INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




# Updating the Solid Ecosystem to a scalable and microservices oriented project

**PEDRO MANUEL RIBEIRO PILOTO**
julho de 2020

POLITÉCNICO
DO PORTO

# Updating the Solid Ecosystem to a scalable and microservices oriented project

**Pedro Manuel Ribeiro Piloto**

Supervisors
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
July 2020

# Resumo

Assim como recentes escândalos (e.g. Cambridge Analytica) mostraram, muitas organizações de base digital recolhem dados de utilizadores, armazenam-nos em locais inacessíveis e utilizam-nos como ativos para gerar lucro. Enquanto isso, os utilizadores perdem completamente a propriedade e o controlo dos seus dados, restando-lhes apenas confiar nas empresas, nas quais terão, provavelmente, de preencher formulários idênticos e replicar toda a sua informação por múltiplas bases de dados.

De forma a dar seguimento ao desenvolvimento da *Web*, mantendo a privacidade do utilizador, este estudo tem o objetivo de entender e iterar sobre um projeto que atua precisamente no âmbito de reinventar uma *Web* mais transparente e centrada no utilizador. A par deste estudo serão exploradas alternativas, no sentido de perceber aquela que está mais orientada para servir como alternativa à actual *Web*.

O projeto com maior destaque nesta dissertação é o *Solid*, este foi fundado por Tim Berners-Lee e conta com uma comunidade forte que dedica os seus esforços a criar contribuições para aquele que é um dos projetos mais promissores neste ramo.

No decorrer desta dissertação são exploradas as suas potencialidades mas também as limitações actuais de escalabilidade derivadas da sua arquitetura monolítica. De forma a mitigar estas limitações de escalabilidade, o trabalho foca-se em detalhar a migração para uma solução orientada a micro-serviços, modelando as diferentes alternativas possíveis, bem como justificando as decisões arquiteturais mais relevantes.

Assim, a presente dissertação prende-se não só com o estudo do tema de descentralização da *Web* mas também em criar uma contribuição positiva e clara no sentido de mitigar os problemas de escalabilidade do sistema *Solid*.

**Palavras-Chave**: decentralization, data, storage, authentication, user privacy, micro-services

# Índice

# Lista de Figuras

# Lista de Tabelas

# Acrónimos

**ACL** Access Control List. v, 16, 40, 45–47, 58

**AHP** Analytic Hierarchy Process. i, vii, 19, 30, 31

**CAP** Consistency Availability Partition. 25, 39, 73

**DDD** Domain Drive Design. 23

**DID** Decentralized Identifier. 15, 17

**FAST** Function Analysis System Technique. i, ii, v, 19, 32

**IPFS** Interplanetary File System. 15, 17

**JWT** JSON Web Token. 22, 46, 57, 72

**LDP** Linked Data Platform. 8, 11, 16, 40

**MIT** Massachusetts Institute of Technology. 8, 71

**P2P** Peer to Peer. 15, 22

**POD** Personal Objects Datastore. v, 8–11, 14, 32, 35–38, 40, 43–46, 49, 50, 56, 65

**REST** Representational State Transfer. 8, 11, 13, 16, 21, 23, 27, 38, 39, 51, 53–56, 64

**UML** Unified Modeling Language. 27

**VPS** Virtual Private Server. vii, 65

**WAC** Web Access Control. 11, 16

**YAML** YAML Ain't Markup Language. 26

# Agradecimentos

Esta dissertação significa o culminar de três anos numa rotina completamente nova em regime de trabalhador-estudante. Foram duas empresas que me acolheram durante este período, bastante diferentes, mas ambas me proporcionaram a flexibilidade e a motivação para não desistir logo no primeiro semestre.

Agradeço aos amigos dos diferentes círculos, em particular aos que fiz durante a minha passagem nesta instituição, foram certamente os momentos que me proporcionaram os que mais contribuíram para manter o foco em terminar o mestrado.

Ao meu orientador Nuno Bettencourt, que foi certamente um dos professores mais marcantes que tive desde a licenciatura, agradeço pela disponibilidade e dedicação que teve desde o dia em que me apresentou este tema.

Por fim, agradeço à minha família que desde sempre me criou condições para ter acesso a todos e mais alguns recursos necessários, não só para alcançar este fim, mas também para me manter motivado.

# Capítulo 1

# Introdução

Numa era em que o principal ativo das empresas é a informação, a ponto de existirem empresas que trocam serviços altamente valiosos por informação pessoal (ex: Google, Facebook, etc) [1], surgem inevitavelmente questões sobre para que são realmente utilizados esses dados e se o utilizador está realmente ciente que estes estão disponíveis para toda e qualquer utilização que seja proveitosa para essas empresas [2].

Este problema torna-se crítico quando mentes geniais aliadas a poderes incríveis de processamento conseguem, literalmente, manipular opiniões que culminam em acontecimentos inéditos [3].

A descentralização da *Web* é um tema que assenta também em como tratar estes dados, não estivesse o tema adjacente ao conceito da nova "World Wide Web", e de entre outros projetos destaca-se o "Solid", fundado por Tim Berners-Lee. O *Solid* está ainda numa fase de desenvolvimento, estando, portanto, aberto a contribuições que possam de facto garantir melhorias e impacto positivo para o futuro [4].

Este projeto mantém atualmente duas ramificações, a original e uma outra desenvolvida e mantida pela *spinoff* denominada *inrupt* fundada também por Tim Berners-Lee. Esta última ramificação tem base a original mas com uma vertente mais comercial e uma visão mais ambiciosa no sentido de proliferar a utilização do Solid.

## 1.1 Motivação

A descentralização da *Web* não é um tema consensual, e certamente vai contra muitos modelos de negócio atuais. Porém, assim como se protege as nossas

propriedades, devem ser protegidos os nossos dados, os seus acessos e respetivas utilizações.

O paradigma atual gira em torno de oferecer serviços em troca de publicidade segmentada, com a justificação de sustentar custos de infraestrutura e de desenvolvimento [1]. Com um novo paradigma de *Web* descentralizada poderiam ser cortados os custos de infraestrutura, porque deixariam de ser necessários *"back-ends"*, e as aplicações cliente utilizariam apenas recursos do cliente. Restariam os custos desenvolvimento, que poderiam eventualmente ser sustentados pela extinção do conceito "gratuito" nas empresas de base digital [4].

É possível que este tema seja apenas um "não tema" e que não seja plausível a sua implementação de forma massiva, mas todo o contexto por trás e a quantidade de dados sensíveis (saúde, seguros, etc) que circulam todos os segundos nos trâmites do atual paradigma da *Web*, merece que sejam dedicados esforços e que pelo menos sirva de alavanca a uma reformulação profunda daquilo que é a *Web* que conhecemos hoje [4].

## 1.2 Objectivos

Os objetivos desta dissertação passam por explorar o tema da descentralização da *Web*. O estudo irá ter em conta as alternativas mais relevantes apresentadas neste contexto até ao dia de hoje. Destas alternativas, serão trabalhadas soluções de escalabilidade para o projeto que se apresentar mais promissor.

Assim, está também subjacente como objetivo a contribuição para um projeto *open-source* com alto potencial de revolucionar a *world wide web*, sendo esta contribuição no sentido de trabalhar numa perspetiva mais global com vista ao incremento da sua escalabilidade, através da migração para uma arquitetura orientada a micro-serviços, o baixo acoplamento, alta coesão e, consequentemente, a gestão rápida dos diferentes componentes.

## 1.3 Hipóteses

De forma a estabelecer um ponto de partida para a solução a desenvolver, apresentam-se as hipóteses que deverão ser corroboradas:

- H1 - A solução a desenvolver será mais escalável que a atual.
- H2 - A solução a desenvolver, nas mesmas condições de escalamento vertical e horizontal, não deverá ter menos *performance* que a atual.
- H3 - A experiência de utilização numa perspetiva de utilizador final não irá sofrer alterações.

## 1.4 Questões de Investigação

Estas hipóteses levantam uma série de questões que deverão servir também por base à investigação adjacente a esta dissertação, das quais destacam-se as seguintes:

1. **Como pode o *Solid* ser escalado horizontal e verticalmente?**

    - A escalabilidade horizontal e vertical correspondem a escalar os recursos de determinada instância e incrementar o numero de instâncias, respetivamente. Estas duas abordagens tem custos e benefícios diferentes, que devem pesados nas diferentes circunstâncias em que exista decréscimo de performance na instância atual.
    - O escalamento horizontal implica que possa existir redundância de determinado serviço ou de todo o monólito no caso de se tratar de uma arquitetura monolítica.

2. **É possível existir apenas uma implantação do *Solid* escalada de forma descentralizada e anónima?**

    - Esta questão está dependente da anterior questão relativa à possibilidade de escalamento horizontal de forma redundante, isto porque esse seria primeiro passo para tornar isto possível.
    - Por outro lado, o escalamento descentralizado iria implicar uma mecanismo de deteção de novas instâncias dos diferentes serviços na rede, para que possam servir tráfego.

3. **O mecanismo de autenticação utilizado atualmente pode ser mantido tendo em conta a migração para uma arquitetura orientada a micro-serviços?**

    - A arquitetura orientada a micro-serviços assenta no princípio de baixo acoplamento entre os diferentes serviços constituintes do sistema, assim o mecanismo de autenticação não deve criar uma forte dependência para o com os restantes serviços [5].

## 1.5 Metodologia de Investigação

Para a investigação, no âmbito do desenvolvimento deste projeto, foi utilizado o método *Design Science Research*. Esta abordagem aplica-se sobretudo a investigações com vista a construir novas soluções ou implantar soluções inovadoras sobre problemas que possam já ter outras resoluções descobertas [6].

Este processo inclui geralmente seis passos [6]:

1. Identificar o problema, assim como identificar os desafios da investigação e os potenciais benefícios da solução;
2. **Definição** dos objetivos para a solução;
3. **Desenho** e desenvolvimento dos artefactos;
4. **Demonstração** da resolução do problema com recurso aos artefactos desenvolvidos;
5. **Avaliação** da solução, comparando os objetivos com os resultados obtidos;
6. **Comunicação** do problema, dos artefactos constituintes da solução, a sua utilidade e a forma como realmente contribuir para criar impacto positivo.

## 1.6 Contribuições da Investigação

O trabalho resultante desta dissertação tem como objetivo criar uma possível solução escalável e capaz de alicerçar a proliferação de uma nova *world Wide Web* descentralizada. Seguem-se algumas contribuições resultantes do mesmo:

- *A Study About Web Development Frameworks Focused on Users' Privacy* [7]
- Repositório do *Node Solid Server* segundo arquitetura orientada a micro-serviços proposta [8]
- Repositório do *OpenID Connect provider* para *Node.js* [9].
- Repositório do *OpenID Connect relying party client* [10].
- Repositório do *OpenID Connect authentication manager* [11].
- Repositório do *Solid auth client* [12].
- Repositório da aplicação cliente *Solid* micro-serviços com funcionalidade de gestão de ficheiros [13]
- Repositório da aplicação cliente *Solid* monólito com funcionalidades clínicas [14].

## 1.7 Estrutura da Tese

Esta dissertação é composta pelos seguintes capítulos.

No capítulo 1 (Introdução) elabora-se uma introdução contextualizada ao tema, de forma a que o projeto e objetivos sejam melhor compreendidos.

O capítulo 2 (Estado da Arte) dedica-se ao estado da arte, no sentido de dar a entender as alternativas mais relevantes existentes até ao momento na descentralização da *Web*.

O capítulo 3 (Análise de Valor) incide na proposta de valor assente nos diferentes projetos referentes à descentralização da *Web* apresentados no estado da arte.

O capítulo 4 (Desenho) apresenta uma proposta de desenho para uma possível solução, tendo em conta que esta dissertação se debruçará sobre um dos projetos de descentralização da *Web* apresentados.

No capítulo 5 (Implementação) são revelados alguns pormenores e linhas gerais da implementação proposta.

O capítulo 6 (Avaliação e Experimentação) dedica-se à avaliação da proposta e verificação dos resultados obtidos.

Por fim, o capítulo 7 (Conclusão) apresenta as conclusões tiradas, assim como as adversidades e possibilidades de desenvolvimentos futuros no sentido de dar continuidade ao projeto atual.

# Capítulo 2

# Estado da Arte

O tema descentralização da *Web* não é novo, existindo até ao momento alguns projetos relevantes nesta área. Este capítulo consiste numa apresentação mais detalhada sobre a alternativas que demonstram mais potencial. É expectável que os detalhes estudados e apresentados permitam tirar mais conclusões no capítulo correspondente à análise de valor.

Assistimos hoje em dia a problemas sérios e crescentes relativos a manipulação de informação, consequentes de um paradigma de *Web* que foi explorado ao limite, por forma a criar modelos de negócio obscuros e com pouca transparência na recolha e utilização dos dados de utilizadores.

O modelo atual é um autêntico colosso, onde a informação se encontra replicada infinitamente, e a camada de persistência de dados nas diferentes arquiteturas é provavelmente aquela que mais destaque tem para todas as estruturas organizacionais, na medida em que persiste o ativo principal das empresas, os dados relativos a toda a atividade todos seus clientes [1]. Isto torna-se um problema, quando não existe respeito pelos utilizadores e a manipulação dos dados passa a ser o *"core business"*. Esta manipulação pode ser positiva, no sentido de criar valor para a empresa e para o cliente, ou pode ser negativa se estiver perante casos de uso de que coloquem em risco a privacidade ou que cedam os dados a terceiros não devidamente autorizados [2].

Um modelo de descentralização da web, vem sendo discutido desde tempos atrás, com várias abordagens possíveis, mas sempre com a garantia de que a informação será, em qualquer circunstância, propriedade do utilizador [4]. A emersão de um novo modelo para a *Web* está dependente de uma migração massiva do "antigo" para o novo, na medida em que estaria pendente da aceitação de plataformas digitais com infraestruturas de dimensão considerável e de toda a comunidade de engenharia informática. Esta adoção massiva por sua vez requer

um conjunto de ferramentas capazes de sustentar o seu funcionamento de forma eficaz e suficientemente escalável [15].

## 2.1 Modelos Descentralizados

Nesta secção são apresentadas diferentes alternativas que podem servir de base para um novo paradigma de *Web* descentralizada. E, como tal, são soluções robustas que oferecem ferramentas auxiliares para o desenvolvimento e posterior integração de novas aplicações.

### 2.1.1 Solid

O *Solid* é um projeto de descentralização da *Web* liderado por Tim Berners-Lee, o inventor da *World Wide Web*, desenvolvido em colaboração com o *Massachusetts Institute of Technology (MIT)*. Um dos conceitos fundamentais do *Solid* é o *Personal Objects Datastore (POD)*, este, na sua definição, é a unidade de armazenamento de recursos no *Solid*, recursos estes que pertencem a utilizadores e que poderão, com as devidas autorizações, ser utilizados por outras aplicações cliente.

A arquitetura implícita para o desenvolvimento de aplicações baseadas em Solid permite que os utilizadores escolham um *POD* à sua escolha, podendo este ser próprio ou recorrendo a um servidor partilhado para conceder às aplicações autorização para armazenar ou ler informação a partir do mesmo. Por si só, este facto permite que os utilizadores escapem dos tradicionais "silos de dados". Além disto, esses *PODs* podem oferecer diferentes granularidades de privacidade, confiabilidade, disponibilidade, proteção legal e reutilização de dados [16]. Desta forma, este componente do Solid confere a camada de persistência de informação dos utilizadores, podendo inclusive ser visto como o novo *backend* das aplicações, que poderão aceder à informação através da interface *Representational State Transfer (REST)* [17] disponibilizada por cada *POD*, baseada em *Linked Data Platform (LDP)*, uma recomendação da comunidade W3C, assim como um mecanismo baseado em queries SPARQL [18].

#### 2.1.1.1 Autenticação

A identidade dos utilizadores no Solid baseia-se em *URIs WebID*, estes funcionam como nomes universais que permitem identificar determinado utilizador de forma descentralizada. Segue-se um exemplo de um possível *WebID*:

*https://john.domain.com/profile/card#me*

Neste sentido, o Solid, tratando-se de uma plataforma descentralizada, tem requisitos diferentes da maioria das aplicações existentes. Precisa de estar assente

em mecanismos de autenticação *cross-domain* suficientemente descentralizados e isolados de qualquer serviço ou entidade [18].

Até ao momento, o mecanismo de autenticação a ser utilizado em determinada instância de um *POD*, pode ser um de dois: *WebID-TLS* ou *WebID-OIDC*. Esta capacidade de abstração do mecanismo de autenticação, permite que possam continuamente ser estudados e adotados novos mecanismos que se entendam relevantes [18].

***WebID-TLS*** foi o primeiro mecanismo de autenticação a ser implementado no Solid. Como forma de autenticar os *WebIDs*, em vez das tradicionais *passwords*, recorre a certificados criptográficos (guardados e geridos no navegador *Web* do utilizador) para comprovar a identidade do utilizador [19].

Este mecanismo, apesar de poder ainda ser utilizado, deixou de ser a configuração por omissão para instanciações de novos *PODs*. Isto, porque muitos navegadores *Web* removeram suporte ao elemento *KEYGEN*, que servia como base para este protocolo gerar certificados através do navegador *Web* [19].

***WebID-OIDC*** é também um mecanismo de autenticação baseado *WebID*, tendo a particularidade de ser baseado no protocolo *OpenID connect* (*OIDC*).

Este mecanismo adiciona ao fluxo normal de *OIDC* um passo extra que permite obter um URI *WebID* através de um *token OIDC* [20], conforme é possível perceber na figura 2.2.

Seguem-se alguns dos conceitos mais relevantes neste protocolo:

- *User* - Utilizador humano (pode ser também uma aplicação ou serviço se estes tiverem o seu próprio *webID*). Também chamado *Resource Owner*;
- *User-Agent* - Nome formal para navegador *Web*;
- *Identity Provider (OP)* - Um servidor de identidade baseado no protocolo *OpenID Connect*. Pode ser utilizado o próprio *POD* uma instância externa;
- *Resource Server* - Um servidor responsável por alojar recursos que o utilizador precisa eventualmente de aceder, tais como *HTML*, imagens, *Linked Data / RDF sources*, entre outros tipos de recursos;
- *Relying Party (RP)* - É um *POD* ou uma aplicação cliente que assenta a sua lógica num *token* providenciado por um servidor de identidade;

- *POD* - Unidade online de armazenamento pessoal, este componente é, assim, responsável por gerir a informação do utilizador, actuando como um *Resource Server*.

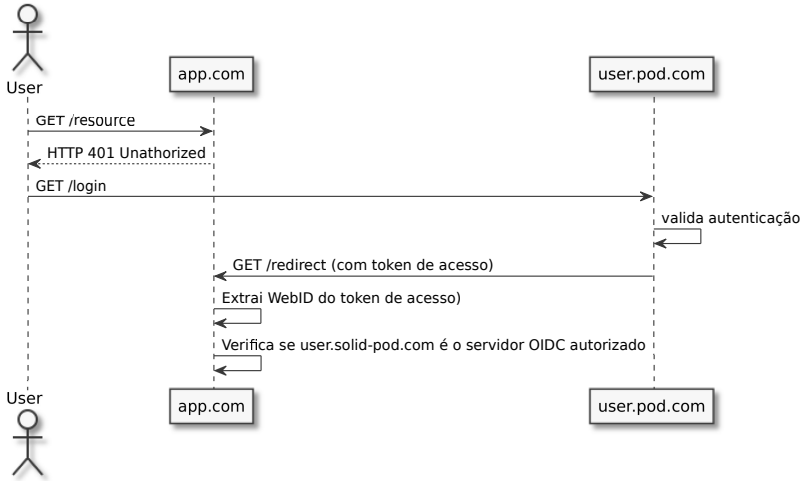


Figura 2.1: Diagrama de sequência *WebID-OIDC*

Este protocolo assemelha-se bastante ao protocolo OpenID Connect (*c.f* figura 2.2), adicionando ao mesmo dois passos [20]:

- Obter o *WebID* através do *token* de acesso;
- Verificar se o servidor indicado para autenticação é de facto o servidor de identidade autorizado para o *WebID* em questão.

