Universidade do Minho
Escola de Engenharia
Departamento de Informática

Paulo Jorge Pereira Martins

Development of an e-portfolio social network using emerging web technologies

December 2021

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Paulo Jorge Pereira Martins

Development of an e-portfolio social network using emerging web technologies

Master dissertation
Master's in Informatics Engineering

Dissertation supervised by
José Carlos Leite Ramalho

December 2021

ACKNOWLEDGEMENTS

The pandemic year, paradoxically, was one of the most productive years in my life, but also one of the most time-consuming: for that reason I would like to thank my family for the loving support and the working hours not spent with them. To my newborn child: thank you for the motivation and choosing me.

I would also like to express my gratitude towards my supervisor, José Carlos Ramalho, for his immense expertise and constant availability. Grateful for him supporting each new project with encouragement, believing, amongst others, in the Major Minors and current dissertation from the get go way before even myself believed in it, and sharing the needed knowledge and encouragement towards the many achievements of 2021 related with this project.

Also very grateful to the Department of Informatics, for the opportunity in participating in the Bosch "Ride Care" project between 2020/2021, which enriched my experience, and the kind invitation and guidance from his headmaster, Pedro Rangel Henriques, to teach two classes, which has been a very joyful experience.

This year I rediscovered the joy of science/research and teaching, which I thought I had long lost, very grateful for each and every opportunity.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Digital portfolios (also known as e-Portfolios) can be described as digital collections of artifacts, being both a product (a digital collection of artifacts) and a process (reflecting on those artifacts and what they represent). It is an extension of the traditional Curriculum Vitae, which tells the educational and professional milestones of someone, while the portfolio proves and qualifies them (e.g.: annually thousands of students finish a Master degree on Informatics, but only one has built Vue, Twitter or Facebook – the Portfolio goes beyond the CV milestones by specifying the person's output throughout life and distinguishing them). e-Portfolios augment this by introducing new digital representations and workflows, exposed to a community, being both a product and a process. This approach can be useful for individual self-reflection, education or even job markets, where companies seek talented individuals, because it expands the traditional CV concept and empowers individual merit. There have been many studies, theories, and methodologies related with e-Portfolios, but transpositions to web applications have been unsuccessful, untuitive and too complex (in opposition to the CV format, which had success in various applications, for example LinkedIn).

This project aims to study new approaches and develop an exploratory web/mobile application of this methodology, by exploring the potential of social networks to promote them, augmented by emergent web technologies. Its main output is the prototype of a new product (a social network of e-Portfolio) and its design decisions, with new theoretical approaches applied to web development. By the end of this project, we will have idealized a web infrastructure for interacting with networks of users, their skills, and communities seeking them.

The approach to the development of this platform will be to integrate emerging technologies like WebAssembly and Rust in its development cycle and document our findings. At the end of this project, in addition to the prototype of a new product, we hope to have contributed to the State of the Art of Web Engineering and to be able to answer questions regarding new emerging web development ecosystems.

KEYWORDS e-Portfolios, Microservices, WebAssembly, Rust, Web Engineering.

RESUMO

Os portfólios digitais (também conhecidos como e-Portfolios) podem ser descritos como coleções digitais de artefatos, sendo tanto um produto (uma coleção digital de artefatos) quanto um processo (refletindo sobre esses artefatos e o que eles representam). É uma extensão do tradicional Curriculum Vitae, onde o primeiro conta os marcos educacionais e profissionais de alguém, enquanto que o segundo, o Portfólio, comprova-os e qualifica-os (e.g.: anualmente milhares de alunos concluem graduações em Informática, no entanto apenas um consebeu o Vue, o Twitter ou o Facebook - o Portfólio vai além dos indicadores quantitativos do CV, especificando e qualificando a produção da pessoa ao longo da vida e distinguindo-a). Os e-Portfolios expandem este conceito com a introdução de novas representações digitais e fluxos de trabalho, expostos a uma comunidade, sendo tanto um produto como um processo. Esta abordagem pode ser útil para a autorreflexão individual, educação ou mesmo mercados de trabalho, onde as empresas procuram indivíduos talentosos, porque expande o conceito tradicional de CV e potencializa o mérito individual. Existem muitos estudos, teorias e metodologias relacionadas com os e-Portfolios, mas as transposições para aplicações web têm sido mal sucedidas, pouco intuitivas e muito complexas (em oposição ao formato CV, que tem tido sucesso em várias aplicações, por exemplo no LinkedIn).

