currently presented to the users. In a global perspective, this dissertation covers

the proposed objectives and approaches the essentials of what the Solid technology can provide.

Since there was no previous knowledge on the main topic and taking into account how different the studied decentralised approach is from what is currently implemented, this dissertation could not cover other important topics (as covered in section 8.5).

Solid is an exciting project able to create a massive impact in how the world approaches privacy and security with software. Hopefully, this dissertation along with the work that the community is developing can create an impact either by creating more adoption of this technology, or serving as a base layer for other similar and disruptive ideas that puts the users first.

Bibliography

[1] Amazon. *Small Business Impact Report*. Tech. rep. 2008. url: https://bit.ly/3nYV7aT (visited on 02/12/2020).

[2] Hilary Osborne and Hannah Jane Parkinson. *Cambridge Analytica scandal: the biggest revelations so far*. 2018.

[3] Business Wire. "Opus & Ponemon Institute Announce Results of 2018 Third-Party Data Risk Study: 59% of Companies Experienced a Third-Party Data Breach, Yet Only 16% Say They Effectively Mitigate Third-Party Risks". In: (2018). url: https://www.businesswire.com/news/home/20181115005665/en/Opus-Ponemon-Institute-Announce-Results-2018-Third-Party (visited on 12/02/2019).

[4] European Commission. *What does the General Data Protection Regulation (GDPR) govern?* (Visited on 02/05/2020).

[5] European Commission. *EU data protection rules*. url: https://ec.europa.eu/info/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules/eu-data-protection-rules%7B%5C_%7Den (visited on 12/02/2019).

[6] Midas Nouwens et al. "Dark Patterns after the GDPR: Scraping Consent Pop-ups and Demonstrating their Influence". 2020. url: http://arxiv.org/abs/2001.02479%7B%5C%%7D0Ahttp://dx.doi.org/10.1145/3313831.3376321 (visited on 12/02/2019).

[7] Andrei Vlad Sambra et al. "Solid: A Platform for Decentralized Social Applications Based on Linked Data".

[8] W3C. *RDF*. (Visited on 02/05/2020).

[9] W3C. *Resource Description Framework (RDF) Model and Syntax Specification*. url: https://www.w3.org/TR/PR-rdf-syntax/ (visited on 02/05/2020).

[10] Solid. *Solid Specification Draft*. url: https://github.com/solid/solid-spec (visited on 02/10/2020).

[11] W3C. *Web-ID TLS*. url: https://www.w3.org/2005/Incubator/webid/spec/tls/ (visited on 01/17/2020).

[12] Solid. *WebID-OIDC Authentication Spec*. url: https://github.com/solid/webid-oidc-spec (visited on 02/08/2020).

[13] M. Ali et al. "Blockstack Technical Whitepaper". In: (2019). url: https://blockstack.org/whitepaper.pdf.

[14] David D. Clark and Marjory S. Blumenthal. "The End-to-End Argument and Application Design: The Role of Trust". In: (2011). (Visited on 02/02/2020).

[15] Blockstack. *Authentication and Gaia*. url: https://docs.blockstack.org/storage/authentication.html (visited on 01/10/2020).

[16] Blockstack. *A decentralized storage architecture*. url: https://docs.blockstack.org/storage/overview (visited on 01/10/2020).

[17] Elastos Foundation. "Elastos white paper". In: (2018), pp. 1–26. url: https://www.elastos.org/static/file/elastos%7B%5C_%7Dwhitepaper%7B%5C_%7Den.pdf (visited on 01/02/2020).

[18] Juan Benet. "IPFS - Content Addressed, Versioned, P2P File System". 2014. (Visited on 01/16/2020).

[19] Elastos Academy. *Elastos DID Login Integration Webpage*. url: `https://elastos.academy/did-login-integration/` (visited on 01/14/2020).

[20] Peter A. Koen et al. "Fuzzy Front End: Effective Methods, Tools, and Techniques". url: `https://web.stevens.edu/cce/NEW/PDFs/FuzzyFrontEnd%7B%5C_%7DOld.pdf` (visited on 02/05/2020).

[21] David Pegg and Alex Hern. *Facebook fined for data breaches in Cambridge Analytica scandal*. 2018. url: `https://www.theguardian.com/technology/2018/jul/11/facebook-fined-for-data-breaches-in-cambridge-analytica-scandal` (visited on 02/10/2020).

[22] Alex Osterwalder et al. *Value Proposition Design*. Wiley, 2014. (Visited on 02/15/2020).

[23] Gina Winsky. *Customer Perceived Value: Understanding What Appeals to the Consumer*. url: `https://aircall.io/blog/customer-happiness/customer-perceived-value/` (visited on 01/12/2020).

[24] Melinda L. Edwards and Brandon C. Smith. *The Effects of The Neutral Response Option on The Extremeness of Participant Responses*. 2014. url: `https://blogs.longwood.edu/incite/2014/05/07/the-effects-of-the-neutral-response-option-on-the-extremeness-of-participant-responses/` (visited on 06/14/2020).

[25] Pedro Pinto, Pedro Piloto, and Nuno Bettencourt. "A study about web development frameworks focused on users' privacy". 2019. (Visited on 09/25/2020).

Appendix A

End-to-End Testing

```
describe('Log into the system', function() {
  browser.ignoreSynchronization = true; // for non-angular websites
  it('Log into the system', function() {

    browser.manage().timeouts().implicitlyWait(30000);

    browser.get("http://localhost:3000");

    browser.getWindowHandle().then(function(parentGUID){

      element(by.buttonText('Connect')).click();
      browser.sleep(3000);

      browser.getAllWindowHandles().then(function(allGUID){

        for(let guid of allGUID){

          if(guid !=parentGUID){
            // switch to the opened login window
            browser.switchTo().window(guid);

            break;
          }
        }
        element(by.buttonText('Inrupt')).click();
        browser.sleep(2000);
        element(by.id("username")).sendKeys("pedropinto");
        element(by.id("password")).sendKeys("password");
        element(by.buttonText('Log In')).click();

        browser.sleep(3000);
        // close the browser
        browser.close();
        // switch back to the parent window
        browser.switchTo().window(parentGUID);
      })
    })
  });

});
```

Listing A.1: System Authentication End-to-End Test