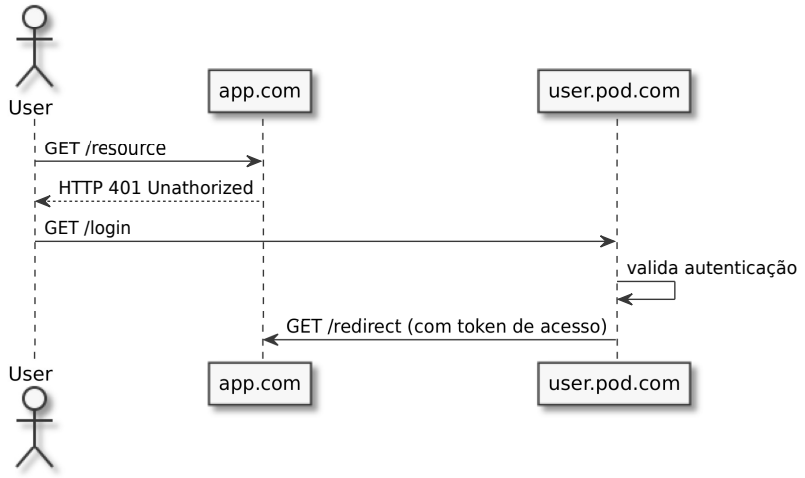


Figura 2.2: Diagrama de sequência *OpenID Connect*

#### 2.1.1.2 Autorização

A autorização no Solid é garantida por um componente com o nome *Web Access Control (WAC)*. Este componente consiste num sistema de controlo de acesso entre domínios de origem cruzada. Os seus conceitos principais baseiam-se em modelos de controlo de acesso a diferentes sistemas de ficheiros e as suas principais responsabilidades consistem em garantir acesso a agentes (utilizadores, grupos, entre outros) para conseguirem realizar ações (ler, escrever, editar, eliminar, entre outras) [21]. Seguem-se algumas das suas principais características:

- Os recursos são identificados por *URIs*, e podem referir-se a qualquer recursos disponível na *Web*;
- As politicas de controlo de acesso seguem uma estrutura declarativa;
- Utilizadores e grupos são também identificados *URIs (WebIDs)*;
- Agnóstico em termos de domínio, ou seja, as politicas de acesso a um determinado ficheiro podem conter utilizadores e grupos alojados em qualquer servidor de identidade existente.

#### 2.1.1.3 Armazenamento

O *POD* representa o componente que lida com toda a gestão de recursos do utilizador através da disponibilização de uma interface de aplicação *REST* (*c.f.* 2.3). Este componente permite ao *Solid* suportar dois tipos de informação:

- Recursos *Linked Data*, que podem recorrer às capacidades implícitas no Solid para aplicar lógica no lado da aplicação cliente;
- Todo o restante tipo de informação, como por exemplo imagens, documentos, vídeos, áudio, etc.

Desta forma, é possível construir aplicações baseadas em Solid com ou sem recursos sob a forma de *Linked Data* [18].

Os recursos são agrupados em contentores baseados em directórios (cumprindo a especificação *LDP Basic Container Spec* [18]).

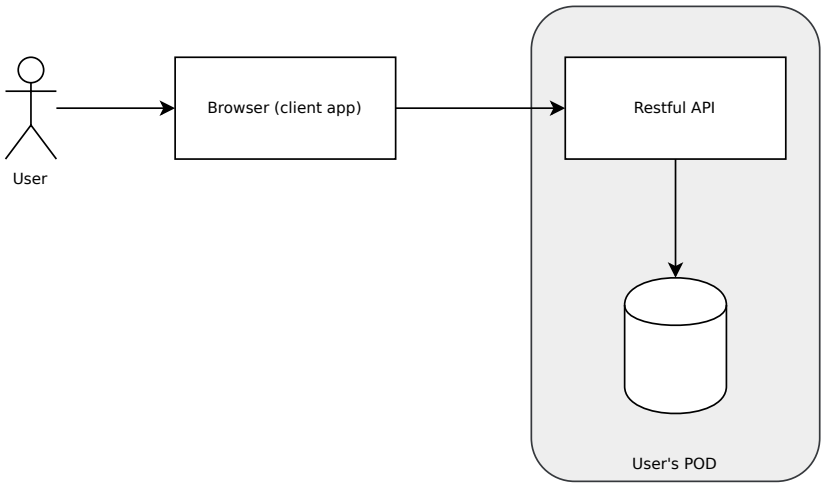


Figura 2.3: Representação arquitetura *Solid*

### 2.1.2 Blockstack

*Blockstack* é um projeto *open-source* que pretende construir uma rede de computação descentralizada assente na camada de transporte da Internet e protocolos de comunicação subjacentes. Para atingir este estatuto, está assente nas seguintes camadas:

- *Stacks Blockchain* - Permite aos utilizadores controlar e registar ativos digitais, como *usernames* e *passwords* ou executar contratos inteligentes [22];
- *Gaia* - Armazenamento descentralizado;
- *Blockstack Authentication* - Protocolo que permite autenticação descentralizada [22];
- Bibliotecas e *SDKs* - Ferramentas para facilitar o desenvolvimento de aplicações baseadas em *Blockstack* [22].

#### 2.1.2.1 Autenticação

Em termos de autenticação, o Blockstack consegue providenciar autenticação descentralizada através do mecanismo de autenticação por chave pública criptográfica.

A partir do login, a aplicação recebe três componentes essenciais de informação: o nome de utilizador, uma chave privada específica de aplicação e a localização do repositório para armazenar informação [22].

#### 2.1.2.2 Autorização

Os pedidos *REST* para aceder a informação de determinado utilizador devem ser acompanhado por um *token* de autenticação assinado e validado pelo repositório. Cada aplicação tem uma chave privada que confere acesso a uma partição especifica do repositório impedindo que, ao contrário do Solid, possa haver partilha de informação entre as diferentes aplicações.

#### 2.1.2.3 Armazenamento

Um dos conceitos chave da arquitetura desta plataforma é o sistema de ficheiros descentralizado chamado *Gaia Hub*, que permite aos utilizadores escolher e providenciar o seu repositório de informação, podendo estes ser partilhados entre a comunidade ou ser propriedade de cada utilizador.

Estes repositórios disponibilizam uma interface de aplicação *RESTful*, permitindo que as aplicações cliente devidamente autorizadas giram a informação do utilizador (*c.f.* figura 2.4).

Sendo que cada pedido deve ser acompanhado por um *token* autenticação assinado e validado pelo repositório. Cada aplicação tem uma chave privada que confere acesso a uma partição especifica do repositório impedindo a partilha de informação entre as diferentes aplicações [22].

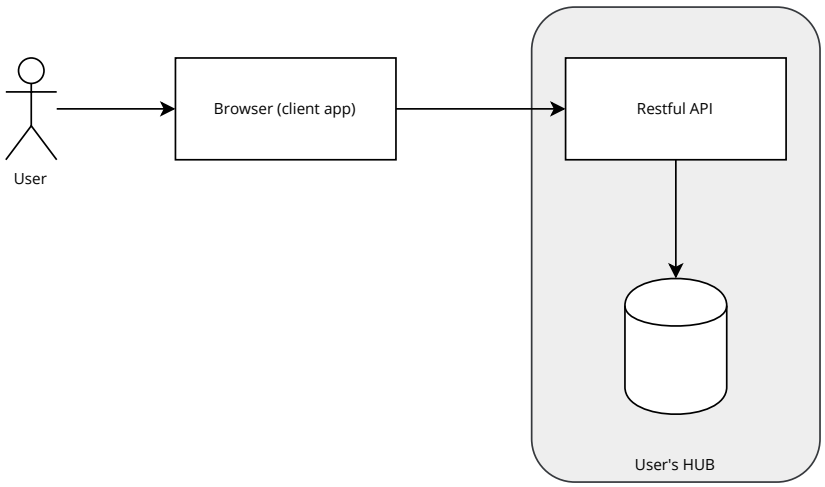


Figura 2.4: Representação arquitetura Blockstack

### 2.1.3 Diaspora

Diaspora é um projeto com uma visão um pouco mais reduzida que as restantes alternativas, na medida em que foca-se apenas em apresentar um protótipo daquilo

que poderia ser uma rede social descentralizada. Este projeto assenta em servidores de informação independentes, que o utilizador pode escolher ou até mesmo configurar um por si próprio [23].

#### 2.1.3.1 Autenticação

Relativamente a autenticação, não é adotado nenhum mecanismo fora do comum, sendo apenas o método comum *username/password*.

#### 2.1.3.2 Autorização

No contexto de autorização, este projeto dispõe do conceito *Aspect*, este permite criar regras de acesso a recursos para determinados conjuntos de utilizadores [23].

#### 2.1.3.3 Armazenamento

O conceito principal desta rede é o *POD* (assim como no Solid), este representa o componente que gere e persiste a informação do utilizador. A informação entre todas as instâncias deste componente é mantida em sincronização utilizando a tecnologia *WebSub*, conforme é possível observar na figura 2.5 [23].

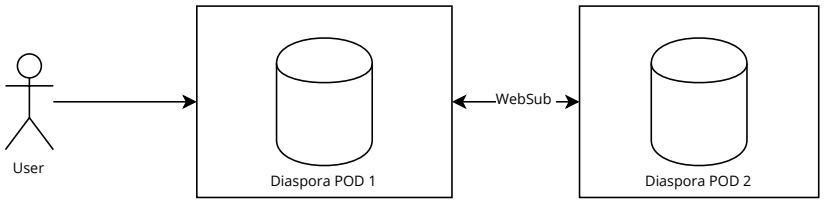


Figura 2.5: Representação arquitetura Diaspora

### 2.1.4 Elastos

Elastos incide no desenvolvimento de um novo paradigma de *Web* potenciado por tecnologia *blockchain* [24]. O foco principal incide não só na proteção dos dados, mas também na defesa dos direitos de autor. Conforme é possível ver na figura 2.6, a plataforma é constituída pelos seguintes quatro componentes [25]:

- *Elastos Blockchain*;
- *Elastos Runtime*;
- *Elastos Carrier*;
- *Elastos SDK*.

O conceito baseia-se num sistema operativo relativamente leve que armazena a informação e previne que as aplicações e serviços tenham ligação direta à *Internet*, sendo esta intermediada por meio do componente *Elastos Carrier*.

Toda a informação tem identificadores providenciados por *Blockchain* (pode ser Bitcoin ou uma rede de *Blockchain* alternativa [1]), sendo estes *IDs* verificados antes de os ativos digitais serem transmitidos entre as diferentes máquinas virtuais, utilizando comunicação *Peer to Peer (P2P)* [26] providenciada pelo *Elastos Carrier* [27].

#### 2.1.4.1 Autenticação

Relativamente à autenticação, esta plataforma introduz um *Decentralized Identifier (DID)*, construído na rede *Blockchain*, com a responsabilidade de providenciar identificadores seguros tanto para recursos como para utilizadores.

O identificador pode ser utilizado para efeitos de rastreamento, autenticação e estabelecimento de ligações seguras. De forma simplificada, o componente *Elastos Carrier* é a combinação entre *DID* e a comunicação *P2P*, assegurando assim que a informação é transmitida em segurança após a autenticação e a autorização serem devidamente validadas [25].

#### 2.1.4.2 Autorização

A autorização neste projeto fica a cargo do componente *Elastos Runtime*. Este implementa um sistema que valida o acesso de determinado utilizador a determinado recurso [25].

#### 2.1.4.3 Armazenamento

Elastos dispõe de um componente de armazenamento descentralizado, i.e. *Elastos Hive*. Recorre a um sistema de arquivos do tipo *Interplanetary File System (IPFS)* [28] como base.

Este oferece uma interface de aplicação para aceder à informação que está disponível nas diferentes regiões do planeta, garantindo, assim, fiabilidade às aplicações [25].

[1] Qualquer outra rede Blockchain, não necessariamente a que serve de base à conhecida criptomoeda Bitcoin

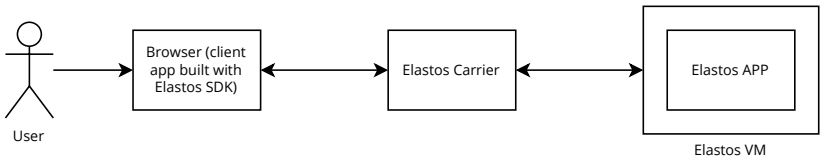


Figura 2.6: Representação arquitetura *Elastos*

### 2.1.5 Comparação Modelos Descentralizados

De forma a resumir, numa perspetiva de comparação, a informação apresentada sobre os diferentes modelos, estes são, de seguida, expostos na tabela 2.1, tendo por base três tópicos:

- Autenticação - De que forma os modelos descentralizados superam a necessidade validar a identidade do utilizador;
- Autorização - O mecanismo utilizado para garantir que o acesso restrito de um utilizador a determinado recurso;
- Armazenamento - Num contexto de descentralização como é a garantido o acesso a determinado recurso.

Tabela 2.1: Tabela de comparação entre modelos descentralizados

| - | Autenticação | Autorização | Armazenamento |
|---|---|---|---|
| *Solid* | Os utilizadores são identificados através de um *WebID*. Autenticação baseada em camada abstrata que permite utilizar o protocolo *WebID-TLS* ou *WebID-OIDC* [18]. | Garantida por um sistema *WAC*. Este sistema por sua vez é composto por documentos que contêm regras sobre a forma declarativas, denominadas *Access Control List (ACL)*, que indicam o tipo de acesso que cada *WebID* tem a determinado recurso [21]. | *PODs* disponibilizam uma camada de persistência baseada em *LDP*. Esta camada é acessível através de uma interface *REST* [18]. |

Tabela 2.1: Tabela de comparação entre modelos descentralizados

| - | Autenticação | Autorização | Armazenamento |
|---|---|---|---|
| *Blockstack* | Autenticação descentralizada através do mecanismo de autenticação por chave pública criptográfica [22]. | Chave privada específica de aplicação confere o acesso a determinada partição do repositório e, consequentemente, aos seus recursos [22]. | Sistema de ficheiros descentralizado chamado *Gaia Hub*, que permite aos utilizadores escolher e providenciar o seu repositório de informação [22]. |
| *Diaspora* | Não é adotado nenhum mecanismo fora do comum, sendo apenas o método comum *username/password* [23]. | Autorização de acesso baseada no conceito *Aspect*, este funciona como um agrupador de utilizadores que tem acesso a um determinado recurso [23]. | Os diferentes *PODs* existentes pelo mundo conferem o funcionamento da rede social e mantêm-se sincronizados utilizando a tecnologia *WebSub* [23]. |
| *Elastos* | Introduz um *ID* descentralizado (DID), construído na rede Blockchain e providencia *IDs* confiáveis para tudo e para todos [25]. | A camada *Elastos Runtime contém um sistema de gestão de permissões que confere ou o acesso de determinado utilizador a um recurso*[25] | Recorre a um sistema de arquivos do tipo IPFS como base [27]. |

## 2.2 Métodos

No decorrer do desenvolvimento do projeto, foram estudados e aplicados diferentes métodos, de forma a responder a diferentes necessidades surgidas. Neste contexto, os métodos consistem em todas as metodologias e técnicas utilizadas com vista a explorar conceitos. Estes tiveram principal impacto nas fase de análise, contribuindo para facilitar processos de decisão e estabelecer metas.

O *Fuzzy Front End* é a fase inicial do processo de desenvolvimento de novos produtos, correspondendo ao momento da identificação do problema ou à captação de oportunidades para o projeto.

Esta fase caracteriza-se por processos e decisões não estruturadas que tem como objetivo a identificação das necessidades dos potenciais clientes.

A fase seguinte é o processo de desenvolvimento do novo produto, sendo esta uma fase onde as ideias já estão mais estruturadas e os objetivos definidos.

A terceira fase é a comercialização do produto e corresponde à fase em que o produto é introduzido no mercado [29].

### 2.2.1 New Concept Development

O processo de inovação é divido em três componentes (*c.f.* figura 2.7): *Fuzzy Front End*, o processo *New Product Development* e comercialização [29].

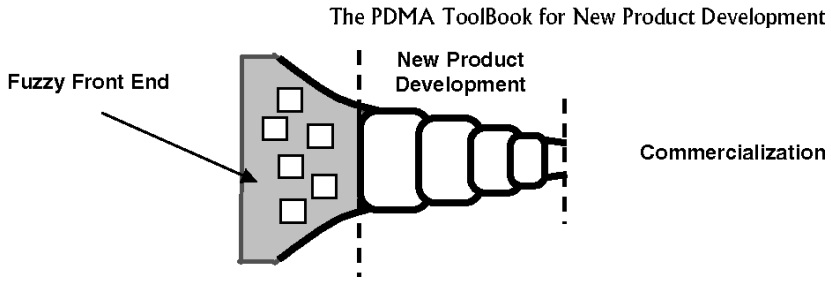


Figura 2.7: Representação processo de inovação

O modelo *New Concept Development Model* é a representação das principais atividades do Fuzzy Front End. Este consiste em três componentes:

- Motor: Mecanismo que impulsiona as atividades de liderança e estratégia de negócio da organização;
- Cinco Processos: São os processos que compõem o centro do modelo, que têm em conta a visão geral, a estratégia pretendida e a sua motivação;
- Fatores de Influência: Variáveis não controláveis de origem interna ou externa ao projeto que afetam a sua inovação.

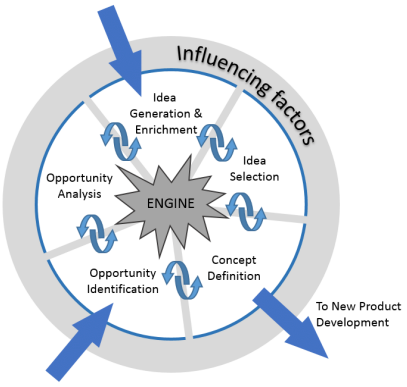


Figura 2.8: Representação New Concept Development Model

O modelo apresenta duas vias de entrada e uma única saída, sendo este iniciado pela identificação de uma oportunidade ou pela geração de uma ideia. Essa irá posteriormente interagir com os restantes componentes do modelo.

### 2.2.2 Analytic Hierarchy Process (AHP)

Este método foi criado por Tomas L. Saaty em 1980, sendo um dos métodos mais conhecidos de apoio à decisão. O seu principio baseia-se em desconstruir a ideia em fragmentos cada vez mais pequenos de modo a tornar a sua análise mais fácil.

Para atingir esse fim, o método divide-se em três fases: Divisão Hierárquica, Definição de Prioridades e Consistência [30].

A divisão hierárquica do AHP consiste na definição do objetivo, dos critérios de resolução do problema e, por último, nas alternativas que resolvem o problema. A lógica é que tendo este conceitos organizados hierarquicamente, conseguimos perceber a relação entre eles.

Depois da divisão hierárquica, é necessário definir prioridades entre os diferentes pares de critérios escolhidos. Esta priorização é subjetiva, sendo necessário calcular o índice de consistência de modo a validar a decisão. Para fazer a priorização de critérios é possível utilizar-se a tabela que apresenta a relação entre cada par de critérios utilizando a escala de Saaty. Esta escala consiste numa escala numerada de 1 a 9, em que 1 corresponde a uma igualdade de importância entre critérios, 3 corresponde a uma fraca diferença de importância e 9 corresponde a uma diferença de importância absoluta.

Por fim, é necessário garantir a consistência da decisão, uma vez que a priorização dos critérios é baseada em opinião e é subjetiva. Para isso são calculados o índice de consistência e a razão de consistência. Desta forma é possível medir a consistência. Se o resultado do cálculo do rácio de consistência for menor que 0.1, consideram-se os critérios consistentes.

### 2.2.3 Function Analysis System Technique (FAST)

Este modelo permite apresentar de forma gráfica as relações entre as funcionalidades de um determinado projeto, produto, processo ou serviço, tendo por base as questões: “Como” e “Porquê?” [31].

Esta técnica ajuda, desta forma, a pensar no problema de forma mais objetiva e orientar os objetivos do seu desenvolvimento em função das relações entre as funcionalidades [31].

### 2.2.4 Modelo de Negócio Canvas

O modelo de negócio Canvas foi apresentado por Alex Osterwalder na sua tese de doutoramento *The Business Model Ontology - A Proposition In A Design Science*

*Approach, em 2004*. Este modelo permite, através de uma representação gráfica, mostrar como determinada empresa vende o seu produto e como ganha dinheiro [32].

### 2.2.5 Avaliação

Avaliação consiste na medição ou avaliação de uma determinada grandeza, utilizando uma metodologia ou técnica específica de modo a validar ou refutar uma hipótese. Sendo, assim, para a realização de um teste de avaliação, a definição de grandezas, hipóteses e as metodologias de avaliação [33].