Este projeto visa estudar novas abordagens neste domínio e desenvolver uma aplicação exploratória web/mobile que melhor exprima os e-Portfolios, explorando o potencial das redes sociais para os promover em conjunto com tecnologias web emergentes. As principais produções esperadadas deste trabalho são um protótipo de um novo produto (uma rede social de e-Portfolio) e documentar novas abordagens teóricas aplicadas ao desenvolvimento web. No final deste projeto, teremos idealizado uma infraestrutura web para interagir com redes de utilizadores, as suas competências e comunidades que os procurem.

A abordagem ao desenvolvimento desta plataforma será integrar tecnologias emergentes como WebAssembly e Rust no seu ciclo de desenvolvimento e documentar as nossas descobertas e decisões. No final deste projeto, para além do protótipo de uma plataforma, esperamos ter contribuido para o Estado da Arte da Engenharia Web e responder a questões sobre novos ecossistemas emergentes de desenvolvimento web.

PALAVRAS-CHAVE e-Portfolios, Microsserviços, WebAssembly, Rust, Engenharia Web.

CONTENTS

LIST OF FIGURES

ACRONYMS

AGILE Agile Software Development. 19
AI Artificial Intelligence. 68
AMQP Advanced Message Queuing Protocol. 25, 68, 75
API Application Programming Interface. *Glossary:* Application Programming Interface, 70, 71, 77, 78, 85, 95, 97, 103, 104, 108
AWS Amazon Web Services. 21

CBA Competency based approach. 13
CDN Content Delivery Network. 69
CLI Command Line Interface. 94, 95, 103, 128
CNCF Cloud Native Interactive Landscape. 17, 35
CSS Cascading Style Sheets. 42
CSV Comma Separated Values. 98
CV Curriculum Vitae. c, d, 5, 7, 12, 13, 14, 15, 43, 44, 51, 52, 53, 55, 56, 57, 58, 60, 61, 63, 64, 65, 78, 85, 112, 113, 114

DOM Document Object Model. 32
DSD Data structure diagram. vii, 82, 83
DSR Design Science Research. vi, 6, 7, 9, 44, 45, 46, 48, 51, 59, 88, 102, 107, 113

e-Portfolio Electronic Portfolio. c, d, vi, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 43, 44, 47, 51, 53, 54, 55, 56, 57, 58, 59, 60, 61, 63, 64, 65, 85, 94, 103, 105, 112, 113, 114, 115
ESCO European Skills, Competences, qualifications and Occupations Ontology. 14, 27, 28, 57, 60, 78, 81, 83, 85, 93, 98, 99, 100, 102, 112, 115
EU European Union. vi, 13, 14, 93, 100, 102, 112, 115

GUI Graphical User Interface. 42, 71, 89, 91, 92, 94, 95, 97, 98, 103, 104, 128

HTML Hypertext Markup Language. 42
HTTP Hypertext Transfer Protocol. 25, 70
HTTPS Hyper Text Transfer Protocol Secure. 70

IoT Internet of Things. 67
IP Internet Protocol. 70, 75

JIT Just-in-time compilation. 41

JNI Java Native Interface. 90
JSON JavaScript Object Notation. 94, 98
JWT JSON Web Token. 71

K8s Kubernets. 7, 8, 22, 76, 102, 113, 114, 115

LAMP Linux, Apache, MySQL, PHP. 6
LEMP Linux, Nginx, MySQL, Python. 68
LXC Linux Container. 20

MEAN MongoDB, Express, Angular, NodeJS. 68
MERN MongoDB, Express, React, NodeJS. 68
MEvN MongoDB, Express, Vue, NodeJS. 68
MVC Model-View-Controller. 6, 59