## 2.3 Tecnologias

As tecnologias, tendo por base os resultados dos processos de análise, servem como veículos para desenhar e atingir os resultados. Assim, são apresentadas, de seguida as tecnologias exploradas no seguimento deste projeto, as mesmas são separadas pelas seguintes categorias:

- Arquiteturas;
- Padrões;
- Linguagens;
- Ferramentas.

### 2.3.1 Arquiteturas

Arquitetura de software corresponde ao processo de converter características de software como flexibilidade, escalabilidade, viabilidade, reutilização e segurança numa solução estruturada que permita responder às necessidades técnicas de negócio [34].

Apresentam-se, de seguida, nesta secção, algumas das arquiteturas de software estudadas no âmbito desta dissertação.

#### 2.3.1.1 Arquitetura orientada a micro-serviços

Uma arquitetura orientada a micros-serviços define uma configuração, na qual os componentes são aplicações independentes [35].

Essas aplicações independentes comunicam utilizando *RESTful Web Services* ou troca de mensagens [5].

Seguem-se algumas características de sistemas orientados a micro-serviços [36]:

- Micro-serviços em execução de forma independente;
- Sistema é partido em micro-serviços, tendo por base *bounded context* (*c.f.* secção 2.3.2.2) e/ou capacidades de negócio;
- O conceito "produto" ganha relevo face ao conceito "projeto";
- Aplicações devem utilizar canais de comunicação relativamente simples, como protocolo *REST* ou troca de mensagens;
- Diferentes micro-serviços podem seguir arquiteturas e implementações diferentes, dependendo das suas especificidades e das decisões levadas a cabo pelas equipas de desenvolvimento;
- Seguindo as boas práticas de desenvolvimento de micro-serviços, cada aplicação deve ter a sua própria unidade de persistência de dados, podendo inclusive a tecnologia adjacente à mesma ser diferente para cada micro-serviço;
- *Pipelines* de desenvolvimento independentes;
- *Deploys* independentes.

#### 2.3.1.2 Arquitetura cliente servidor

A arquitetura cliente servidor é aquela que é mais utilizada no desenvolvimento de Software à data desta dissertação.

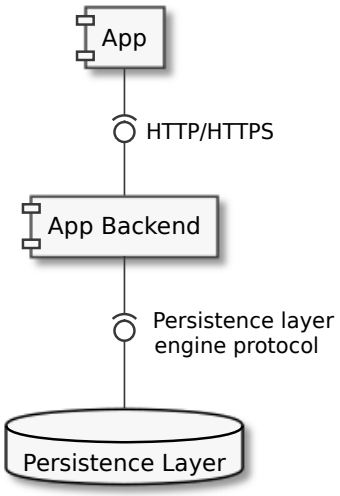


Figura 2.9: Diagrama de Componentes arquitetura cliente servidor

Nesta, existem tipicamente três camadas (*c.f.* figura 4.1) [37]:

- Cliente - Responsável por gerir interação com o utilizador final;
- Servidor - Camada detentora da lógica de negócio responsável por servir um ou mais clientes;
- Persistência - Responsável pela persistência dos dados.

#### 2.3.1.3 P2P

*P2P* é um formato de rede de computadores, cuja principal característica é descentralização das funções convencionais de rede. Neste conceito, o computador de cada utilizador conectado acaba por realizar simultaneamente as funções de servidor e de cliente [26].

O seu principal objetivo é a transmissão de recursos e o seu surgimento possibilitou a partilha em massa de músicas e filmes [26].

### 2.3.2 Padrões

Um padrão consiste numa solução que resolve sucessivamente um problema comum no contexto das diferentes fases de desenvolvimento de software, sendo nesta secção expostos padrões de desenho e padrões arquiteturais.

Por norma, um padrão não corresponde a uma solução final, mas sim a um guia ou um *template* que pode ser adaptado a diferentes situações [38].

#### 2.3.2.1 JSON Web Token (JWT)

*JWT* é um padrão (RFC-7519) da Internet.

Este padrão define como devem ser transmitidos e armazenados objetos em formato JSON de forma compacta e segura entre diferentes aplicações. Um dos seus grandes fatores de adoção é o facto de ser assinado digitalmente, garantindo que os dados nele contidos podem ser validados a qualquer momento [39].

O objeto deve ser constituído por três secções:

- *Header* - Esta secção indica o tipo de token e o o algoritmo de criptografia utilizado para a sua assinatura digital;
- *Payload* - Contém todo objeto *JSON* inicialmente criado, este na maioria dos casos deverá conter informação sobre o utilizador autenticado;
- *Signature* - Este campo tem como principal objetivo garantir a integridade do *token*.

#### 2.3.2.2 Bounded Context

*Bounded Context* é um padrão crucial em *Domain Drive Design (DDD)*, na medida em que este é o foco da fase de desenho de grandes modelos.

Para lidar com grandes modelos, *DDD* recorre a divisão em diferentes *bounded contexts*. Estes tem como objetivo delimitar o domínio complexo em contextos baseados na intenção do negócio [40].

#### 2.3.2.3 REST

*REST* é um estilo arquitetural que torna mais fácil a comunicação entre sistemas. Sistemas compatíveis com *REST*, são caracterizados por manterem um elevado desacoplamento entre o cliente e o servidor. Neste estilo arquitetural, as aplicações cliente fazem pedidos para obter ou alterar recursos em aplicações servidor, estes pedidos são constituídos por [41]:

- Verbo HTTP - Define o tipo de ação a realizar, pode ser *GET*, *POST*, *PUT*, *PATCH* ou *DELETE*;
- *Header* - Permite ao cliente passar detalhes sobre o pedido;
- Caminho - URL onde o pedido deve ser respondido;
- *Body* - Mensagem que contém a informação sobre o recurso a ser criado ou alterado.

#### 2.3.2.4 API Gateway

Uma *API Gateway* é uma interface que se situa entre a aplicação cliente e os micro-serviços. Esta interface é utilizada para criar uma camada de abstração entre as aplicações e as APIs providenciadas pelos diferentes micro-serviços, permitindo, assim, um elevado desacoplamento entre as diferentes APIs do sistema. A adoção deste padrão confere ao sistema vantagens como as seguintes destacadas:

- Escalabilidade - No futuro, a separação dos serviços existentes ou a criação de novos, não adicionará complexidade para além de alterar as regras de encaminhamento na API Gateway.
- Segurança - A camada da API Gateway deverá centralizar responsabilidades como *throttling* e inibição de tráfego com base em características do pedido (por exemplo *IP* de origem). O componente Solid-ID, não deverá estar "escondido" atrás da API Gateway, porque se trata de uma camada que deverá estar visível para o cliente, devendo esta ser utilizada para obter posteriormente acesso às funcionalidades expostas através da API Gateway [].

#### 2.3.2.5 Troca de Mensagens Assíncrona

A troca de mensagens assíncrona consiste numa forma de troca de mensagens em que a aplicação que emite a mensagem não conhece o(s)seu(s) recetor(es), contribuindo para o incremento do desacoplamento entre as diferentes aplicações de um sistema. Dois padrões associados a este mecanismo de troca de mensagens são *Message Queueing* e *Publish/Subscribe* [42].

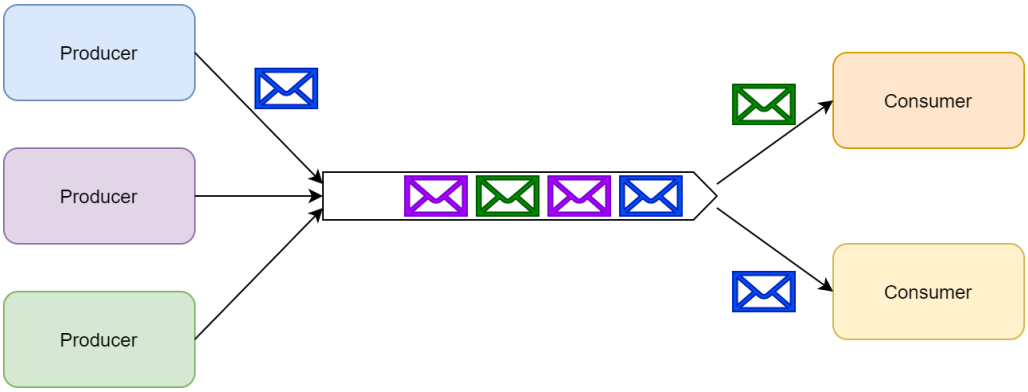


Figura 2.10: *Message Queueing*

*Message Queueing* consiste num padrão em que as aplicações emissoras podem enviar para uma mesma fila, no entanto, existindo apenas uma fila, qualquer mensagem poderá ser consumida apenas por um consumidor (*c.f.* figura 2.10).

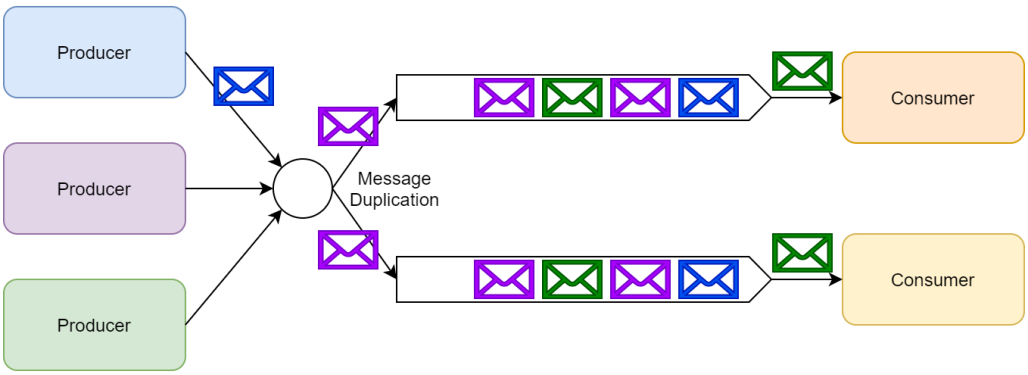


Figura 2.11: Publish/Subscribe

*Publish/Subscribe* tem um comportamento idêntico ao padrão *Message Queueing*, adicionando a este uma camada que permite que a mesma mensagem seja enviada para mais do que uma fila (*c.f.* figura 2.11).

#### 2.3.2.6 Command Query Responsability Segregation (CQRS)

Nas arquiteturas mais tradicionais, é utilizado o mesmo modelo de dados tanto para consultar como para atualizar a camada de persistência da aplicação. Esta abordagem é simples e funciona bem para operações CRUD básicas [35].

Por outro lado, em aplicações mais complexas onde as cargas de leitura e de escrita são assimétricas, esta abordagem torna-se mais problemática e pode colocar em risco a escalabilidade do sistema [43].

Neste contexto, o padrão CQRS incide em separar o modelo de leitura do modelo de escrita, existindo múltiplas abordagens para a sua implementação [44].

#### 2.3.2.7 Service

Este padrão indica que deve ser estabelecida uma camada na aplicação responsável por gerir a lógica de pedidos a serviços externos. Este padrão tem como principal objectivo abstrair a lógica de funcionamento dos diferentes serviço da lógica de negócio [45].

#### 2.3.2.8 Teorema Consistency Availability Partition (CAP)

O teorema *CAP* é uma ferramenta utilizada para ajudar a tomar decisões durante o desenho de sistemas distribuídos.

Eric Brewer inferiu, no ano de 2000, que qualquer sistema distribuído é construído com base num equilíbrio entre consistência, disponibilidade e tolerância a falhas. Afirmação corroborada em 2002 por Seth Gilbert e Nancy Lynch [43].

Este teorema indica que sistemas distribuídos apenas podem garantir duas das seguintes três propriedades [46]:

- Consistência - Garantia que qualquer nó num sistema distribuído retorna a mesma, mais recente, escrita com sucesso. Consistência consiste em qualquer cliente ter a mesma vista de determinada informação num determinado momento;
- Disponibilidade - Todos os nós funcionais retornam resposta para os pedidos de leitura e escrita num tempo razoável de resposta;
- Tolerância a falhas - O sistema continua a funcionar e mantêm consistência mesmo em cenários de falha de rede.

#### 2.3.2.9 Event Sourcing

*Event sourcing* é um padrão que tem como objetivo garantir que as mudanças realizadas numa aplicação são armazenadas como uma sequência de eventos [47].

Estes eventos persistidos podem ser obtidos e utilizados para reconstruir estados da aplicação. Neste sentido, este padrão é muitas vezes utilizado em conjunto com o padrão CQRS [48].

### 2.3.3 Linguagens

Nesta secção incluem-se as linguagens de programação e modelação estudadas e utilizadas no decorrer da dissertação. Nesta secção incluem-se, portanto, linguagens que tiveram principal impacto no desenho e implementação da solução final.

#### 2.3.3.1 JavaScript

*JavaScript* é uma linguagem de programação criada em 1994 com o principal objetivo de adicionar funcionalidade e interação a *web sites* [49]. Assim, o *JavaScript* foi originalmente desenhado para ser usado exclusivamente no *front-end*, porém a introdução do *engine V8* pela Google e, consequentemente, as melhorias de velocidade funcionalidade, introduziram não só o desenvolvimento de novas frameworks de *front-end*, mas também uma nova linguagem de programação paradigma para o *back-end*, o *Node.js* (*c.f. 2.3.3.2*) [50].

#### 2.3.3.2 Node.js

Node.js pode ser considerado, mais do que uma linguagem de programação, um ambiente de execução multi-plataforma para JavaScript [51].

Uma das grandes particularidades do *Node.js* é o facto de não criar uma nova *thread* para cada pedido, optando por adotar mecanismos assíncronos para lidar com operações de *I/O* e, desta forma, prevenir que exista código bloqueante. Estes mecanismos conseguem garantir alto desempenho sem a necessidade de lidar com problemas de concorrência entre *threads* [51].

#### 2.3.3.3 YAML Ain't Markup Language (YAML)

*YAML* foi desenhada para ser uma linguagem com características capazes de a tornar fácil de perceção. Esta linguagem é maioritariamente utilizada em ficheiros de configuração em diversas ferramentas (e.g. docker) e frameworks (e.g. Spring Boot) [52].

#### 2.3.3.4 Unified Modeling Language (UML)

*UML* é uma forma de criar representações visuais de um software, permitindo, também, fazer uma análise orientada a objetos através de notações gráficas [53].

Foi inventada com objetivo de criar uma linguagem de modelação gráfica padrão para o design, arquitetura e execução de sistemas de software complexos [54].

### 2.3.4 Ferramentas

Por ferramentas entende-se todas as aplicações e conjuntos de funcionalidades disponíveis com vista a criar valor nos diferentes processos de desenvolvimento [55].

#### 2.3.4.1 RabbitMQ

*RabbitMQ* é uma implementação de um *Message Broker*, este suporta, de forma nativa, os dois padrão de troca de mensagens assíncrona (*c.f.* secção 2.3.2.5 e 2.3.2.5). Para a implementação da lógica no sentido do padrão *Publish/subcribe*, *RabbitMQ* introduz o conceito de *Exchange*, este componente é responsável por reencaminhar a mesma mensagem para as diferentes filas que subscreveram aquele tipo de mensagem [56].

#### 2.3.4.2 Apache Kafka

*Apache Kafka*, por outro lado, é uma plataforma distribuída de streaming. Ao contrário de *RabbitMQ*, que é baseado em filas e *exchanges*, a camada de armazenamento do *Kafka* é implementada utilizando transações de *logs* particionadas [56].

Esta plataforma introduz o conceito de tópico, este representa uma categoria para a qual será mantida uma partição ordenada e imutável de *logs* com as mensagens recebidas.

#### 2.3.4.3 JMeter

*JMeter* é uma ferramenta *open-source* desenhada para realizar testes funcionais de carga, com vista a obter dados relativos a performance de determinado serviço [57].

Esta ferramenta disponibiliza a funcionalidade de executar pedidos contra interfaces de aplicação *REST* através da configuração dos seguintes parâmetros [58]:

- Caminho - O caminho que permite chegar ao serviço;

- Método - Método HTTP do *endpoint* específico;
- Numero de *Threads* - Cada *thread* permite simular um utilizador a executar pedidos, sendo que múltiplas *threads* irão ser executadas em paralelo e, por consequência, simular múltiplos utilizadores a fazer pedidos em simultâneo;
- Período *Ramp-up*: Variável em segundos que indica quão gradual devem ser criadas novas *threads*. A divisão do numero de *threads* pelo valor configurado nesta variável indica o numero de novas *threads* que serão criadas por segundo até que seja atingido o numero total;
- Total repetições - Numero de pedidos que cada *thread* irá fazer. Assim que todas terminarem de executar o valor total de repetições, está concluído.

#### 2.3.4.4 Node Package Manager (NPM)

*NPM*, conforme sugere o nome, é um gestor de dependências *Javascript*, que vem pré-instalado com o *NodeJS*. Esta ferramenta permite ligar-se a um repositório pré-definido ou configurar repositórios customizados, nestes é possível encontrar diferentes bibliotecas desenvolvidas e publicadas que podem ser utilizadas em novos projetos [59].

#### 2.3.4.5 Containers

A tecnologia subjacente à utilização de *containers* consiste num mecanismo de criar uma espécie de contentor com o código base da aplicação e as suas dependências, e, posteriormente, este executar de formar isolada de outros processos [60]. Estes *containers* podem partilhar o *kernel* do sistema operativo, apresentando, assim, uma solução mais leve e robusta que as tradicionais máquinas virtuais [61].

## 2.4 Sumário

Estudados os diferentes conceitos relevantes para esta dissertação, é, agora, possível correlacionar os modelos descentralizados (*c.f.* secção 2.1) com os métodos (*c.f.* secção 2.2) e elaborar uma análise de valor (capítulo 3) sobre as diferentes alternativas apresentadas.

Por outro lado, as tecnologias (*c.f* secção 2.3) servirão de base não apenas não apenas ao desenho da solução (capítulo 4), mas também à implementação (capítulo 5) e à avaliação da solução final desenvolvida (capítulo 6).

# Capítulo 3

# Análise de Valor

A análise das diferentes alternativas de modelos descentralizados apresentados na secção 2.1 consolida-se tendo em conta certos parâmetros e tem como objetivo decidir a solução sobre a qual serão dedicados esforços no sentido de criar contribuições com potencial de melhoria de escalabilidade.

## 3.1 Análise de Valor

Aplicando o modelo de Peter A. Koen, *New Concept Development* (*c.f* 2.2.1), é possível mapear uma oportunidade para um conceito, seguindo todos elementos chave do modelo e percebendo assim se esta oportunidade representa ou não valor para a organização.

### 3.1.1 Identificação da Oportunidade

Neste contexto, a oportunidade surge no sentido de prevenir o acontecimento de escândalos como o conhecido Cambridge Analytica e outros *"leaks"* [3] de informação que acontecem a um ritmo cada vez superior numa *Web* baseada num paradigma que parece cada vez mais desajustado para a realidade que vivemos nos dias de hoje.

### 3.1.2 Análise da Oportunidade

O conceito "descentralização da *Web*" não é novo, mas tem ganho ênfase nos últimos tempos e existem esforços notáveis para tornar este conceito numa realidade e tornar viável o surgimento de uma *Web* mais livre e mais transparente.

Esta dissertação passa inicialmente por perceber que alternativas existem e qual pode ter mais potencial numa vertente de adoção massiva a médio prazo.

Os projetos que serão analisados são aqueles que foram abordados com maior destaque no estado da arte (*c.f. secção 2.1*): *Solid*, *Blockstack*, *Diaspora* e *Elastos*.

### 3.1.3 Enriquecimento da ideia

Nesta fase é importante perceber os fatores que se apresentam como mais relevantes para uma *framework* capaz de potencializar um novo paradigma de *Web* descentralizada:

- Gestão de informação - No sentido de prevenir escândalos de acessos indevidos a informação pessoal e combater a monopolização dos dados, existe necessidade de garantir a defesa da sua propriedade.
- Mecanismo de autenticação e autorização - A propriedade dos dados é assegurada por mecanismos de autenticação e autorização.

### 3.1.4 Seleção da ideia

Para esta componente foi utilizada a técnica AHP (*c.f.* secção 2.2.2).

#### 3.1.4.1 Divisão Hierárquica

A primeira fase desta técnica é apresentar os critérios mais relevantes para a tomada de decisão.