NLP Natural Language Processing. 94

OAS OpenAPI Specification. 70
ORM Object Relational Mapping. 81, 83
OWL Ontology Web Language. 27, 41, 102, 113

PBT Planned Behavior Theory. 47
PC Personal Computer. 86
POC Proof of Concept. 5, 47, 48, 49, 51, 53, 58, 59, 102, 112, 113
PWA Progressive Web App. 59, 68, 86, 88, 92

RDF Resource Description Framework. 27, 29, 36, 77, 78, 98, 102, 113
REST Representational State Transfer. 70, 78, 94, 108

SEO Search Engine Optimization. 31
SLR Self-regulated learning. 13
SOA Service Oriented Architecture. 16, 72, 73
SPA Single Page Application. 68, 69, 70, 85, 88, 90, 91, 92
SPARQL SPARQL Protocol and RDF Query Language. 27, 36, 77, 78, 98, 99
SWOT Strengths, Weaknesses, Opportunities, and Threats. 44, 46, 47, 48

TAM Technological Acceptance Model. 44, 46, 47, 48, 49, 104, 111, 114

UI User Interface. 86
URL Uniform Resource Locator. 69
UX User Experience. 86

W3C World Wide Web Consortium. 42

WASM WebAssembly. c, d, 6, 7, 8, 9, 21, 26, 32, 35, 36, 37, 38, 40, 41, 42, 43, 59, 60, 61, 68, 69, 89, 92, 93, 103, 106, 107, 109, 110, 111, 113, 114, 115

WWW World Wide Web. 6, 67

YAML YAML Ain't Markup Language. 71

Part I

CONTEXT - INTRODUCTORY MATERIAL

1

INTRODUCTION

This chapter introduces the underlying motivation for this project, its goals and objectives that must be accomplished. Furthermore, it also describes the structure of the remaining document.

1.1 CONTEXT & GOALS

This project aims to create a social network of e-Portfolios using novelty emergent technologies and methodologies. In the end, it is expected that we will have built a web infrastructure based on microservices and new tech stacks that will feed a mobile application and a website where individuals and companies will be able to dialogue via e-Portfolios. We named this resulting product *Hiperfolio* (originating from the junction of two words: from the English *hiper*, meaning very active, and *folio*, with origin in Latin, meaning sheets of paper).

Our objective is to conceptualize and develop a Proof of Concept (POC) of this web product, reflecting on new strategies for applying e-Portfolios in a digital context and improving on the traditional static CV model, showcasing new approaches for interacting with social graphs of users, and ways to bring together their skills and communities/companies with common interests. This process will also serve as a reflection on the current state of Web Engineering, we aim to explore emergent technologies and novelty approaches during its implementation, rethinking the implementation of a web application in an high-availability and scalability context.

Digital portfolios (also known as e-Portfolios) can be described as digital collections of artifacts, being both a product (a digital collection of artifacts) and a process (reflecting on those artifacts and what they represent). It is an extension of the traditional Curriculum Vitae, where the first tells the educational and professional milestones of someone, the later, the Digital Portfolio, proves and qualifies them (e.g.: annually thousands of students finish a Master degree in Computer Science, but only one has built Vue, Twitter or Facebook – the Portfolio goes beyond the CV milestones, by also specifying the person's output throughout life and individually distinguishing them). This approach can be useful, for example, for companies seeking talented individuals, because it expands the CV concept (but not only that, in further sections we will expand on this concept that goes beyond the job market, it can also be integrated in education or self-realization). There are some services for Digital Portfolios creation (in the following sections we will mention some), but the majority are obsolete and without significant success like, for instance, LinkedIn, that uses the same concept in the context of the traditional Curriculum Vitae - this project aims to explore the potential of social networks to expand upon the e-Portfolios methodology.

During this project development, we adopted a Design Science Research (DSR) methodology composed of multiple recursive cycles of product design and evaluation, where the end product is not the only and main target, but also the process of creating and evaluating it can generate new knowledge - the exploratory nature of this project, the reflection, evaluation and experimentation of new approaches will result on new data and prototypes for the scientific community to reflect upon. The project is mainly exploratory, expecting a final prototype, but also a reflection upon new methodologies to implement it (e.g. it would be easy to develop this project using tradition and well-known methods like a monolithic MVC architecture and obsolete tech stacks like LAMP, but it would not provide new relevant data or discussion to the process, for that reason in each choice during our project we seek the most novelty and challenging route, because these not well tested or documented approaches are the ones that would provide new and most significant insights for the scientific community, and to the reflection upon the future of modern web development).

Some of the main novelty emerging technologies that we integrated, where related with Microsservices architecture (in various manifestations), Semantic Web, the Rust programming language and the new web standard WebAssembly. Throughout their integration, we analyzed their contribution to the contemporary Web Development. The first, Microsservices, have been revolutionizing software development and practices like AGILE and DevOps, but not only in the context of distributed systems and Cloud backends, it is also manifesting itself on other fronts like Frontend development for example. Semantic Web is opening the doors for the WWW of the future, the Web 3.0. The later, Rust and WebAssembly, were developed by Mozilla in recent years, being in a stage of early adoption, but with a very fast growth and promising to revolutionize some areas (e.g. Rust was the most trending language in the TIOBE index in 2020; NPM was rewrote using it; Deno, the successor to NodeJs, is being coded with Rust; Microsoft rewrote some Windows components using it and now Linux is also adopting it for developing its Kernel; won the StackOverflow Survey “Most Loved Language of the Year” for 5 years in a row since the release of it’s first stable version; etc.). Even though in early stages, it’s ecosystem of libraries for web development is very impressive, being at the same time a low-level programming language with C like performances, but at the same time it has security and maintainability as one of its main mottos competing with high-level languages. On the other hand, WebAssembly (WASM), is a low level, assembly like language that can be run on browsers (Rust was an early adopter with full compilation support). Theoretically it could improve and revolutionize web development (but not only, also desktop, because it can run in interpreters or imported) with great impact on the future of development (compile once, run everywhere, including desktop or in the browser, with Assembly performance), but due to its novelty it lacks real world examples yet. This project aims to integrate these emerging technologies and analyze their potential contribute to the future of web development.

1.2 WORK PLAN

This project is centered around three key concepts: it aims to integrate the concept of **e-Portfolio** in a **social network** context, using **emergent technologies**. The goal is to expand both concepts to new domains and target audiences with the objective of publishing the final product in a real business context - for this reason it would not be enough to develop a minimal and obsolete infrastructure for the social network component, our proposal en-

compasses a scalable solution with a vision for the future, framed in the most current and emerging technological possibilities, which is the only way to stand out in such a competitive market, while also contribution to the scientific discussion of these thematics. This dissertation will therefore address two aspects: the **conceptual**, that is, the idealization of the fusion of the concepts of e-Portfolio with social network in a new product and, in a second aspect, the **technical**, that is, the technologies, challenges and design decisions regarding the implementation of this product in the context of emerging technologies for web development, in particular Microservices, Semantic Web, Rust and WebAssembly (WASM).

Our approach to the development of this product was to integrate these emerging technologies (microservices, micro-frontends, K8s, WebAssembly, Rust, Semantic Web, etc.) into its development cycle. At the end of this project, in addition to providing the prototype of a product (web and mobile applications), we will be able to answer questions about what are the advantages of microservices architectures and challenges, in which way Semantic Web could be the future Web 3.0, what is the state of the art of WASM and Rust, what is the current status of the web development ecosystem and challenges to the future, etc.

Before theorizing and documenting the implementation of this project, it is urgent to define and understand the objects under study. The next chapter (2 intends to present a brief and succinct theoretical contextualization of the main themes and tools that were central to its implementation. Namely, we will focus the State of the Art, theoretical framework and history of the most novelty and essential concepts, like e-Portfolios, Microservices (and derived technologies), Web Semantic and Web 3.0, *Micro Frontends*, WebAssembly and *Rust*.