Neste sentido, para perceber qual a relevância de cada critério é apresentada uma tabela com a matriz dos critérios (*c.f.* tabela 3.1) e a tabela com a matriz de critérios normalizada (*c.f.* tabela 3.2).

Tabela 3.1: Matriz de critérios

| - | Autenticação | Autorização | Armazenamento |
|---|---|---|---|
| Autenticação | 1 | 1 1/2 | 1,75 |
| Autorização | 2/3 | 1 | 1 1/4 |
| Armazenamento | 4/7 | 4/5 | 1 |
| Soma | 2 5/21 | 3 3/10 | 4 |

Tabela 3.2: Matriz de critérios normalizada

| - | Autenticação | Autorização | Armazenamento | Média |
|---|---|---|---|---|
| Autenticação | 21/47 | 5/11 | 7/16 | 45% |
| Autorização | 14/47 | 10/33 | 5/16 | 30% |
| Armazenamento | 12/47 | 8/33 | 1/4 | 25% |
| Soma | 1 | 1 | 1 | 100% |

#### 3.1.4.2 Definição de Prioridades

Após os critérios e as devidas prioridades estarem identificadas, segue-se a apresentação das diferentes alternativas, bem como a sua prestação face aos diferentes critérios: Autenticação (tabela 3.3), autorização (tabela 3.4) e armazenamento (tabela 3.5).

Tabela 3.3: Prioridades relativas autenticação

| - | Solid | BlockStack | Diaspora | Elastos | Prioridade Relativa |
|---|---|---|---|---|---|
| Solid | 41% | 48% | 44% | 33% | |
| BlockStack | 21% | 24% | 22% | 33% | 25% |
| Diaspora | 10% | 12% | 11% | 11% | 11% |
| Elastos | 28% | 16% | 22% | 22% | 22% |

Tabela 3.4: Prioridades relativas autorização

| - | Solid | BlockStack | Diaspora | Elastos | Prioridade Relativa |
|---|---|---|---|---|---|
| Solid | 26% | 35% | 10% | 45% | 29% |
| BlockStack | 18% | 25% | 32% | 21% | 24% |
| Diaspora | 43% | 13% | 17% | 10% | 20% |
| Elastos | 13% | 28% | 42% | 24% | 27% |

Tabela 3.5: Prioridades relativas armazenamento

| - | Solid | BlockStack | Diaspora | Elastos | Prioridade Relativa |
|---|---|---|---|---|---|
| Solid | 23% | 35% | 8% | 44% | 28% |
| BlockStack | 17% | 25% | 32% | 22% | 24% |
| Diaspora | 47% | 13% | 17% | 10% | 22% |
| Elastos | 13% | 28% | 42% | 24% | 27% |

#### 3.1.4.3 Consistência

Tabela 3.6: Resultados *AHP*

| - | Autenticação | Autorização | Armazenamento | Valor Final |
|---|---|---|---|---|
| Solid | 42% | 29% | 28% | 33% |
| BlockStack | 25% | 24% | 24% | 24% |
| Diaspora | 11% | 20% | 22% | 18% |
| Elastos | 22% | 27% | 27% | 25% |

Tendo em conta os estudos apresentados na tabela 3.6, o caminho a seguir será aprofundar a framework Solid de forma a torná-la mais escalável e preparada para a uma possível adoção massiva.

### 3.1.5 Definição do Conceito

O momento da definição do conceito corresponde à fase final da formalização da ideia já escolhida anteriormente. O conceito introduzido pelo Solid, em conjunto com o possível potencial de escalabilidade, permite tornar o conceito de "*Web* Descentralizada" numa realidade.

## 3.2 Proposta de Valor

Para estudar da proposta de valor serão utilizados o modelo *FAST* (*c.f. secção 3.2.1*) e o modelo de negócio Canvas (*c.f.* secção 3.2.2). Estes modelos tem como objetivo ajudar a desenhar o sistema tendo por base objetivos numa perspetiva de negócio e de sustentabilidade.

### 3.2.1 Modelo FAST

Com base na sua representação (*c.f. figura* 3.1) é possível perceber que o "core" de toda framework são os dados e que estes estão sob alçada do seu utilizador, podendo o mesmo a qualquer momento tomar ações como revogar acesso a determinadas aplicações (*relying party* [1]).

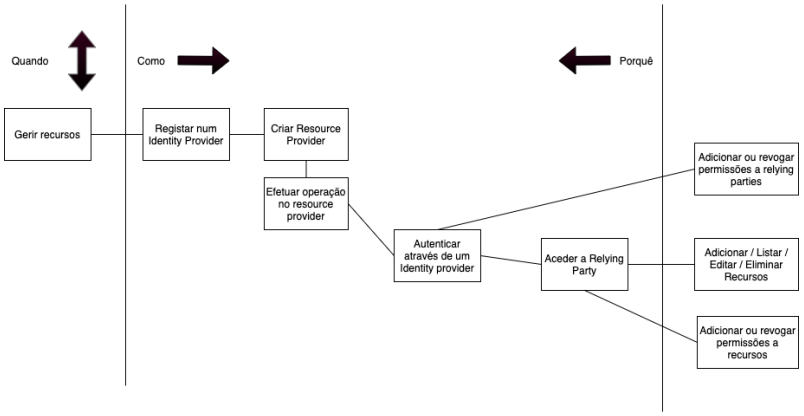


Figura 3.1: Modelo *FAST*

[1] *POD* ou aplicação cliente que utiliza um token providenciado por um servidor de identidade.

### 3.2.2 Modelo Canvas

O modelo canvas é uma ferramenta de gestão estratégica que permite o desenvolvimento do modelo de negócio para determinado projeto. A sua estrutura conta com nove blocos pré-definidos que permitem, de forma visual, criar ou adaptar um modelo já existente [62].

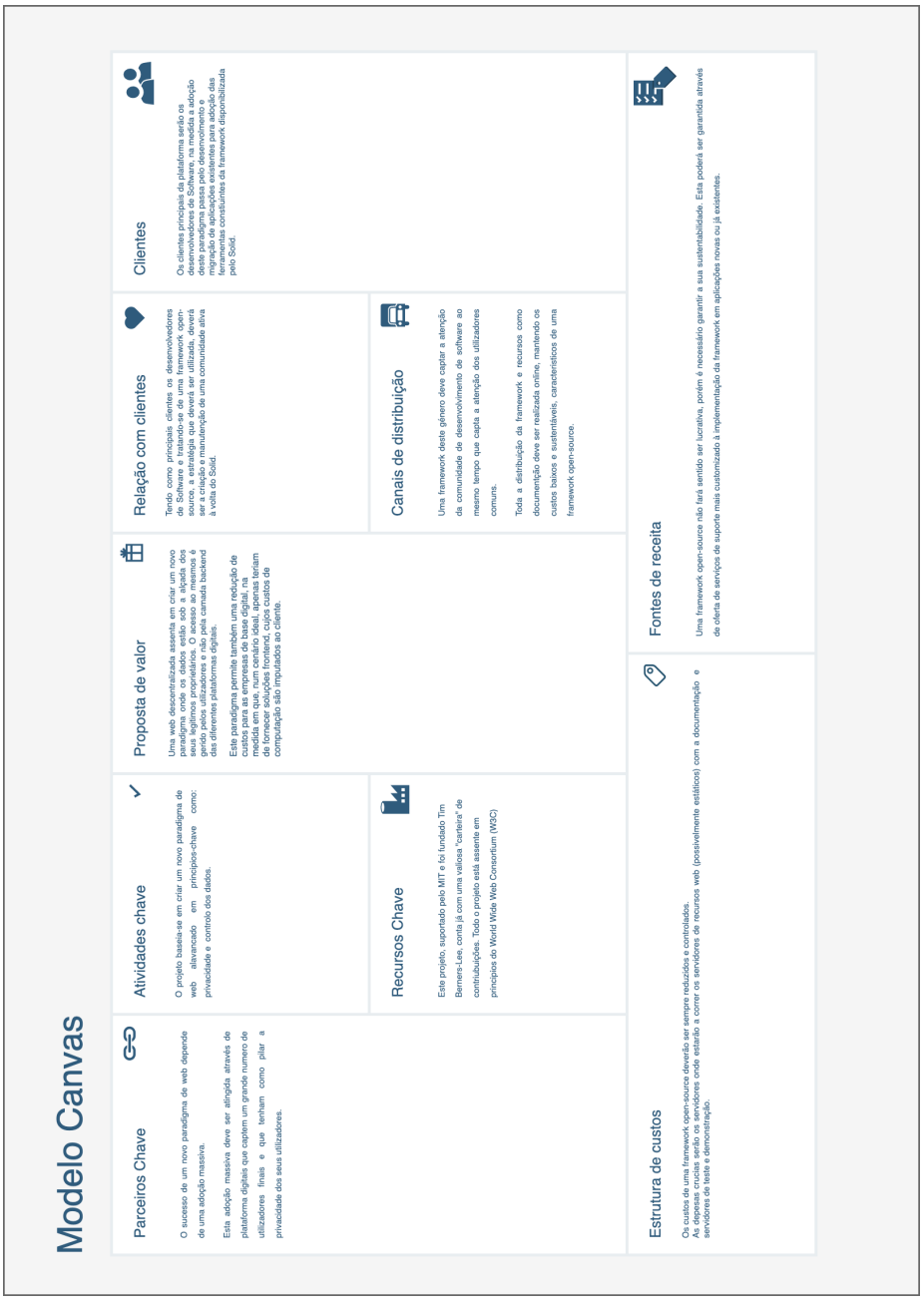


Figura 3.2: Modelo Canvas

## 3.3 Sumário

Tendo por base o modelo de análise *New Concept Development* e o processo de apoio à decisão *AHP*, são apresentados dados que permitem concluir que o *Solid* é o modelo descentralizado com maior potencial tendo em conta os parâmetros estabelecidos (*c.f.* secção 3.1).

A proposta de valor apresentada na secção 3.2 permite criar uma ideia geral da arquitetura e casos de uso que deverão ser desenhados no capítulo 4 de forma a melhorar o potencial da plataforma *Solid*.

# Capítulo 4

# Desenho

Este capítulo incide na apresentação da forma como a framework Solid está desenvolvida até ao momento e como será estruturada a nova arquitetura proposta.

O Solid é um projeto open-source, desenvolvido principalmente por uma comunidade criada e ativamente impulsionada por Tim Berners-Lee (*c.f.* 2.1.1). Esta comunidade dedica os seus esforços principalmente no desenvolvimento do *POD* como um todo, bem como documentação e ferramentas que permitem uma mais fácil integração da solução de novas aplicações.

O pilar do Solid é a descentralização, e por descentralização entende-se redefinição da propriedade da informação e por consequência alteração de todo o paradigma subjacente à camada *backend* dos sistemas actuais.

No atual paradigma mais frequentemente utilizado, a arquitetura cliente servidor (*c.f.* 2.3.1.2), a propriedade dos dados está condenada a perder-se na confiança para com a entidade de gere a camada de persistência de dados.

Numa perspetiva um pouco mais global, o que o Solid ambiciona substituir é a camada de persistência e possivelmente nas aplicações menos complexas a camada de backend também, entendendo-se por complexidade a possibilidade da lógica de negócio poder ser ou não assegurada em outro sistema.

Num cenário ideal, as aplicações cliente (*Relying Party App* conectam-se ao *POD* escolhido no momento de autenticação e, através de um ontologia própria ou uma já existente, são capazes proporcionar a mesma experiência existente nos dias de hoje ao utilizador.

Existem dois cenários possíveis de evolução do conceito Solid: (i) Manter a existência de múltiplas instâncias de Solid, e o utilizador pode escolher ao que pretende ligar-se; (ii) Existir apenas uma que pode ser entendida de forma semelhante a uma rede *P2P* (*c.f.* secção 2.3.1.3).

O cenário de múltiplas instâncias corresponde ao atual e permite que possam existir globalmente diversas instâncias de *POD*, podendo inclusive existirem instâncias mais restritas ou até mesmo instâncias privadas para apenas um utilizador (*c.f.* figura 4.1). Este modelo apresenta, contudo, alguns desafios de *continuous delivery*, na medida em que tem de ser criados mecanismos que permitam manter todas as instâncias de *POD* serem atualizadas.

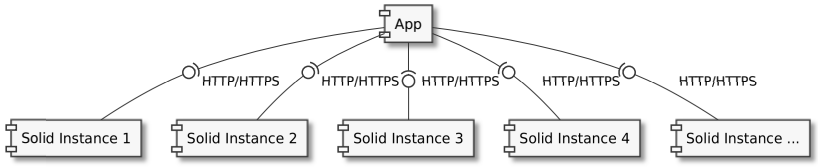


Figura 4.1: Diagrama de Componentes Solid - múltiplas instâncias

Por outro lado, e numa perspetiva um pouco mais disruptiva, O Solid poderia evoluir para uma solução única globalmente distribuída através de uma rede de "pequenas partes" do *POD* que iriam comunicar entre si. Este modelo parte do principio que o Solid segue uma arquitetura orientada a micro-serviços e que estes podem executados e ligados à rede em qualquer parte do globo, de forma a que iríamos assistir a uma única instância global de Solid, escalada por uma comunidade de cidadãos (*c.f.* figura 4.2). Este conceito tem sérios desafios como:

- *continuous delivery* - Seria também um desafio neste modelo, na medida em que seria necessário garantir que todas as instâncias dos diferentes micro-serviços estão devidamente atualizadas;
- Descoberta de novas instâncias - O Solid teria de implementar um mecanismo de registo de novas instâncias para que o tráfego pudesse ser encaminhado para novas instâncias.

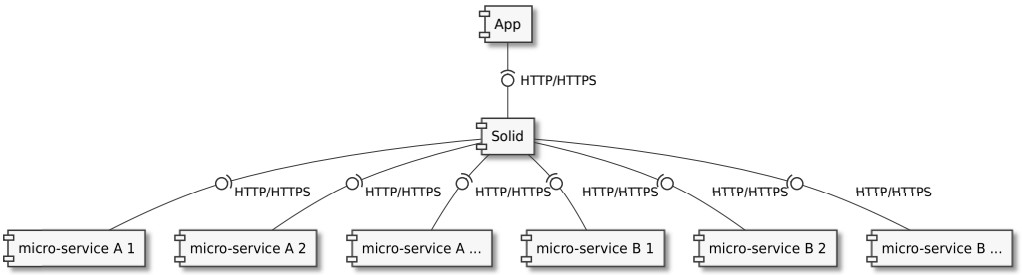


Figura 4.2: Diagrama de Componentes Solid - *P2P Network*

A visão do *Solid* tende a privilegiar o sentido de liberdade de escolha do utilizador, sendo possível este escolher em que instância do *Solid* confia ou até mesmo criar uma instância particular (c.f. secção 2.1.1. Assim, e de forma a cumprir a hipótese de manter a experiência para utilizador final (*c.f.* secção 1.3), foi optado por seguir o modelo atual do Solid, mas com foco em migrar o Solid para uma arquitetura orientada a micro-serviços, mantendo assim as portas abertas para no futuro poder ser pensada uma evolução do modelo para uma rede distribuída pela comunidade.

Assim, nas secções que se seguem apresentam-se as alterações estruturais propostas de forma a potenciar uma migração de uma arquitetura em monólito para uma arquitetura orientada a micro-serviços.

## 4.1 Arquitetura

Um dos problemas da arquitetura atual é a centralização de muitas camadas importantes no mesmo processo, afetando assim a sua escalabilidade, e pode consequentemente condenar o sucesso desta plataforma (*c.f.* secção 2.1.1).

Existe, desta forma, espaço para evoluir para uma arquitetura orientada a micro-serviços, potencialmente assente na divisão por capacidades de negócio, resumindo-se estas às seguintes:

- Gestão de contas - Camada responsável por garantir as operações de CRUD sobre o conceito de conta
- Autenticar utilizadores - Esta autenticação consiste sobretudo em garantir que a autenticidade do utilizador e conferir acesso da aplicação cliente ao seu WebID;
- Autorizar acesso de utilizador a determinado recurso - Deve ser possível consultar, alterar e criar novas regras de acesso a recursos num qualquer *POD* existente;
- Aceder e efetuar operações sobre recursos - O utilizador deve conseguir (se autenticado e autorizado) efectuar operações sobre recursos.

Desta forma, a arquitetura proposta deverá ter em consideração a separação da plataforma atual em quatro possíveis componentes (cada um deles responsável por uma das capacidades de negócio identificadas) [63]:

- *Solid-ID* - Camada responsável pela autenticação do utilizador, esta autenticação.
- *Permissions* - Camada responsável pela autorização.

- *Storage* - Camada responsável pelo acesso e manipulação de recursos.
- *Accounts* - Camada responsável pelas operações de *CRUD* sobre contas

Cada um dos diferentes componentes implementados deverá, numa perspetiva mais baixo nível, seguir uma estrutura com os seguintes módulos (*c.f.* figura 4.3):

- *Controllers* - Responsáveis por gerir a lógica relativa a casos de uso. Estes podem servir de resposta a pedidos tanto por interface *REST API* como por via de consumo de mensagens assíncronas provenientes de um *Message Broker*.
- *Services* - Camada responsável por gerir lógica de comunicação com sistemas externos (por exemplo publicação de eventos para *Message Broker* (*c.f.* secção 2.3.2.7).
- *Handlers* - Camada maioritariamente responsável por deter utilitários (p.e handler para erros durante execução).
- *Models* - Módulo responsável por gerir as estruturas de dados e lógica de negócio relativa a entidades relevantes do *POD*.

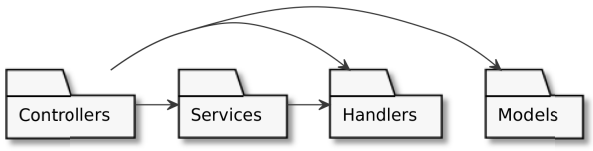


Figura 4.3: Diagrama de Módulos

Tendo por base os componentes apresentados, são descritas nas secções seguintes duas arquiteturas baseadas em pedidos *HTTP/HTTPS* com potencial para migração da plataforma Solid para micro-serviços: (i) Arquitetura não orientada a eventos (*c.f.* secção 4.1.1); (ii) Arquitetura orientada a eventos) (*c.f.* secção 4.1.2).

### 4.1.1 Arquitetura não orientada a eventos

Esta arquitetura tem subjacente a aplicação do padrão *API Gateway* como o ponto de entrada para todos o micro-serviços. Este padrão (*c.f.* secção 2.3.2.4) permite, desta forma, encapsular os diferentes serviços em pedidos *HTTP/HTTPS*, que reencaminham o pedido para o serviço mais indicado. Esta abstração oferece vantagens como escalabilidade, segurança e monitorização (*c.f.* secção 2.3.2.4)

É importante denotar que todos os componentes nesta arquitetura disponibilizam apenas uma interface de aplicação *REST* através do protocolo *HTTP*, que poderá ser seguro (*HTTPS*) quando a sua utilização encontra-se subjugada à utilização de um certificado válido emitido por uma entidade acreditada.

A figura 4.4 apresenta uma demonstração gráfica da interação entre os diferentes componentes nesta arquitetura.

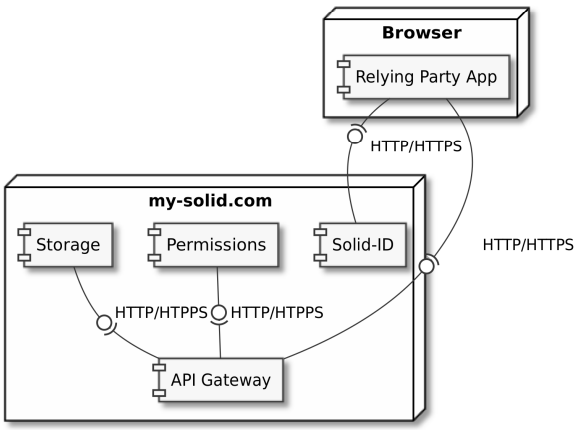


Figura 4.4: Diagrama de Componentes Arquitetura não orientada a eventos

### 4.1.2 Arquitetura orientada a eventos

Esta arquitetura mantêm o conceito de API Gateway, mas introduz a adoção de mecanismos de troca de mensagens assíncrona (*c.f.* secção 2.3.2.5) para a propagação de eventos lógicos para as diferentes aplicações que necessitem de os consumir (*c.f.* figura 4.5).