We will then proceed to define the methodologies, goals and requirements in chapter 3 (namely, we opted for a DSR methodology focused in an exploratory cycle of design and evaluation), concluding this introductory contextualization. The second part of this thesis (part ii) will focus the concrete implementation of the product, starting by an analysis and theorizing regarding the atomic elements of an e-Portfolio and strategies related with how to effectively transpose and increment it with a Social Network approach (chapter 4. This theoretical basis will be used to start a design and evaluation cycle of the project, and chapter 5 will summarize the central technical aspects the resulting product. The last part (iii will summarize and analyze the outputs of the project, starting by analyzing the obtained data, methodologies and products (chapter 6 and last but not least concluding remarks (chapter 7). We also highlight the last appendix section (iv), where we shared Git repositories for the source-codes, endpoints for running prototypes, surveys used, published articles and conferences where material where disclosed, etc.

1.3 SUMMARY

In this chapter we established a roadmap for the current project, underlying the motivation, questions and the main objectives we aim to achieve. We can summarize this question in the following way:

- How to convert the e-Portfolios concept to an intuitive web/mobile product?
- What distinguishes Curriculum Vitae based products from the e-Portfolio?
- Which projects already exist on this area, did they succeed and why?

- How to reimagine Social Networks with an e-Portfolio baseline?
- How to reimagine architectures for high-availability and scalability?
- Which emergent technologies could redefine Web Engineering for the following decades?
- How to integrate Microsservices inspired architectures into a Social Network context and how could they benefit from it?
- How can Microsservices (in specific orchestration through Kubernets) contribute to the future of Web Development?
- How can Rust contribute to the future of Web Development?
- How can WebAssembly contribute to the future of Web Development?
- How can Semantic Web approaches contribute to the future of Web Development?
- What is the relevance of Micro Frontends to the future of Web Development?

We also summarized a brief overview of the document structure and what aspects further chapters will cover. It was divided into three main parts: Part I: Context (i); Part II: Theory & Implementation (ii); Part III: Outputs (iii. These parts were divided into chapters that we summarized in more depth.

2

STATE OF THE ART

This project aims to integrate and expand the concept of a e-Portfolio into a social network context, using emergent technologies. The goal is to expand both concepts to new domains and target audiences with the objective of publishing a final product in a real business context - for this reason it would not be enough to develop a minimal and obsolete infrastructure for the social network component, our proposal targets a scalable high-availability base with a vision in the long term, framed in the most current and emerging technological possibilities, which is the only way to stand out in such a competitive market. This exploratory approach would also be the best route of action to obtain new data and stimulate the scientific debate in the context of a Design Science Research which we target, there would be no novelty value in researching and developing this product with obsolete methodologies. This dissertation will therefore address two aspects: the conceptual, that is, the idealization of the fusion of the concepts of e-Portfolio and social network into a new product approach and, in a second layer, the technical aspect, that is the technologies, challenges and design decisions regarding the implementation of this product in the context of novelty emergent technologies and their contribute to the future of web development, in particular Microservices, Semantic Web, Rust and WebAssembly.

Before idealizing and documenting the implementation of this project, it is important to define and understand the concepts and technologies under study. The current chapter intends to present a brief and succinct theoretical contextualization of the most important themes and tools that are central to this project implementation - we will focus on the state of the art, theoretical framework and history of the most novelty and essential elements applied during development and conceptualization, namely: e-Portfolios, Microservices (and derived technologies), Web 3.0, Semantic Web, *Micro Frontends*, WebAssembly and *Rust*.

2.1 E-PORTFOLIOS

In the following sections, based on literature review, we'll summarize the origins of e-Portfolio, and the its context and application today. e-Portfolios are the key thematic for the current project.

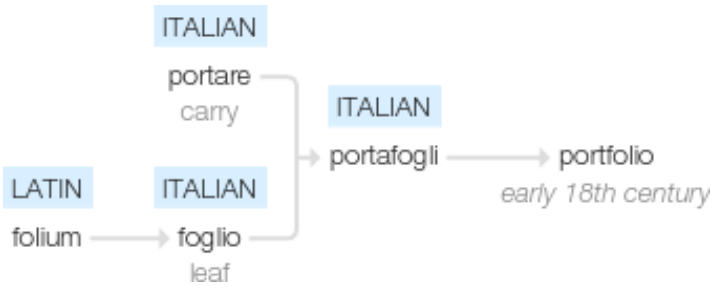

Figure 1: Evolution of the word *portfolio*

2.1.1 *Origins*

The origin of e-Portfolios[1], is not easy to define because it encompasses several perspectives and a thinly delimited temporal boundary as to its origins. We can generalize by saying that the implementation of e-Portfolios coincides with the emergence of personal computers, because by definition the term assumes a digital medium for its expression. The term derives from traditional portfolios, expanding the concept through a symbiosis with "new technologies" introduced by the digital age.

Initially, when personal computers were popularized, the diffusion of e-Portfolios was more rudimentary, carried out, for example, by sharing files via floppy disks, CDs, DVDs, etc. During the 90s and Web 1.0 the popularization of *homepages* resulted in a new level of expressiveness. But it was with *web 2.0*, more interactive websites based on user-generated content that the concept was really expanded: first through access to intuitive and interactive tools, second through access to ways of easily publishing and sharing content and facilitated exposure to different types of communities.

web 2.0 has come to mediate the technological literacy divide, introducing intuitive tools that facilitate expressiveness and global reach by all types of users, with or without technological literacy. In addition, it is legitimate to defend that the internet can be considered one of the most important technological revolutions of humanity, because it democratized access to information and communication, being a materialization of the freedom and enlightenment utopia that begun in the Age of Enlightenment (XVIII century). Never before in human history has the library of Alexandria fit into the palm of the ordinary individual's hand, nor have the doors of the ancient Agora of Athens been so wide open and universally open to visitors for democratic debate, today a common smartphone and internet connection is enough to obtain access to human rights that for millennia have been restricted from most of mankind.

Portfolios predate the e-Portfolio, in fact the basic concept has been around for a long time, initially very commonly adopted by artists to exhibit their work. The term portfolio as we know it today has its origins in Italy, in the 18th century, where the fusion of "portare" (carry) and "foglio" (sheet) resulted in "portfogli", meaning something that carries a set of sheets of paper (see Figure 1).

e-Portfolios expanded the traditional portfolios with the introduction of three axes (Figure 2) supported by the digital age: interactivity/creativity (through technology there are infinite ways to present and relate content to the

1 Other terms also referring to e-Portfolio: Electronic Portfolio, Digital Portfolio, Online Portfolio, e-Folio, etc.

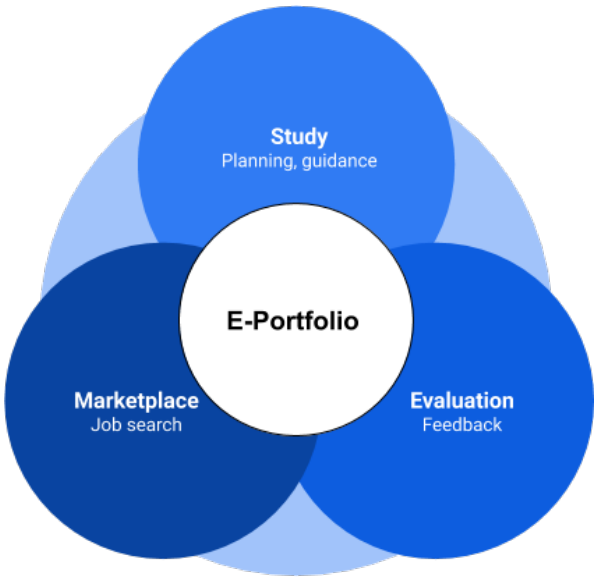

Figure 2: Triple range of action of e-Portfolios

public), universality (every individual has access the same technologies and media exposure) and community (immediate access to exposure and debate in specific communities).