Na prática o que isto permite é, por exemplo, a propagação de um evento de criação de um novo utilizador. Aplicando-se, desta forma, o padrão CQRS (*c.f.* secção 2.3.2.6), contribuindo para um menor acoplamento entre as diferentes aplicações que lidam com escritas e com leituras de informação acerca dos utilizadores. Este padrão introduz indiretamente um outro: *Eventual Consistency*, este por sua vez consiste na assunção de que alterações a um determinado modelo eventualmente induzirão num estado consistente no sistema como um todo. De acordo com o teorema de CAP (c.f. *secção* 2.3.2.8), consistência é o parâmetro que devemos abdicar para conseguirmos escalabilidade e tolerância a falhas.

A utilização desta arquitetura permite também a adoção do padrão *Event Sourcing* (*c.f.* secção 2.3.2.9), este consiste na persistência dos eventos de alter-

ação de estado e permite que o estado do sistema seja obtido a qualquer momento pela "soma" das alterações de estado a que o mesmo foi sujeito.

Em termos de componentes, é possível perceber na figura 4.5 que o componente Permissions desaparece em detrimento do componente Accounts. O racional para esta decisão prende-se no facto de, na solução atual, o componente que controla as permissões baseia-se em ficheiros *ACL* geridos pela mesma camada responsável pela restante informação para determinada conta e, desta forma, o custo-benefício de criar um novo serviço que estaria, na prática, a replicar o serviço que gere informação da conta pode não ser de todo justificável.

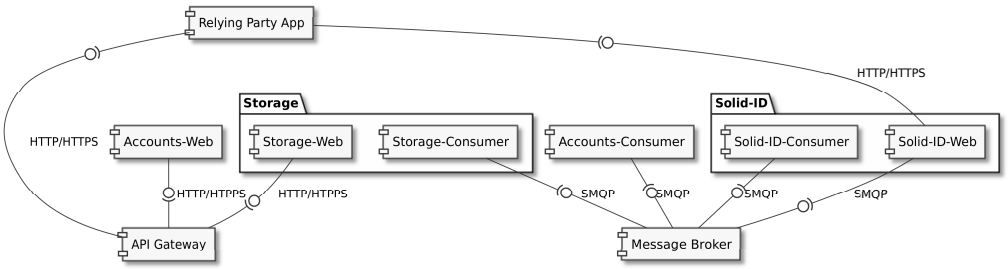


Figura 4.5: Diagrama de Componentes arquitetura orientada a eventos

### 4.1.3 Análise comparativa

Ambas as arquiteturas apresentadas compreendem uma migração para micro-serviços, sustentada numa separação em serviços baseada em capacidades de negócio.

A grande diferença entre as duas é a introdução da componente Message Broker na arquitetura baseada em pedidos HTTP e eventos, este factor adiciona complexidade, mas, por outro lado, garante um maior desacoplamento entre a componente LDP e a componente *ACL* e garante também uma diminuição da latência na obtenção de um recurso. O ponto negativo desta abordagem é a complexidade que adiciona ao desenvolvimento e ao processo de instalação de um *POD* como um todo, na medida em que a sua infraestrutura será mais robusta e complexa também (*c.f.* tabela 4.1).

Tabela 4.1: Análise comparativa arquiteturas propostas

| - | Arquitetura não orientada a eventos | Arquitetura orientada a eventos |
|---|---|---|
| Acoplamento | Elevado | Reduzido |
| Complexidade | Reduzida | Elevada |
| Latência | Elevada | Reduzida |

Assim, e tendo em conta uma decisão fundada em critérios como escalabilidade e manutenabilidade, será seguida a arquitetura orientada a eventos. Seguem-se alguns detalhes arquiteturais numa perspetiva de mais baixo nível.

## 4.2 Casos de uso

Nesta secção é apresentada uma vista arquitetural numa perspetiva de mais baixo nível, dando a entender de que forma os diferentes componentes interagem no contexto dos casos de uso.

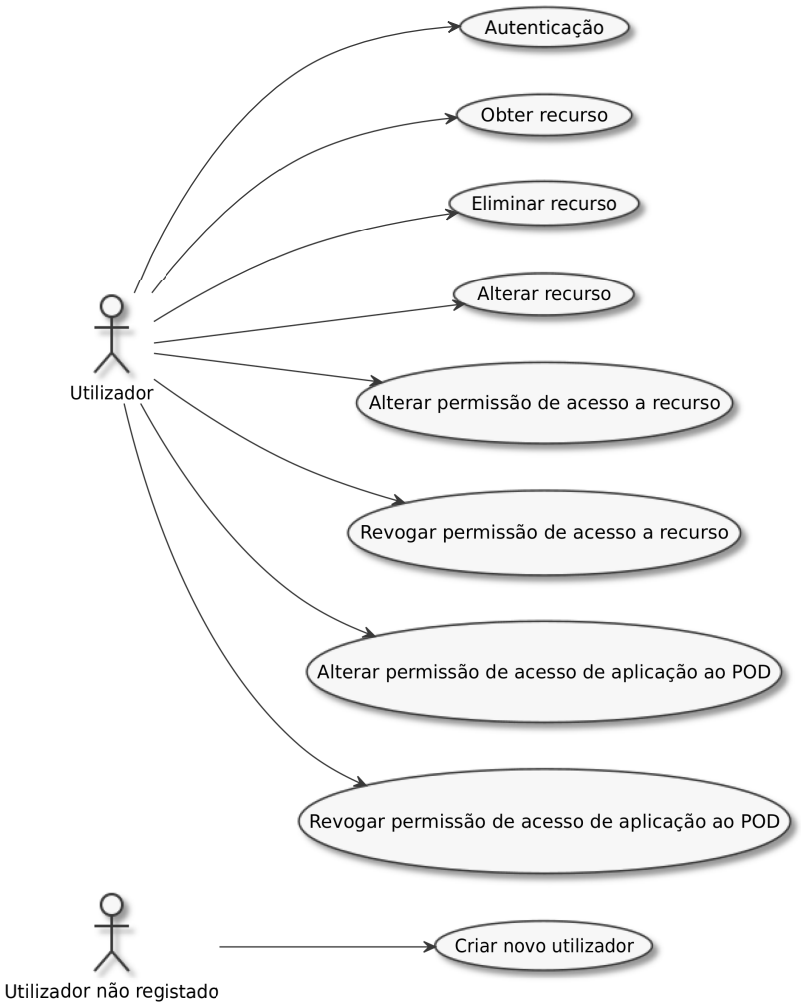


Figura 4.6: Diagrama de casos de uso *Solid*

De forma a evitar redundância de informação, esta secção, de todos os casos de uso (*c.f.* figura 4.6), terá foco naqueles que são arquiteturalmente mais relevantes.

### 4.2.1 Criar Novo Utilizador

O caso de uso de criar um novo utilizador representa uma funcionalidade arquiteturalmente relevante na medida em que implica interação entre todos os serviços constituintes do sistema Solid.

De forma a tornar mais clara a interação entre os diferentes componentes neste caso de uso, e pelo facto do diagrama de sequência ser extenso, o caso de uso é apresentado divido em nas três subsecções com os respetivos diagramas de sequência:

#### 4.2.1.1 Registo de conta

O ponto de entrada para este caso de uso é a *API Gateway* que reencaminha o pedido para o *Accounts Web* através do protocolo *HTTP/HTTPS*. Este, por sua vez é responsável por validar os campos preenchidos pelo utilizador segundo a lógica de negócio e no caso da informação ser válida propagá-la para *Message Broker* através do protocolo *AMQP* (*c.f.* figura 4.7).

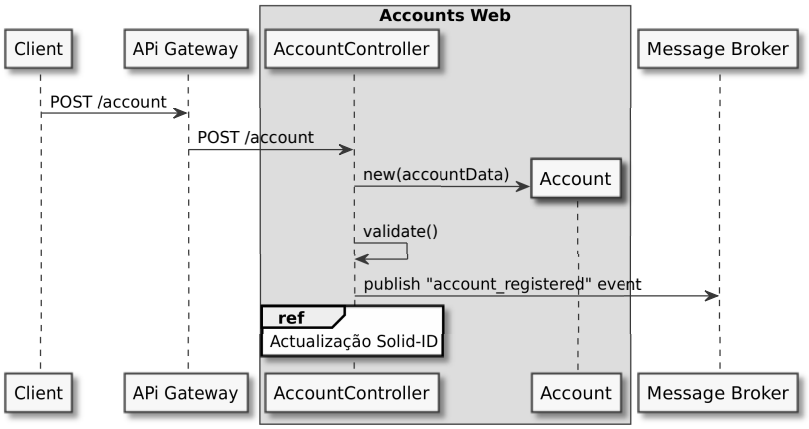


Figura 4.7: Diagrama de Sequência criação novo utilizador - registo de conta

#### 4.2.1.2 Actualização Solid-ID

Assim que o evento de criação de uma nova conta chega ao *Message Broker*, este encaminha-o para as *queues* que estiverem configuradas para a sua *routing key*. A partir desse momento, o *Solid-ID Consumer* e qualquer outro consumidor destas *queues* irá processar assincronamente os eventos.

O *Solid-ID Consumer* é responsável por gerir a conta e todo o processo de autenticação, necessitando, desta forma, de processar este evento para que possa persistir a nova conta e propagar para os outros sistema que o conseguiu fazer com sucesso (*c.f.* figura 4.8).

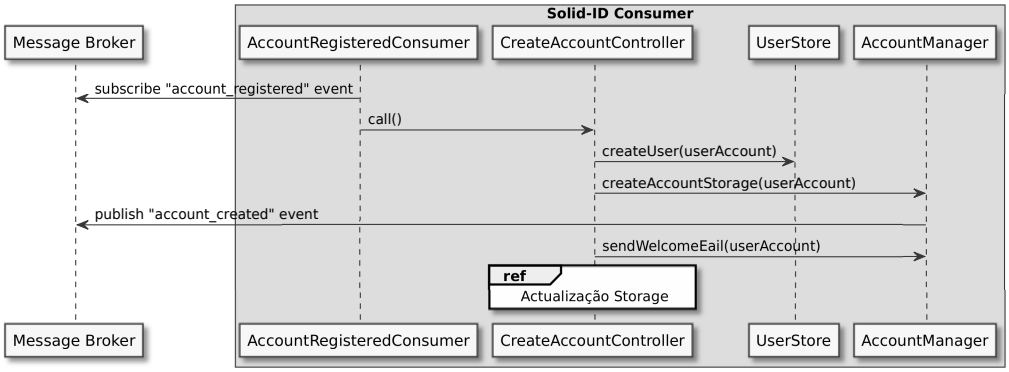


Figura 4.8: Diagrama de Sequência criação novo utilizador - atualizar *Solid-ID*

#### 4.2.1.3 Actualização Storage

O *Storage Consumer* irá, por sua vez, processar o evento de utilizador registado com sucesso para providenciar a estrutura de ficheiros template para todas as novas contas e terminar o processo de criação de uma nova conta num *POD* do *Solid* (*c.f.* figura 4.9).

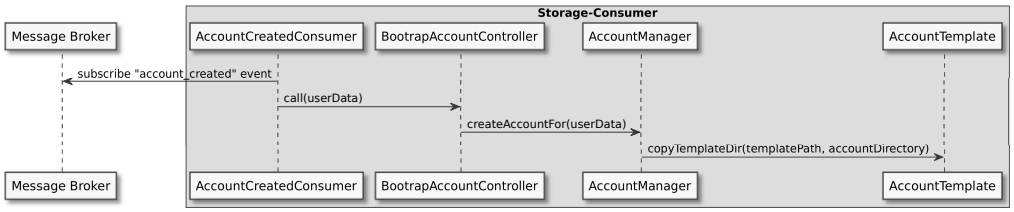


Figura 4.9: Diagrama de Sequência criação novo utilizador - atualização *Storage*

### 4.2.2 Autenticação utilizador

Nesta secção é apresentado o processo de autenticação através de uma *Relying Party App*, este cenário é um dos pilares do funcionamento do *Solid*, na medida em que o *POD* terá de substituir toda a camada *backend* das aplicações cliente.

Este caso de uso será também subdividido em sub-partes de forma a apresentar diagramas de sequência menos complexos e mais claros.

#### 4.2.2.1 Início autenticação

O cenário de autenticação deverá começar com a interação numa *Relying Party App*, esta por sua vez irá encaminhar para um formulário de login servido pelo *Solid-ID Web*.

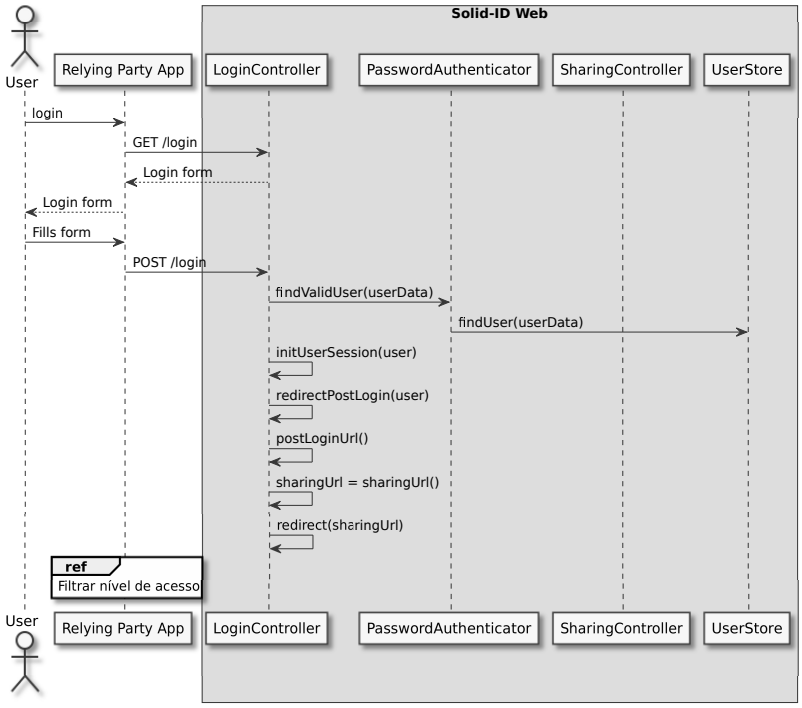


Figura 4.10: Diagrama de Sequência autenticação - Início

Os dados preenchidos no formulário serão de seguida validados, de forma a corresponder os as credenciais com as de algum utilizador registado naquele determinado *POD* (*c.f.* figura 4.10).

#### 4.2.2.2 Filtrar nível de acesso

O passo seguinte corresponde à escolha do nível de acesso da aplicação cliente aos dados no *POD*, esta escolha poderá ser editada pelo utilizador a qualquer altura (*c.f.* figura 4.11).

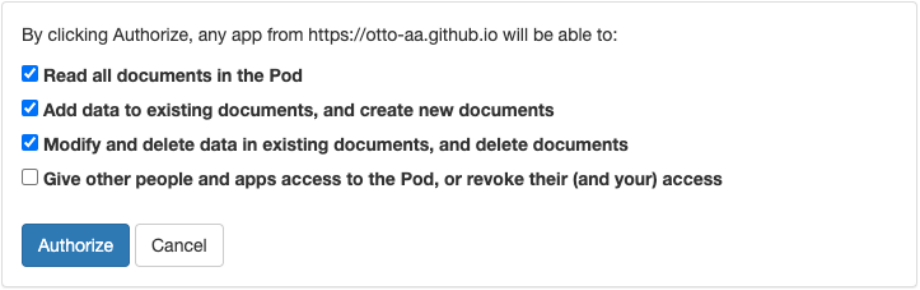


Figura 4.11: Ecrã correspondente a autorização de acesso

Este passo deverá ser necessário realizado durante a primeira autenticação (*c.f.* figura 4.12).

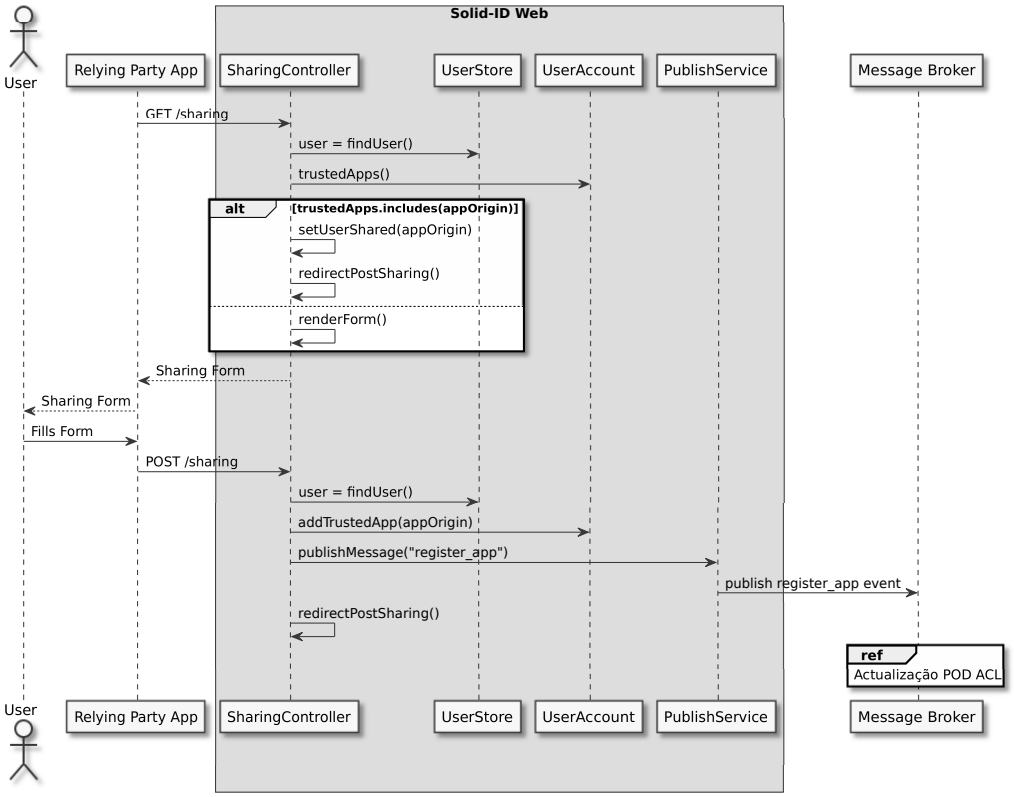


Figura 4.12: Diagrama de Sequência autenticação - Filtrar nível de acesso

#### 4.2.2.3 Atualização POD ACL

No seguimento da escolha do nível de acesso de determinada aplicação, é necessário que essa informação seja adicionada à respetiva *ACL*.

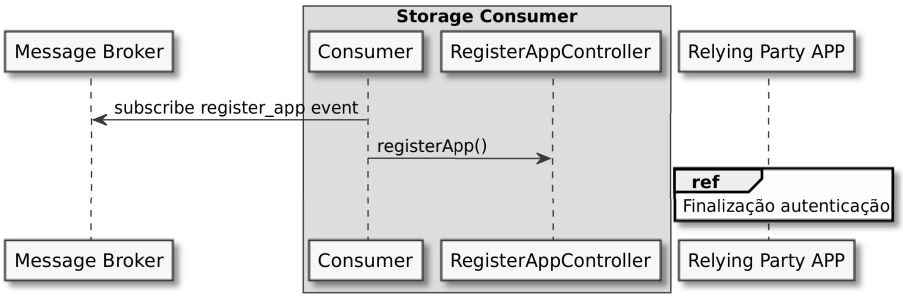


Figura 4.13: Diagrama de Sequência autenticação - Actualização *POD ACL*

Para que seja possível esta actualização da *ACL*, o *Storage Consumer* está a escutar uma *queue* de eventos de registo de origens dadas como seguras e que irá adicionar aos ficheiros *ACL* do respetivo utilizador (*c.f.* figura 4.13).

#### 4.2.2.4 Finalização autenticação

O último passo do processo de autenticação corresponde a conceder o acesso propriamente dito da aplicação *Relying Party App* ao *POD* utilizador.

Este acesso é conferido através da emissão de um *token* no formato *JWT* (*c.f.* secção 2.3.2.1) devidamente assinado (*c.f.* figura 4.14).