This concept have been finding numerous advocates, for example, within the educational community, who suggest it as an incremental assessment and study tool for both students and educators[Bar05]. Furthermore, following the mold of its analogue non-digital brother, it is also common as a job search and promotion tool. It can also be considered a mediator for self-reflection and evaluation.

Regarding the axis related to reflection, Barrett (2010)[Bar10] emphasizes that "Reflection is the 'heart and soul' of a portfolio, and is essential to brain-based learning".

Many researchers have been discussing and gradually evolving the concept. We highlight Rick Stiggins Helen C. Barrett. Rick Stiggins (1994)[Sti94] initially defined the e-Portfolios as "a collection of student work that demonstrates achievement or improvement (...) The material to be collected and the story to be told can vary greatly as a function of the assessment context".

Helen C. Barrett (2010)[Bar10] expanded this concept, considering that it is, simultaneously, a process and a product. A process is a series of events, involving time and effort, with the goal of producing a result. A product is the result generated by a process. The author greatly problematized the integration of e-Portfolios in education, considering that they both document learning (process) and results created during it (product), being an efficient tool at that. In this regard, the author sketched Figure 3, representing this

Several authors have contributing to this debate and enriching it around what e-Portfolio is, for example Lorenzo and Ittelson (2005)[LI05] define e-Portfolios as "a digitized collection of artifacts, including demonstrations resources, and accomplishments that represent an individual, group, community, organization, or institution". He concludes that this collection can be comprised of text-based, graphic, or multimedia elements archived on a Web site or on other electronic media such as CD-ROM or DVD.

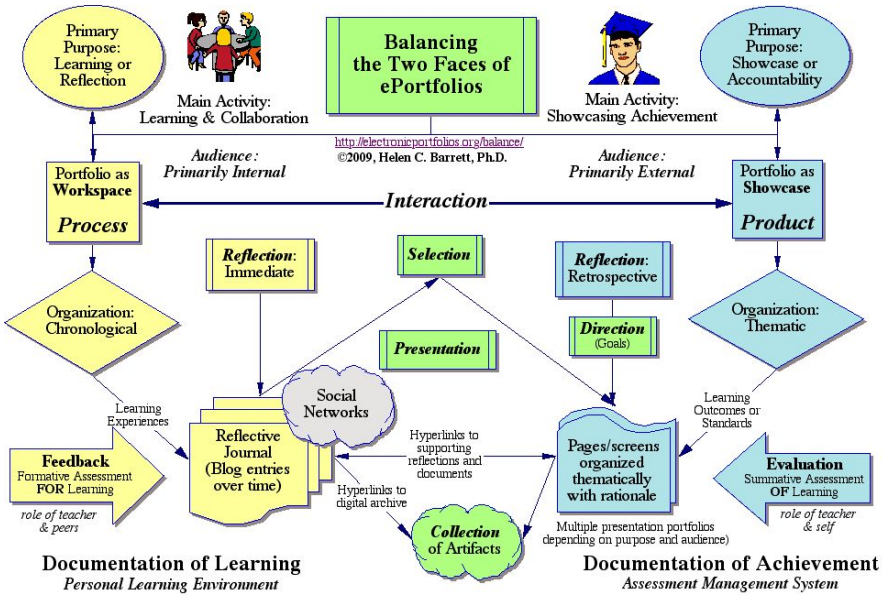

Figure 3: The two faces of e-Portfolios[Bar10]

Also Philippa Butler (2006)[But06] contributed to the debate by defining the term as "an electronic collection of evidence that shows your learning journey over time. Portfolios can report to specific academic fields or your lifelong learning. Evidence may include writing samples, photos, videos, research projects, observations by mentors and peers, and/or reflective thinking". He concludes that the key aspect of an eportfolio is our reflection on the evidence, such as why it was chosen and what we learned from the process of developing our e-Portfolio.

This debate and evolution of the concept is vast and continues to this day. In our approach to this subject, we condensed the definition of e-Portfolios as a compendium of a collection of resources (product), resulting from a process (learning), encouraging reflection.