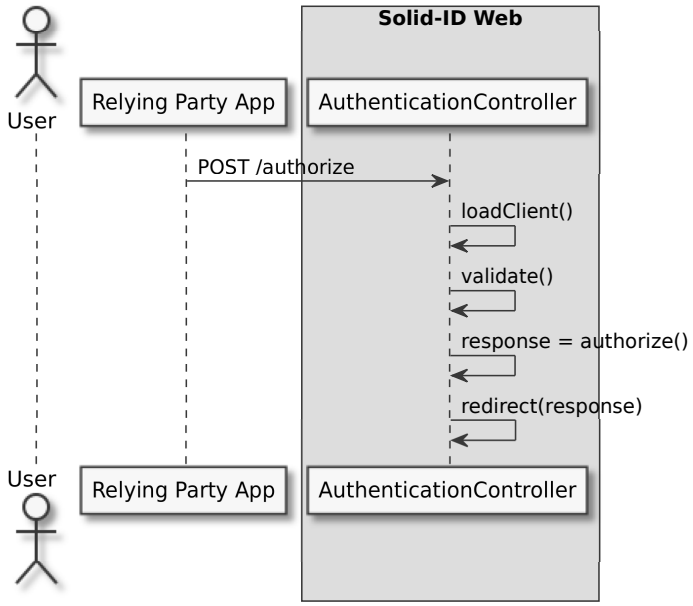


Figura 4.14: Diagrama de Sequência autenticação de utilizador - Finalização autenticação

### 4.2.3 Obter recurso

Este caso de uso implica aceder à unidade de armazenamento do utilizador na instância de *Solid*, o *POD* (*c.f.* secção 2.1.1.3).

Assim, este caso de uso inicia-se com a interação entre uma *Relying Party App* e o *Storage Web*, esta interação acontece sob o protocolo *http/https* e deve ser autenticada através do token de acesso fornecido do passo de autorização. Como o token de acesso foi gerado no *Solid-ID*, o componente *Storage* necessita de garantir que aquele token é válido e também se pode conceder acesso ao recurso pedido.

O primeiro passo é necessário efetuar a validação do token de acesso, para tal deverá ser validada a assinatura, a data de expiração e os *scopes* (*c.f.* figura 4.15)

Após a validação do token de acesso, é necessário validar se o *webID* obtido através do token tem de facto acesso ao recurso que pretende. Esta validação é realizada com recurso a uma biblioteca desenvolvida pela comunidade do Solid - *ACL-Checker* - que recursivamente valida os ficheiros *ACL* com hierarquia sobre o recurso pretendido e analisa se o webID tem acesso e que tipo de acesso tem (*c.f.* figura 4.15).

A partir do momento em que os dois *middleware* (validação token e validação *ACL*) permitem avançar, é possível seguir para a lógica de leitura / escrita do recurso.

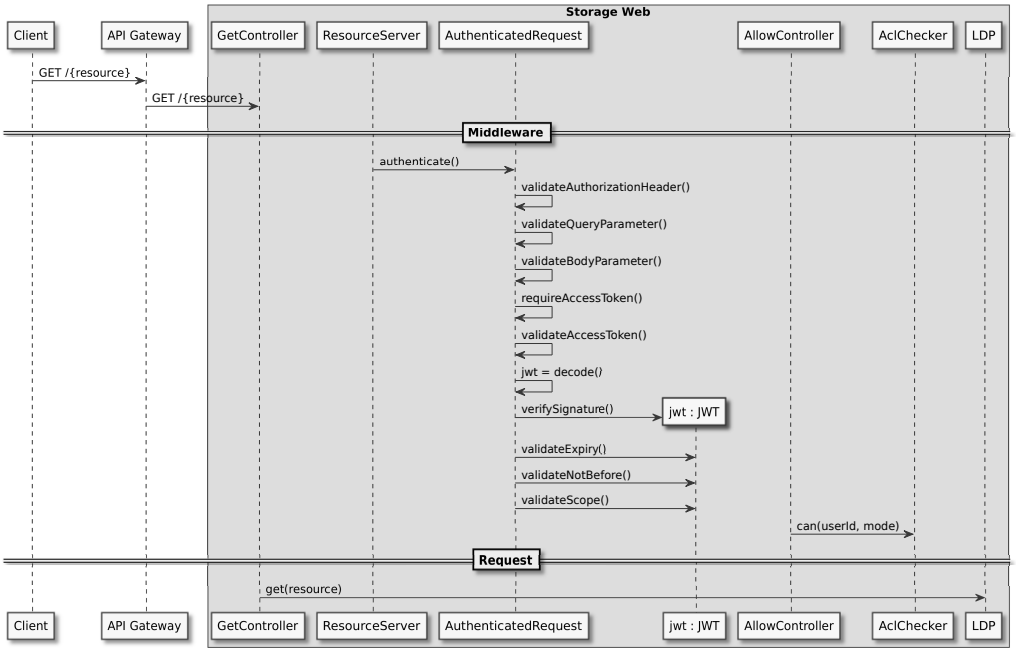


Figura 4.15: Diagrama de Sequência obter recurso

## 4.3 Sumário

O capítulo apresenta o desenho da solução da solução contextualizando com as tecnologias estudadas no capítulo 2. Este desenho é apresentado segundo duas perspetivas arquiteturais: Uma mais alto nível (*c.f.* secção 4.1) e uma outra mais baixo nível (*c.f.* secção 4.2).

A perspetiva mais alto nível, apresenta alternativas de decomposição do monólito em micro-serviços. Para cada uma das alternativas, é, também, explicada a

disposição dos diferentes componentes que surgem como potenciais constituintes da plataforma *Solid*, bem como as responsabilidades inerentes a cada um destes componentes.

Por outro lado, numa perspetiva mais baixo nível, o objetivo é demonstrar de forma mais detalhada e granular as interações entre os diferentes componentes no contexto de casos de uso.

Os diagramas e notas arquiteturais resultantes deste capítulo são o ponto de partida para a implementação (*c.f.* capítulo 4) e consequente avaliação da solução desenvolvida (*c.f.* capítulo 6).

# Capítulo 5

# Implementação

O projeto que serviu de base a esta implementação encontra-se em num repositório público[1], este representa todo o *POD* desenvolvido em Node.JS sob uma arquitetura em monólito.

A implementação necessária passa por reestruturar esta aplicação em 5 microserviços, seguindo o diagrama de componentes da arquitetura proposta apresentado na figura 5.1 (*c.f.* secção 4.1.3):

- *Accounts Web*;
- *Solid-ID Web*;
- *Solid-ID Consumer*;
- *Storage Web*;
- *Storage Consumer*.

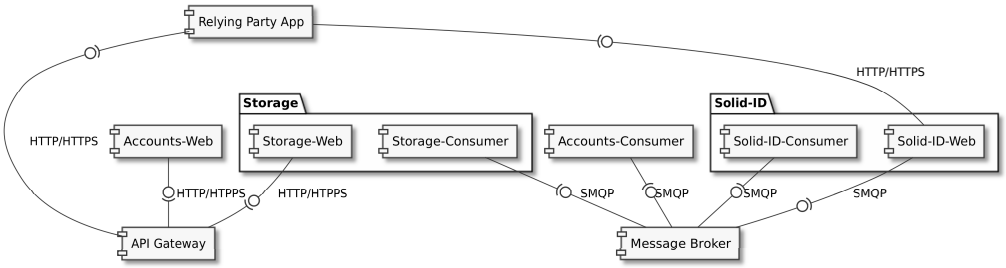


Figura 5.1: Diagrama de Componentes arquitetura proposta

[1] https://github.com/solid/node-solid-server

As nomenclaturas dos projetos seguem uma estrutura semelhante (*c.f.* excerto de código 5.1) para facilitar a perceção da responsabilidade e do tipo de interação disponível em cada projeto. Sendo o primeiro nome relativo à responsabilidade lógica e o segundo nome, *Web* ou *Consumer* conforme a interface de aplicação que disponibiliza.

```
1 (.*?)\b[^] (Web|Consumer)
```

Excerto de código 5.1: Expressão regular

De forma a manter a consistência os cinco serviços foram desenvolvidos na mesma linguagem (*Node.JS* e seguem, numa perspetiva mais baixo nível, uma arquitetura relativamente semelhante à documentada na secção 4.1, tendo esta por base o diagrama de módulos exposto na figura 5.

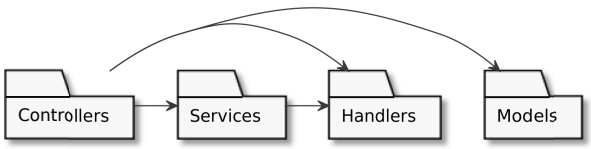


Figura 5.2: Diagrama de Módulos

O sistema foi construído com recurso à tecnologia de *containers*, neste caso adjacente às ferramentas *docker* e *docker-compose*, permitindo que o processo de implantação seja simples e sem necessitar de configuração de máquinas. Cada projeto tem o seu *dockerfile* (c.f. *excerto de código 5.3*) e na raiz do projeto principal e existe um ficheiro *docker-compose.yml* (*c.f.* excerto de código 5.4) que permite instanciar um sistema *POD* executando apenas a instrução apresentada no excerto de código 5.2.

```
1 $ docker-compose build && docker-compose up
```

Excerto de código 5.2: Instrução para instanciar *Solid*

```
1 FROM node:8.11.2-onbuild
2 COPY . .
3 COPY config.json-default config.json
4 RUN npm install
5 RUN chmod +x wait-for-it.sh
```

Excerto de código 5.3: Ficheiro Dockerfile

```yaml
version: '2'
services:
  version: "3.7"

services:
  solid-id-web:
    build:
      context: ./solid-id
      dockerfile: Dockerfile
    command:
      ["./wait-for-it.sh", "rabbitmq:5672", "--", "node", "bin/solid",
    "start"]
    environment:
      AMQP_URL: amqp://guest:guest@rabbitmq:5672
      SOLID_SSL_KEY: ../privkey.pem
      SOLID_SSL_CERT: ../fullchain.pem
      SOLID_SERVER_URI: https://solid-id.com:8443
      SOLID_PORT: 8443
      SOLID_GATEWAY_URI: https://solid.com:8442
    volumes:
      - ./store:/usr/src/app/.db
    depends_on:
      - rabbitmq

  rabbitmq:
    image: rabbitmq:3
```

Excerto de código 5.4: Ficheiro *docker-compose.yml*

No Excerto de código 5.4 é possível ver que o ficheiro *docker-compose.yml* está organizado pelos diferentes serviços constituintes da solução, e para cada um destes são definidas as dependências de outros projetos e, caso existam, variáveis de ambiente.

Com o propósito de mostrar pormenores relativos a cada sistema, estes serão, de seguida, abordados em subsecções.

## 5.1 Accounts Web

Este projeto apenas dispõe uma interface de aplicação *REST*. É responsável por receber o pedido de criação de conta e propagar o mesmo para os restantes sistemas. (*c.f.* 4.2.1).

Conforme é possível perceber no excerto de código 5.5, após a validação dos dados de entrada, a mensagem é publicada para o *RabbitMQ* no *controller* responsável pelo caso de uso de registo de conta - *AccountController*.

```
1  if(!validator.validate(req.body.email)){ throw (Object.assign(new Error
       ('invalid email'), { status: 400 }))}
2      let account = new Account(req.body.username, req.body.name, req.
       body.email, req.body.password);
3      publisher.publishMessage(
4        "accounts-api.account.registered",
5        JSON.stringify({
6          event: "account_registered",
7          object: { account },
8        })
9      );
10 }
```

Excerto de código 5.5: *AccountController* responsável por validar e propagar informação do utilizador

A emissão de eventos é da responsabilidade do ficheiro *Publisher* presente na camada *Services* do projeto, este recorre à biblioteca *amqplib* obtida através do *NPM* (*c.f.* secção 2.3.4.4).

```
1
2  require("dotenv").config();
3  'use strict'
4
5  const EXCHANGE = 'solid.events'
6  const createChannel = async () => {
7    const channel = async () => {
8      const conn = await require('amqplib').connect(
9        process.env.AMQP_URL
10     )
11     return conn.createChannel().then(channel => {
12       return Promise.resolve(channel)
13     })
14   }
15   let rabbitMQChannel = await channel()
16   rabbitMQChannel.assertExchange(EXCHANGE, 'topic')
17   global.globalRabbitMQChannel = rabbitMQChannel}
18 const publishMessage = async (routingKey, message) => {
19   globalRabbitMQChannel.publish(EXCHANGE, routingKey, Buffer.from(
       message))}
20 module.exports = { createChannel, publishMessage }
```

Excerto de código 5.6: *Publisher* responsável por emitir os eventos para o *RabbitMQ*

Conforme é possível perceber no excerto de código 5.6, o ficheiro *Publisher* está a criar uma ligação a um *exchange* de uma instância de *RabbitMQ*, para onde publica mensagens com um determinado tópico.

## 5.2 Solid-ID Web

Este sistema é o único, para além da *API-Gateway*, exposto ao mundo por interface *REST* (*c.f.* 4.1.2), esta decisão arquitetural implica que tenha de conhecer o endereço da *API-Gateway*, de forma a que possa indicar à *Relying Party App* onde deve aceder para obter recursos após concluir o processo de autenticação.

As rotas e os *Middleware* de autenticação (*c.f.* secção 4.2.2) são inicializados no arranque da aplicação e está implementado de forma a suportar futuros diferentes mecanismos de autenticação diferentes do atual *Open-ID Connect* (*c.f.* excerto de código 5.7).

```
1 function initAuthentication (app, argv) {
2   const auth = argv.forceUser ? 'forceUser' : argv.auth
3   if (!(auth in API.authn)) {
4     throw new Error(`Unsupported authentication scheme: ${auth}`)
5   }
6   API.authn[auth].initialize(app, argv)
7 }
```

Excerto de código 5.7: Inicialização da autenticação no arranque

O fluxo de autenticação *Open-ID Connect* é garantido neste sistema através de duas bibliotecas desenvolvidas pela comunidade Solid:

- *oidc-auth-manager* - Responsável por gerir todo o fluxo, esta biblioteca é agnóstica e pode ser utilizada tanto no *Identity Provider* como no *Resource Server* (*c.f.* excerto de código 5.8).
- oidc-op - Esta biblioteca representa o *Identity Provider* e todas as entidades lógicas associadas ao mesmo (*c.f.* excerto de código 5.9).

Assim, a alteração relativa a incluir os dados sobre *API Gateway* foram realizadas nestas bibliotecas externas.

```
1
2 this.gatewayUri = options.gatewayUri
3 // ...
4 gatewayUri: config.gatewayUri,
5 //  ...
6 providerConfig.gateway = this.gatewayUri
```

Excerto de código 5.8: Alteração ao *OIDCManager* na biblioteca nativa do *Solid oidc-auth-manager*

```
const schema = new JSONSchema({
  type: "object",
  properties: {
    \\ ...
    gateway: {type: "string", format: "uri"},
    // ...
  }
});
```

Excerto de código 5.9: Alteração ao *ProviderSchema* na biblioteca *oidc-op*

Este sistema expõe uma interface *REST* através da biblioteca externa *express* e um emissor de eventos igual ao do serviço *Accounts Web* recorrendo à biblioteca externa *amqplib*.

## 5.3 Solid-ID Consumer

O código desta aplicação partilha o mesmo projeto que o *Solid-ID Web*, visto terem em comum grande parte das dependências e apesar de terem interfaces diferentes, os controladores de casos de uso são os mesmo.

Este sistema é responsável por disponibilizar um consumidor para eventos lógicos de criação de novos utilizadores, que deverá posteriormente utilizar para criar os dados necessários para aplicar o fluxo de autenticação (*c.f.* excerto de código 5.10).

```
function start (argv, accountManager, userStore) {
  amqp.connect(argv.amqpUrl, function (error0, connection) {
    //...
    connection.createChannel(function (error1, channel) {
      channel.assertExchange('solid.events', 'topic', {durable: true})
      channel.assertQueue(
        'solid-id-consumer.queue',
        {   exclusive: false    },
        function (error2, q) {
          channel.bindQueue(q.queue, 'solid.events', '*.account.*')
          channel.consume q.queue,
            function (msg) {
              const body = JSON.parse(msg.content.toString())
              if (body.event === 'account_registered') {
                new CreateAccountController(accountManager, userStore,
   body.object.account).call()
              }}, {   noAck: true })
  //...
}
```

Excerto de código 5.10: *Consumer* responsável pela ligação ao *RabbitMQ*

Este *consumer* está a criar uma fila que irá escutar todos os eventos com a *routing key "*.account.*"* e está a aplicar lógica de negócio ao evento com o tipo *"account_registered"* que deverá ser emitido pelo component *Accounts Web* após receber um pedido *REST* de criação de um novo utilizador (*c.f.* 4.2.1).

```
async call () {
    try {
      this.validate()
      await this.createAccount()
    } catch (error) {
        debug('Error on creating account')
    }
  }

  async createAccount () {
    const userAccount = this.userAccount
    const accountManager = this.accountManager

    this.cancelIfUsernameInvalid(userAccount)
    this.cancelIfBlacklistedUsername(userAccount)

    if (!await this.userStore.findUser(userAccount.id)) {
      await this.createAccountStorage(userAccount)
      await this.saveCredentialsFor(userAccount)
    }

      if (userAccount && userAccount.email) {
        accountManager.sendWelcomeEmail(userAccount)
      }
    return userAccount
  }

  createAccountStorage (userAccount) {
    return this.accountManager.createAccountFor(userAccount)
  }

  saveCredentialsFor (userAccount) {
    return this.userStore
      .createUser(userAccount, this.password)
      .then(() => {
        debug('User credentials stored')
        return userAccount
      })
  }
```

Excerto de código 5.11: código CreateAccountController

A lógica relativa a criar o utilizador neste sistema é posteriormente tratada no *CreateController* (*c.f.* secção 4.2.1).

## 5.4 Storage Web

Este sistema é responsável por garantir o acesso autenticado e autorizado aos recursos do utilizador, sendo esta uma peça fulcral no *puzzle* do *POD*. A funcionalidade é relativamente simples, tratando-se na prática apenas de um repositório de ficheiro, mas as particularidades relativas ao processo de autenticação e autorização de pedidos conferem-lhe uma grande robustez.

Em termos de autenticação, segundo o fluxo *Open-ID Connect WebID* (*c.f.* figura 2.1), O componente *Storage Web* representa o *Resource Server*, tendo responsabilidade de garantir uma barreira de validação ao token de acesso passado no pedido *HTTP/HTTPS*.

A implementação do processo de autenticação dos pedidos neste componente, tratando-se de algo transversal, é alcançada com recurso à utilização de um *Middleware* (funcionalidade exposta pela biblioteca externa *Express*). As alterações necessárias a efectuar a este middlewar passam por garantir que irá, para cada pedido, validar a autenticidade do *token de acesso* tal como explicado na secção 4.2.2. Assim como o projeto *Solid-ID web*, este recorre às seguintes bibliotecas desenvolvidas pela comunidade Solid (*c.f.* excerto de código 5.12):

- *oidc-auth-manager* - Gere todo o fluxo de autenticação sob o protocolo *Open-ID Connect WebID*;
- *oidc-rs* - Implementa a lógica relativa à camada *Resource Server*.

```
1   const oidc = OidcManager.resourceServerOnly(argv)
2   oidc.initRs()
3   app.use('/', async (req, res, next) => {
4     oidc.rs.authenticate()(req, res, (err) => {
5       if (err) {
6         req.authError = err
7         res.status(200)
8       }
9       next()
10    })
11  })
12
13  // ...
14
15  }
```

Excerto de código 5.12: Inicialização do *Middleware* de autenticação de pedidos *REST*

O *Middleware* de validação do pedido, compreende duas importantes partes (*c.f.* excerto de código 5.13):

- Validação Pedido - Incide principalmente em garantir que o pedido está bem formado e tem todos os campos necessários
- Validação do *token* de acesso - Tratando-se de um *token* em formato *JWT*, esta parte incide em aplicar as validações padrão (*c.f.* figura 4.15) de forma a garantir a autenticidade do token, bem como extrair informação como o *webID* utilizador para que seja possível prosseguir para a fase de autorização

```
1  static authenticate (rs, req, res, next, options) {
2    let request = new AuthenticatedRequest(rs, req, res, next, options)
3    Promise.resolve(request)
4      .then(request.validateAuthorizationHeader)
5      .then(request.validateQueryParameter)
6      .then(request.validateBodyParameter)
7      .then(request.requireAccessToken)
8      .then(request.validateAccessToken)
9      .then(request.success)
10     .catch(error => request.error(error))
11 }
12
13 // ...
14
15 validateAccessToken (request) {
16   let {token, options} = request
17
18   if (options.optional && !token) {
19     return Promise.resolve(request)
20   }
21
22   return Promise.resolve(request)
23     .then(request.decode)
24     .then(request.validatePoPToken)
25     .then(request.allow)
26     .then(request.deny)
27     .then(request.resolveKeys)
28     .then(request.verifySignature)
29     .then(request.validateExpiry)
30     .then(request.validateNotBefore)
31     .then(request.validateScope)
32 }
```

Excerto de código 5.13: Camada de validação do pedido e do token de acesso na biblioteca *oidc-rs*

Após o pedido estar devidamente autenticado, é necessário garantir que o mesmo tem acesso ao recurso que está a tentar aceder, bem como autorização para a ação que pretende ver realizada sobre o mesmo.

Para tal é utilizado também um *Middleware* que será responsável por permitir que o pedido avance ou não para o acesso propriamente dito.

Esta funcionalidade é implementada recorrendo à biblioteca externa *acl-checker* desenvolvida pela comunidade Solid, e permite iterar sobre o sistema de ficheiros em busca de ficheiros *.acl* hierarquicamente relevantes ao arquivo em causa de forma a averiguar se o pedido está de facto autorizado (*c.f.* excerto de código 5.14).

```
1  function LdpMiddleware (corsSettings) {
2    const router = express.Router('/')
3
4    // Add Link headers
5    router.use(header.linksHandler)
6
7    if (corsSettings) {
8      router.use(corsSettings)
9    }
10
11   router.copy('/*', allow('Write'), copy)
12   router.get('/*', index, allow('Read'), header.addPermissions, get)
13   router.post('/*', allow('Append'), post)
14   router.patch('/*', allow('Append'), patch)
15   router.put('/*', allow('Write'), put)
16   router.delete('/*', allow('Write', true), del)
17
18   return router
19 }
```

Excerto de código 5.14: Alteração ao *OIDCManager* na biblioteca *oidc-auth-manager*

## 5.5 Storage Consumer

Esta aplicação tem como responsabilidade escutar eventos aplicacionais que possam ter impacto na unidade de persistência de dados, estes eventos podem ser (*c.f.* excerto de código 5.15):

- *account_created* - Este evento cria a necessidade de que seja criado no sistema de ficheiros um directório com base num template base de recursos para novos utilizadores.

- *register_app* - Indica ao sistema que deve adicionar à *ACL* base do sistema de ficheiros do utilizador um novo domínio autorizado.

```
amqp.connect(config.amqpUrl, function (
  error0,
  connection
) {
  if (error0) {
    throw error0
  }
  connection.createChannel(function (error1, channel) {
    if (error1) {
      throw error1
    }
    var exchange = 'solid.events'

    channel.assertExchange(exchange, 'topic', {
      durable: true,
    })
channel.assertQueue(
      'ldp-consumer.queue',
      {
        exclusive: false,
      },
      function (error2, q) {
        channel.bindQueue(q.queue, exchange, '*.account.*')

        channel.consume(
          q.queue,
          function (msg) {
            let body = JSON.parse(msg.content.toString())
            if (body.event === 'account_created') {
              handleBootstrapAccount(accountManager, body.object)
            }
            if (body.event === 'register_app') {
              registerApp(accountManager.store, body.object.appOrigin,
    body.object.accessModes, body.object.webID)
            }
          },
          {
            noAck: true,
          }
        )
      }
    )
```

Excerto de código 5.15: *Consumer* responsável por interpretar os eventos aplicacionais com impacto no sistema de ficheiros

## 5.6 Sumário

O presente capítulo apresenta os detalhes mais técnicos da implementação da solução desenhada no capítulo 4.

Numa primeira fase, são expostos pormenores sobre a instalação da plataforma através de *containers* e é, também, explicada a nomenclatura atribuída a cada micro-serviço.

Posteriormente, são apresentados detalhes técnicos relevantes da implementação contextualizados para cada um dos micro-serviços implementados.

A solução implementada deverá ser avaliada face aos objetivos inicialmente propostos, com base em hipóteses e metodologias previamente estabelecidas (*c.f.* capítulo 6), permitindo, desta forma, que sejam obtidas conclusões sobre limitações e trabalho futuro (*c.f.* capítulo 7).

# Capítulo 6

# Experimentação e Avaliação

## 6.1 Experiências e Testes

Neste contexto, uma experiência ou teste, assim como definido na secção 2.2.5, consiste na utilizando uma metodologia ou técnica específica de forma a avaliar uma determinada grandeza.

Assim, para a realização de um teste é preciso definir as grandezas, as hipóteses e as metodologias de avaliação.

### 6.1.1 Grandezas a Avaliar

Os objetivos definidos na secção 1.2 induzem uma contribuição para o projeto *Solid* com vista a incrementar o seu potencial de escalabilidade, e desta forma cumprir, também, as hipóteses formuladas na secção 1.3.

Tendo isto em consideração, existem duas potenciais grandezas a avaliar:

- Qualidade da implementação - A qualidade da implementação é crucial para tornar viável a aceitação da solução como uma contribuição para a comunidade em volta do Solid. A qualidade deverá ser mensurada tendo em conta métricas como resultados de software;
- Performance - Tendo em conta o mesmo escalamento, é relevante conseguir apurar se a solução desenvolvida consegue melhores resultados em termos de performance;
- Escalabilidade - Como complemento ponto referente à performance, é importante perceber se de facto a solução implementada é mais escalável horizontalmente que a plataforma atual.

### 6.1.2 Hipóteses

Uma hipótese consiste numa afirmação que se quer corroborar através de testes estatísticos utilizando grandezas identificadas.

No contexto deste projeto, foram definidas duas hipóteses em linha com aquelas que foram formuladas na secção 1.3:

- H1 - Hipótese referente a testes aplicacionais - Sucesso 100% dos testes de software e cobertura superior a 80%. Este valor vai em linha com o valor recomendado para criar um balanço entre cobertura de cenários de utilização e cobertura de código testado [64];

- H2 - Hipótese referente a testes de performance - Capacidade de cada micro-serviço deve ser de pelo menos 75% da suportada pela plataforma atual (monolítico) para uma determinada carga, mantendo o tempo máximo de resposta de cada pedido inferior a 400 milissegundos. O racional para a escolha do valor de 75% surge no sentido de que podemos assumir que a performance individual dos serviços pode ser um pouco inferior do que o monólito, sem que isso afete o sistema orientado a micro-serviços como um todo, na medida em que o fator de escalabilidade flexível irá garantir melhor performance que o monólito. Por outro lado, o valor de 400 milissegundos foi obtido a partir da exploração das médias de tempo resposta para um conjunto basto de aplicações web através da fonte [65].

### 6.1.3 Metodologias de Avaliação

As metodologias de avaliação consistem na forma como serão verificadas as hipóteses definidas. A metodologia a utilizar irá depender da hipótese e da grandeza em causa.

A hipótese referente a testes aplicacionais, tendo em conta a qualidade da solução implementada, deve recorrer a testes de software, nomeadamente testes unitários e testes de integração que são executados através de ferramentas específicas para este efeito. Tendo em conta que esta metodologia é directa, não deverá ser necessário tratamento estatístico [66].

A hipótese referente a testes de performance consiste em corroborar que cada micro-serviço do sistema, sob uma determinada carga, consegue suportar pelo menos 75% do *throughput* da plataforma atual (monólito), mantendo o tempo de resposta de cada pedido abaixo de 400 milissegundos.

Para esta hipótese deverão ser realizados testes de performance e capacidade aos diferentes micro-serviços e à plataforma atual.

Estes testes deverão ser executados recorrendo a ferramentas para este efeito (como por exemplo JMeter) e os resultados devem ser apontados de forma a poderem ser tiradas conclusões.

É importante referir que no caso dos micro-serviços, é expectável que os pedidos realizados durante os testes de carga passem pela *API-Gateway*, de forma a que o cenário se aproxime o mais possível da realidade.

## 6.2 Resultados

Definidas as hipóteses e os meios e as metodologias de avaliação, seguem-se os resultados para cada uma das hipóteses propostas.

### 6.2.1 Hipótese referente a testes aplicacionais

Esta hipótese indica que os testes de Software devem ter uma taxa de sucesso de 100%, bem como uma cobertura superior a 80%. No contexto da implementação foram utilizadas as bibliotecas *mocha* e *chai* para criar e mensurar os testes, tanto unitários como de integração.

Os números apresentados na tabela 6.2 correspondem ao conjunto de todos os projetos e bibliotecas externas alteradas no decorrer da implementação desta nova arquitetura para o Solid (*c.f.* secção 1.6). Numa perspetiva de estabelecer um ponto de comparação para perceber se existem diferenças substanciais nos números de testes e percentagem face à solução monolítica.

Tabela 6.1: Resultados dos testes aplicacionais *Solid* monólito

| Testes Unitários | Testes de Integração | Cobertura |
|---|---|---|
| 305 | 451 | 80% |

Tabela 6.2: Resultados dos testes aplicacionais *Solid* micro-seviços

| Testes Unitários | Testes de Integração | Cobertura |
|---|---|---|
| 382 | 466 | 82% |

Com base nos resultados apresentados na tabela 6.2 é possível inferir que tanto a taxa de sucesso como a taxa de cobertura permitem corroborar a hipótese estipulada.

Na mesma linha de raciocínio, quando comparados estes valores com os correspondentes à solução monolítica (*c.f.* tabela 6.1) é possível perceber que houve uma ligeira melhoria tanto em termos de cobertura como em número de testes realizados à plataforma.

### 6.2.2 Hipótese referente a testes de performance

A segunda hipótese incide na análise de performance dos micro-serviços desenvolvidos, com vista a inferir se houve de facto ganhos justificativos em relação à arquitetura atual.

#### Setup dos Testes

A premissa base foi de que tanto o monólito, como os micro-serviços seriam testados estando a ser executados apenas numa instância sem qualquer tipo de escalamento, isto porque o monólito não suporta escalamento horizontal e, desta forma os resultados obtidos poderiam não ser fiáveis.

Esta premissa vai de encontro à hipótese referente a performance do sistema desenvolvido, formulada na secção 1.3.

Para a realização dos testes foi utilizada a ferramenta de testes de carga *JMeter*, que permite simular utilizadores e pedidos simultâneos a um determinado serviço. Esta ferramenta disponibiliza a funcionalidade de executar contra interfaces de aplicação *REST* através da configuração dos seguintes parâmetros:

- Caminho - O caminho que permite chegar ao serviço;
- Método - Método HTTP do *endpoint* específico;
- Numero de *Threads* - Cada *thread* permite simular um utilizador a executar pedidos, sendo que múltiplas *threads* irão ser executadas em paralelo e, por consequência, simular múltiplos utilizadores a fazer pedidos em simultâneo;
- Período *Ramp-up* - Variável em segundos que indica quão gradual devem ser criadas novas *threads*. A divisão do numero de *threads* pelo valor configurado nesta variável indica o numero de novas *threads* que serão criadas por segundo até que seja atingido o numero total;
- Total repetições - Numero de pedidos que cada *thread* irá fazer. Assim que todas terminarem de executar o valor total de repetições, está concluído.

De forma a perceber o numero de utilizadores simultâneos que deveriam ser configurados, foram efectuados sucessivos testes de performance com incremento do número de *threads* e, assim, perceber qual o número máximo de utilizadores simultâneos que uma instância de *Solid* consegue servir mantendo um tempo de resposta inferior a 400milissegundos.

Utilizando as configurações da tabela 6.3, ao fim de 10 segundos teremos 10 utilizadores simultâneos que irão fazer um total de 200 pedidos cada um.

Tabela 6.3: Tabela configurações base JMeter

| Número de *threads* | Período *Ramp-up* | Número Repetições |
|---|---|---|
| 10 | 10 | 200 |

De forma a manter a equidade em termos de *hardware*, os diferentes serviços serão testados a executar em *Virtual Private Server (VPS)* adquiridos apenas para estas experiências, com as configurações base referidas na tabela 6.4.

Tabela 6.4: Tabela configurações base *VPS*

| CPU | RAM | Disco | Sistema Operativo |
|---|---|---|---|
| 2 cores | 4GB | 50GB SSD | CentOS 7 |

Estas experiências deverão incidir, por um lado, sobre casos de uso que permitam exercitar os diferentes micro-serviços na arquitetura proposta (c.f. secção 4.1), e, por outro lado, sobre casos de uso que representem funcionalidades cruciais para o *POD*. Assim, os testes de performance irão incidir sobre três casos de uso:

- Criação de uma nova conta
- Autenticação
- Obter recurso

No caso do monólito, o teste deverá incidir sobre a mesma instância aplicacional para os *endpoints* correspondentes aos três pedidos de forma simultânea, permitindo simular circunstâncias o mais reais possíveis.

Para a arquitetura orientada a micro-serviços, estes testes deverão fazer incidir os pedidos, de forma independente, nos serviços responsáveis pelos respetivos casos de uso:

- *Accounts Web* - Criação de uma nova conta;
- *Solid ID Web* - Autenticação;
- *Solid Storage Web* - Obter recurso.

#### Teste Monólito

Para o teste de performance ao monólito foi implantado o sistema *Solid* sob a arquitetura monolítica numa das *VPS* (*c.f.* figura 6.1). Estes testes incluíram,

assim como proposto na secção 6.2.2, pedidos a três diferentes *endpoints*, correspondentes a interações de registo de conta, autenticação e obtenção de recurso.


Figura 6.1: Implantação *solid server* monólito

Os dados relevantes para este cenário de teste são o tempo médio de resposta (em milissegundos) e o *throughput* (em pedidos/segundo), na medida em que estes são os dados que permitirão corroborar a hipótese formulada (*c.f. tabela 6.5*).

Tabela 6.5: Resultados teste performance *solid server* monólito

| - | Média tempo de resposta (ms) | *Throughput* (p/s) |
|---|---|---|
| Total | 380 | 20.6 |

### Teste arquitetura micro-serviços em apenas um nó

Para este cenário de teste a solução orientada a micro-serviços (*c.f.* secção 4.1) seguiu uma implantação em apenas um nó, como é possível observar na figura 6.2.2.

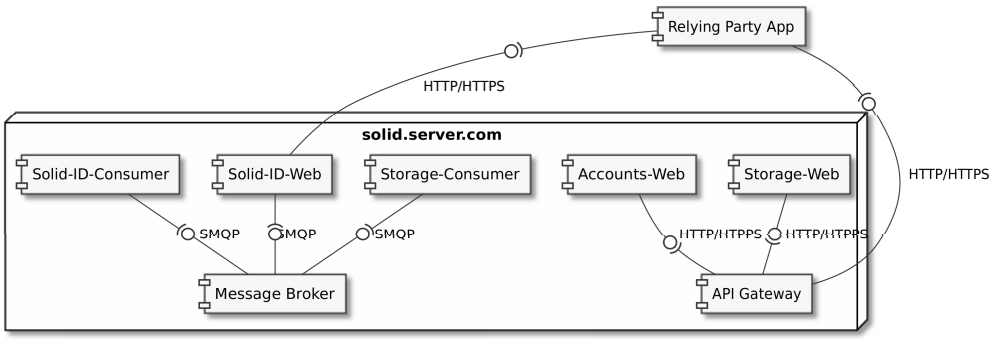


Figura 6.2: Implantação arquitetura orientada a micro-serviços em apenas um nó

Os resultados para este teste são apresentados na tabela 6.6, incluindo para cada um dos micro-serviços, os valores de tempo médio de resposta e *throughput*. Conforme é possível perceber, face aos resultados do teste ao monólito expostos na tabela 6.1, os valores demonstram perdas significativas em ambas as métricas,

colocando, assim, em risco a inferência da hipótese formulada no âmbito dos testes de performance (*c.f.* 6.1.2).

Tabela 6.6: Resultados teste de performance a arquitetura micro-serviços instalada em apenas um nó

| - | Tempo médio de resposta (ms) | *Throughput* (p/s) |
|---|---|---|
| *Solid ID Web* | 524 | 11.2 |
| *Solid Storage Web* | 600 | 12 |
| *Accounts Web* | 490 | 14.2 |

O facto de todos os micro-serviços, bem como o *Message Broker* e o componente com a função de *API Gateway*, terem sido instalados no mesmo nó (*c.f* figura 6.2.2), pode estar a criar alguma sobrecarga no nó e, consequentemente, resultar em perda de performance. Esta teoria, deve, por sua vez, ser também corroborada com base nos testes subsequentes.

### Teste arquitetura micro-serviços distribuídos

Tendo em conta o resultados dos testes documentados na secção 6.2.2 apresentarem perdas de performance significativas face aos valores obtidos para o monólito (*c.f.* tabela 6.5), segue-se um novo teste com uma abordagem de implantação diferente. Neste teste, o sistema *Solid* sob arquitetura orientada a micro-serviços foi instalado seguindo uma implantação distribuída em mais do que um nó (*c.f.* figura 6.3).

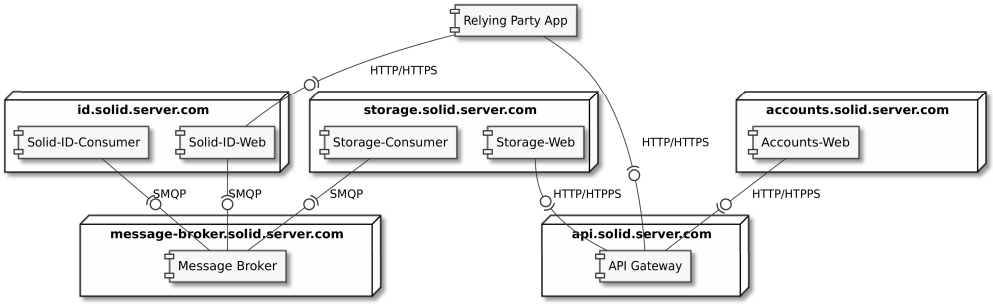


Figura 6.3: Implantação arquitetura orientada a micro-serviços distribuída

Os resultados para este cenário são apresentados sob a forma tabular para cada um dos micro-serviços em causa, seguindo assim o plano de *setup* dos testes (*c.f.* secção 6.2.2).

Com base nos valores apresentados na tabela 6.7 é possível corroborar a teoria de que o *hardware* terá sido o *bottleneck nos testes* apresentados na secção

Tabela 6.7: Resultados teste de performance a arquitetura micro-serviços distribuídos

| - | Média tempo de resposta (ms) | *Throughput* (p/s) |
|---|---|---|
| *Solid ID Web* | 390 | 16.2 |
| *Solid Storage Web* | 385 | 17.6 |
| *Accounts Web* | 320 | 19.6 |

6.2.2, estando assim mitigado esse problema. É possível também perceber que os resultados dos testes para este cenário, apesar de apresentarem perdas menos significativas tanto a nível de tempo médio de resposta como de *throughput*, revelam que podem, ainda, existir possíveis alterações à implantação com potencial de melhoria de performance.

#### Teste arquitetura micro-serviços distribuídos sem containers

Conforme explicado, apesar dos resultados dos testes documentados na secção 6.2.2 serem relativamente satisfatórios tendo em conta àquilo que foi estipulado pela hipótese formulada, segue-se um novo teste com uma abordagem de implantação sem recurso à tecnologia *containers*.

Neste teste, o sistema *Solid* sob arquitetura orientada a micro-serviços seguiu, tal como no teste 6.2.2, uma implantação distribuída em mais do que um nó (*c.f.* figura 6.3), mas neste cenário não foi utilizada qualquer tecnologia de *containers*, estando os serviços a ser executados diretamente sobre o sistema operativo da máquina.

Tabela 6.8: Resultados teste performance a arquitetura micro-serviços distribuídos sem *containers*

| - | Média tempo de resposta (ms) | *Throughput* (p/s) |
|---|---|---|
| *Solid ID Web* | 385 | 19.0 |
| *Solid Storage Web* | 370 | 20.2 |
| *Accounts Web* | 300 | 22.6 |

Os resultados apresentados na tabela 6.8 parecem corroborar a teoria de que a tecnologia *containers*, apesar das vantagens referidas na secção 2.3.4.5, cria alguma latência que terá tido algum impacto nos testes da secção 6.2.2. No mesmo contexto, estes valores estão, já, bastante próximos aos valores resultantes dos testes relativos ao *Solid* monolítico, apresentando até, em grande parte dos casos, algumas melhorias.

## 6.3 Sumário

Com base nos testes de performance realizados ao monólito (*c.f.* tabela 6.5), foi possível obter um *throughput* de 20.6 pedidos por segundo com uma média de tempo de resposta de 380ms, desta forma, segundo a hipótese formulada, o valor de *throughput* para os micro-serviços deverá ser no mínimo de 75% esse valor, ou seja, superior a 15.45 pedidos por segundo e mantendo um tempo médio de resposta inferior a 400ms.

A tabela 6.9 foi construída com o objetivo de, tendo por base testes realizados ao *Solid* monolítico (*c.f.* secção 6.2.2), salientar as evoluções percentuais dos valores de tempo médio de resposta e *Throughput* para os diferentes testes realizados à arquitetura orientada a micro-serviços:

- este arquitetura micro-serviços em apenas um nó (*c.f.* secção 6.2.2);
- Teste arquitetura micro-serviços distribuídos (*c.f.*6.2.2);
- Teste arquitetura micro-serviços distribuídos sem *containers* (*c.f.* secção 6.2.2).

Tabela 6.9: Tabela de resultados percentuais dos testes face ao teste realizado ao *Solid* monolítico

| | | Evolução Tempo médio de resposta (%) | Evolução *Throughput* (%) |
|---|---|---|---|
| Micro-serviços apenas um nó | *Solid ID Web* | -39 | -46 |
| | *Solid Storage Web* | -58 | -42 |
| | *Accounts Web* | -29 | -31 |
| Micro-serviços distribuídos | *Solid ID Web* | -3 | -21 |
| | *Solid Storage Web* | -1 | -15 |
| | *Accounts Web* | 16 | -5 |
| Micro-serviços distribuídos sem *containers* | *Solid ID Web* | 1 | -8 |
| | *Solid Storage Web* | -1 | -2 |
| | *Accounts Web* | 21 | 10 |

O primeiro teste aos micro-serviços seguiu uma implantação relativamente simples e com potencial de custos reduzidos, tendo sido toda a arquitetura implantada numa única máquina.

Porém, utilizando esta solução sem qualquer tipo de escalamento vertical ou horizontal, acaba por tornar o *hardware* o *bottleneck* da arquitetura, afetando tanto o *throughput* como o tempo médio de resposta de cada pedido processado.

Tendo em conta que o teste anterior não cumpre com as metas inferidas nesta hipótese, um novo teste foi desenhado recorrendo a uma implantação distribuída (*c.f.* figura 6.3), mantendo, desta forma, a premissa de não recorrer nestes testes a escalamento vertical ou horizontal e ao mesmo tempo eliminar o *bottleneck* do *hardware*.

Os resultados deste segundo teste foram mais satisfatórios e permitem inferior que esta hipótese está também cumprida. Os resultados um pouco inferiores aos do monólito podem ser explicados pelo facto dos micro-serviços estarem a ser executados sobre um ambiente de *containers* com a tecnologia *docker*, sendo esta ultima teoria comprovada com base nos testes apresentados na secção 6.2.2.

Estes resultados permitem, desta forma, aferir que o sistema desenvolvido confere, garantindo a mesma experiência de utilização, uma performance não inferior quando submetido às mesmas condições de escalamento horizontal e vertical, corroborando os pressupostos adjacentes a esta dissertação (*c.f. secção 1.3*).

# Capítulo 7

# Conclusões

Esta dissertação permitiu conhecer um conceito novo e disruptivo para redesenhar a *Web* que está a ser desenvolvido e alavancado por diferentes projetos e comunidades (*c.f.* secção 2.1).

Numa primeira fase as diferentes alternativas com maior potencial foram estudadas com vista a perceber de que forma respondiam a desafios tecnológicos como escalabilidade, segurança e privacidade, este estudo foi refletido tanto no estado da arte do presente documento (*c.f.* capítulo 2), como também em contribuições públicas (*c.f.* secção 1.6).

O estudo das diferentes alternativas permitiu escolher um projeto para aprofundar numa perspetiva mais técnica com vista a criar um impacto positivo de escalabilidade ao mesmo tempo que suscitou uma mentalidade de contribuir para uma comunidade *open-source*. Tratando-se de um projeto cujo objetivo passa por revolucionar a *Web*, estas contribuições implicaram pensar fora da caixa num mundo já repleto de fortes alicerces implantados por grandes empresas e grandes engenheiros da área da informação e tecnologias de comunicações (ICT).

O *Solid* foi o projeto escolhido e sobre o qual incidiu uma grande parte do trabalho relatado nesta dissertação. O esforço no sentido de migrar a arquitetura monolítica obrigou a perceber todo o trabalho até então realizado e distribuído por mais de 20 repositórios no *github* (*c.f.* secção 1.6). Para além do *solid-server* proposto (*c.f.* secção 4.1), houve também trabalho desenvolvido no sentido de criar provas de conceito com interface gráfica, uma delas com co-autoria desenvolvida no sentido de colaboração com uma tese de dissertação do *MIT* e outra como forma de suporte à apresentação do presente documento.

## 7.1 Questões de investigação e contribuições

Desde o início que o problema foi decomposto em questões mais simples com vista a colmatar as diferentes indefinições.

**Q1: Como pode o *Solid* ser escalado horizontal e verticalmente?**

O escalamento vertical não deverá ser um problema, uma vez que corresponde a aumento de recursos que suportam a instância a executar.

Por outro lado, o escalamento horizontal é um problema na arquitetura atual, na medida em que cada nova execução do *Solid* corresponde a um nova instância totalmente independente das outras que existem no mundo (*c.f.* figura 4.1).

Uma possível solução consiste em embeber os diferentes serviços (no caso da arquitetura orientada a micro-serviços) em *containers* (utilizando por exemplo a tecnologia *docker*) e executar estes *containers* através de uma ferramenta orquestração (e.g. *kubernetes*).

É importante denotar que para isto ser possível é também necessário que os recursos estáticos estejam disponíveis e de alguma forma sincronizados entre as diferentes instâncias de um determinado serviço(*c.f.* secção 4.1).

**Q2: É possível existir apenas uma implantação do *Solid* escalada de forma descentralizada e anónima?**

Uma implantação única e global do *Solid* seria uma possível evolução para o conceito. Esta evolução está dependente da migração para uma arquitetura orientada a micro-serviços, da adoção de um mecanismo de descoberta de novas instâncias para determinado serviço (*c.f.* capítulo 4).

**Q3: O mecanismo de autenticação utilizado atualmente pode ser mantido tendo em conta a migração para uma arquitetura orientada a micro-serviços?**

O mecanismo de autenticação utilizado atualmente pelo *Solid* é o *Open-ID Connect* em combinação com a atualização de *tokens* de acesso em formato *JWT* (*c.f.* secção 2.1.1.1).

O fluxo de autenticação do protocolo *Open-ID connect*, cria uma clara separação entre o componente responsável por gerir a autenticação (*Identity Provider* e o componente que serve recursos com base na autenticidade do *token* de acesso (*Resource Server* (*c.f.* figura 2.1). O *token* garante que a validação de autenticidade não depende do componente emissor, e desta forma contribui para o baixo acoplamento característico numa arquitetura orientada a micro-serviços (*c.f.* secção 2.3.2.1).

## 7.2 Resultados

o trabalho desenvolvido demonstra que é possível migrar a atual arquitetura monolítica do *Solid* para uma arquitetura orientada a micro-serviços capaz de garantir escalabilidade horizontal e vertical dos diferentes serviços.

Esta nova arquitetura não compromete nem as funcionalidades de negócio nem a segurança, sendo possível manter todos os mesmos protocolos que suportam o projeto hoje em dia.

## 7.3 Limitações

No seguimento do trabalho desenvolvido é possível denotar que existem limitações que devem ser tidas em conta, das quais destacam-se:

- Dependência sistema de ficheiros - O sistema *Storage Web* e o *Storage Consumer* tem uma dependência do sistema de ficheiros como forma de persistência que pode ser um possível *bottleneck* em termos de escalabilidade desta nova arquitetura, devendo ser estudada a migração para um sistema de ficheiros virtual ou mesmo uma diferente forma de persistência mais robusta;

- Eventual falta de consistência - Característico das arquiteturas orientadas a micro-serviços, conforme explicado pelo teorema CAP (*c.f.* secção 2.3.2.8), é possível em determinadas circunstâncias o sistema *Solid* ter de lidar com eventuais faltas de consistência entre os diferentes serviços. As situações em que a utilização fica comprometida devem ser identificadas e criados mecanismos de mitigação.

## 7.4 Trabalho futuro

No mesmo sentido, são identificados pontos de trabalho futuro que deverão servir como forma de continuação ao trabalho desenvolvido no decorrer da dissertação:

- Orquestração de *containers* - O projeto desenvolvido foi desenhado no sentido de todos os diferentes micro-serviços poderem ser executados em *containers*. Assim, os próximos passos passam pela integração com uma ferramenta de orquestração de *containeirs* (e.g. *kubernetes*);

- *Continuous Integration* e *Continuous Delivery* - A adoção em grande escala deste projeto implica o desenvolvimento de *pipelines* robustas de integração e entrega contínua.

## 7.5 Apreciação final

A disrupção foi certamente o combustível de motivação para o desenvolvimento desta dissertação, que, numa perspetiva global, cumpriu os objetivos inicialmente propostos, permitindo extrair conclusões e desafios futuros com potencial de contribuição positiva para uma sociedade cibernética mais protegida e livre.

# Bibliografia


[1] A. K. Dawit Yiman, "Centralization vs. decentralization issues in internet-based knowledge," 2019. [cited on p. 1, 2, 7]

[2] Georges Abi-Heila, "Your Facebook data is creepy as hell." https://medium.com/@John.Val.John/your-facebook-data-is-creepy-as-hell-319ae47117e6, Last access: February 12, 2020. [cited on p. 1, 7]

[3] Mike Butcher, "Cambridge Analytica email chain with Facebook sheds new light on data misuse scandal." shorturl.at/fGR07, Last access: February 12, 2020. [cited on p. 1, 29]

[4] Jay Hamideh, "Why decentralization is the future for social media." https://arcticstartup.com/why-decentralization-is-the-future, Last access: July 16, 2020. [cited on p. 1, 2, 7]

[5] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. John Wiley and Sons, 2015. [cited on p. 3, 20]

[6] Rauno Pello, "Design science research — a short summary." https://medium.com/@pello/design-science-research-a-summary-bb538a40f669, Last access: June 27, 2020. [cited on p. 3]

[7] N. B. Pedro Pinto, Pedro Piloto, "A study about web development frameworks focused on users' privacy," vol. 1, no. 3, 2019. [cited on p. 4]

[8] Pedro Piloto, "Solid server microservices." https://github.com/pedropiloto/solid-server-microservices, Last access: June 27, 2020. [cited on p. 4]

[9] Pedro Piloto, "OpenID Connect for Nodejs." https://github.com/pedropiloto/oidc-op-adapted, Last access: June 27, 2020. [cited on p. 4]

[10] Pedro Piloto, "OpenID Connect Relying Party (oidc-rp)." https://github.com/pedropiloto/oidc-rp-adapted, Last access: June 27, 2020. [cited on p. 4]

[11] Pedro Piloto, "OIDC Decentralized Authentication Manager." https://github.com/pedropiloto/oidc-auth-manager-adapted, Last access: June 27, 2020. [cited on p. 4]

[12] Pedro Piloto, "A library for reading and writing to Solid pods." https://github.com/pedropiloto/solid-auth-client-adapted, Last access: June 27, 2020. [cited on p. 4]

[13] Pedro Piloto, "Solid filemanager." https://github.com/pedropiloto/solid-filemanager-adapted, Last access: June 27, 2020. [cited on p. 4]

[14] Pedro Piloto, "Steve Clinic." https://github.com/pedropiloto/steve-clinic, Last access: June 27, 2020. [cited on p. 4]

[15] Nathaniel Popper, "Twitter and Facebook Want to Shift Power to Users. Or Do They? ." `https://www.nytimes.com/2019/12/18/technology/facebook-twitter-bitcoin-blockchain.html`, Last access: February 12, 2020. [cited on p. 8]

[16] Solid, "Solid." `https://solid.mit.edu/`, Last access: February 10, 2020. [cited on p. 8]

[17] M. Massé, *REST API Design Rulebook*. O'Reilly, 2011. [cited on p. 8]

[18] SOLID, "Solid." `https://github.com/solid/solid`, Last access: February 10, 2020. [cited on p. 8, 9, 11, 16]

[19] Solid, "Solid WebID-TLS Protocol Spec." `https://github.com/solid/solid-spec/blob/master/authn-webid-tls.md`, Last access: February 10, 2020. [cited on p. 9]

[20] Solid, "WebID-OIDC Authentication Spec." `https://github.com/solid/webid-oidc-spec`, Last access: February 10, 2020. [cited on p. 9, 10]

[21] Solid, "Solid." `https://github.com/solid/web-access-control-spec`, Last access: February 10, 2020. [cited on p. 11, 16]

[22] B. PBC, "Blockstack technical whitepaper v 2.0," 2019. [cited on p. 12, 13, 17]

[23] Diaspora, "An introduction to the Diaspora source." `https://wiki.diasporafoundation.org/An_introduction_to_the_Diaspora_s`, Last access: February 10, 2020. [cited on p. 14, 17]

[24] Mayur Ingle, "A Bit About Blockchain." https://dzone.com/articles/a-bit-about-blockchain, Last access: February 12, 2020. [cited on p. 14]

[25] E. Foundation, "Elastos white paper," 2018. [cited on p. 14, 15, 17]

[26] Codrut Neagu, "What are P2P (peer-to-peer) networks and what are they used for?." https://www.digitalcitizen.life/what-is-p2p-peer-to-peer, Last access: February 12, 2020. [cited on p. 15, 22]

[27] Elastos, "Elastos Documentation." https://developer.elastos.org/, Last access: February 10, 2020. [cited on p. 15, 17]

[28] IPFS, "IPFS powers the Distributed Web." https://ipfs.io/, Last access: February 12, 2020. [cited on p. 15]

[29] S. D. B. A. C. E. S. F. S. G. F. A. J. P. S. P. R. S. Peter A. Koen, Greg Ajamian, "1 fuzzy front end : Effective methods , tools , and techniques," 2002. [cited on p. 18]

[30] T. L. Saaty, "Decision making with the analytic hierarchy process," vol. 1, no. 3, pp. 83 – 98, 2008. [cited on p. 19]

[31] Value Analysis, "VA In Depth." https://www.valueanalysis.ca/fast.php, Last access: February 15, 2020. [cited on p. 19]

[32] Team Pitchspot, "The Business Model Canvas Explained." https://medium.com/pitchspot/the-business-model-canvas-explained-1f5b76207f7f, Last access: February 15, 2020. [cited on p. 20]

[33] Nis Frome, "Experimentation principles." https://medium.com/coderbyte/experimentation-principles-6dc06e4a4b3c, Last access: July 01, 2020. [cited on p. 20]

[34] Mohamed Aladdin, "Software Architecture - The Difference Between Architecture and Design." https://codeburst.io/software-architecture-the-difference-between-architecture-and-design-7936abdd5830, Last access: July 04, 2020. [cited on p. 20]

[35] E. Wolff, *Microservices: Flexible Software Architecture*. Addison-Wesley Professional, 2016. [cited on p. 20, 25]

[36] Mrityunjay Kumar, "Microservices Architecture: What, When, and How." https://dzone.com/articles/

microservices-architecture-what-when-how, Last access: February 18, 2020. [cited on p. 20]

[37] H. Cervantes, *Designing Software Architectures*. Prentice Hall, 1st ed., 2016. [cited on p. 21]

[38] R. C. Martin, *Clean Architecture*. Prentice Hall, 2016. [cited on p. 22]

[39] Wellington Nascimento, "Understanding JWT tokens (Json Web Token)." https://medium.com/tableless/entendendo-tokens-jwt-json-web-token-413c6d1397f6, Last access: June 28, 2020. [cited on p. 22]

[40] Martin Fowler, "BoundedContext." https://martinfowler.com/bliki/BoundedContext.html, Last access: February 18, 2020. [cited on p. 23]

[41] Brian Sletten, Chase Doelling, "Foundations of RESTful Architecture." https://dzone.com/refcardz/rest-foundations-restful?chapter=1, Last access: February 18, 2020. [cited on p. 23]

[42] P. K. e. R. P. Neal Ford, *Building Evolutionary Architectures*. O'Reilly, 2017. [cited on p. 24]

[43] I. N. Ronnie Mitra, *Microservices: Up and Running: A Step-by-Step Guide to Building a Microservice Architecture*. O'Reilly, 1st ed., 2019. [cited on p. 25]

[44] Martin Fowler, "CQRS." https://martinfowler.com/bliki/CQRS.html, Last access: February 22, 2020. [cited on p. 25]

[45] Davis Levine, "Service Design Patterns)." https://medium.com/@davislevine/service-design-patterns-930203c8df37, Last access: July 01, 2020. [cited on p. 25]

[46] Akhil Mehra, "Understanding the CAP Theorem." https://dzone.com/articles/understanding-the-cap-theorem, Last access: February 22, 2020. [cited on p. 25]

[47] R. C. Martin, *Clean Code*. O PRHA, 2009. [cited on p. 26]

[48] Divya Dua, "Introduction to Event Sourcing." https://dzone.com/articles/introduction-to-event-sourcing, Last access: February 22, 2020. [cited on p. 26]

[49] Mahendra Choudhary, "Introduction to JavaScript: Basics." https://medium.com/swlh/introduction-to-javascript-basics-cf901c05ca47, Last access: July 04, 2020. [cited on p. 26]

[50] A. D. Scott, *JavaScript Everywhere: Building Cross-Platform Applications with GraphQL, React, React Native, and Electron.* O'Reilly, 1st ed., 2018. [cited on p. 26]

[51] Node.js, "Introduction to Node.js ." https://nodejs.dev/learn, Last access: July 04, 2020. [cited on p. 26]

[52] Robert Gibb, "What is YAML?." https://blog.stackpath.com/yaml/, Last access: July 04, 2020. [cited on p. 26]

[53] Maduha Jamal, "UML Diagrams: What you need to know?." https://medium.com/datadriveninvestor/uml-diagrams-what-you-need-to-know-d07d4101e7c4, Last access: July 04, 2020. [cited on p. 27]

[54] R. M. Kim Hamilton, *Learning UML 2.0.* O'Reilly, 1st ed., 2006. [cited on p. 27]

[55] Good Firms, "What is a Software Tools?." https://www.goodfirms.co/glossary/software-tools/, Last access: July 04, 2020. [cited on p. 27]

[56] B. Stopford, *Designing Event-Driven Systems.* O'Reilly, 1st ed., 2018. [cited on p. 27]

[57] JMeter, "Apache JMeter$^{TM}$." `https://jmeter.apache.org`, Last access: July 01, 2020. [cited on p. 27]

[58] B. D. M. Antonio Gomes Rodrigues, Philippe Mouawad, *Master Apache JMeter - From Load Testing to DevOps: Master performance testing with JMeter.* Packt Publishing, 1st ed., 2019. [cited on p. 27]

[59] Rajat Sharma, "NPM basics in less than 10 minutes." `https://medium.com/swlh/npm-in-less-than-10-minutes-6b321d566271`, Last access: July 01, 2020. [cited on p. 28]

[60] Jonas DeMuro, "What is container technology?)." `https://www.techradar.com/news/what-is-container-technology`, Last access: July 01, 2020. [cited on p. 28]

[61] P. S. Kocher, *Microservices and Containers.* Addison-Wesley Professional, 1st ed., 2018. [cited on p. 28]

[62] A. Osterwalder, *The Business Model Ontology.* John Wiley and Sons, 2004. [cited on p. 33]

[63] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith.* O'Reilly Media, 2019. [cited on p. 37]

[64] R. Patton, *Software Testing*. Sams publishing, 2nd ed., 2005. [cited on p. 62]

[65] Little Data, "What is the average server response time?." `https://www.littledata.io/average/server-response-time`, Last access: July 01, 2020. [cited on p. 62]

[66] R. B. Dorothy Graham, Erik P. W. M. Veenendaal, *Foundations of Software Testing*. Cengage Learning Emea, 1st ed., 2006. [cited on p. 62]