The scientific community, in addition to to the theoretical debate, have been very actively developing projects, studies, applications and supporting material, for the expression, implementation and testing of e-Portfolios. An analysis of the main reference bibliography gives many interesting examples of this pragmatic approach, of concrete applications, some of the many examples include: Neal Sumner et al. (2008)[SF08], Jinshi Chen (2017)[Che17], Hafizan Som (2019)[SMG19], Omar Mahasneh (2020)[Mah20], Magdaleen Swardt et al. (2019)[DSJVPM19], Paul Stephensen (2010)[Ste10], Michel Arnaud (2006)[Arn06], Andreas Schmidbauer (2013)[Sch13], etc.

But the concept have not found yet a common ground and the majority of these tools and models were not very successful or popular amongst generic users. The impact of the traditional Curriculum Vitae have been decreasing[TdSDAKT20] and the demand for new approaches like e-Portfolios have been growing (see survey on chapter 4). But the formula of e-Portfolios has not found yet mainstream success or a common platform of expressiveness, unlike its relatives focused on the static old CV, for instance LinkedIn or Europass. This projects aims at solving this gap, presenting a product that condenses the e-Portfolios methodology and theory into a mainstream accessible product.

2.1.2 *Education & Job Market*

Traditionally, Curriculum Vitae has been the main communication tool between employers and the individuals who seek them out. The term derives from Latin and means "life path". We can define a CV as a list of temporal milestones, mainly academic and professional, achieved during an individual's life, for example achieved outcomes related with work, education or personal life: "I attended this course", "I worked in that company", "I won this prize". In other words, more synthetically, a CV documents goals achieved by an individual along a timeline.

e-Portfolios expand on this concept by adding products (artifacts) and a reflection process upon the timeline milestones format. In fact, simple milestones say little about an individual skills: they may had many positions/-experiences (several courses, several jobs, etc.) but the quantity does not necesserily translates to quality, for instance, not every graduated student is automatically competent in his field of study. An e-Portfolio, by incrementing the individuals path with pragmatic results (something not clear with a milestones system), introduces a qualitative factor to the process, resulting on a better self and external reflection on the individual's path. While CV focuses on promises ("completed this degree, for that reason I promise to be able to do that"), e-Portfolios focus on being ("Was able to create this, for that reason will be able to recreate it for you").

Increasingly attention has been paid to this nuance between both concepts and various platforms and entities have been changing the focus of their approach in that regard. We can find various examples of integration of portfolios/e-Portfolios mainly in the context of academia (various projects where cited in the last section), but also in the context of the job market where more attention is being given to soft/hard skills. In Portugal, for example, we can find examples of its application (in more simpler terms) in an hybrid context of learning and work in medicine graduates attending the specialization phase, where the use of (e)portfolios is very common (many academias ask them to compile a portfolio during specialization years and delivery/present it at the end for evaluation) - it serves at the same time as an interesting auto-reflexion tool but also as an evaluation tool.

Love et al. (2004)[LMG04] highlights the impact of e-Portfolios as "the most significant effect on education since the introduction of formal schooling". In fact, there are several education paradigms that could benefit from e-Portfolios and try to integrate them, we underline for example Nguyen and Ikeda (2014)[NI14] about Self-regulated learning (SLR) and Rezgu et al. (2017)[RMG17a] about Competency based approach (CBA).

However, Tzeng (2010)[Tze11] argues that there is a logistical barrier for implementation, both financially, with regard to the difficulties of democratic and equal implementation in all educational establishments, as well as the level of friction to adhesion and receptivity of students and educators to a paradigm shift, which may make it difficult to generalize this type of approach in the education sector. While barriers exist whenever paradigm shifts arise, we would argue that technological illiteracy rather than a barrier should be considered a target to be tackled and solved in the context of a modern world, not an insurmountable obstacle to the proliferation of these types of approaches.

As for approaches focused on the labor market instead, an interesting example is the European Union itself (EU) which has been working towards integrating e-Portfolios into the *Euro Pass* platform. On the official page we can read[Com21a] that in 2018 the process was started and in the fall of 2019 the first tests began and criteria

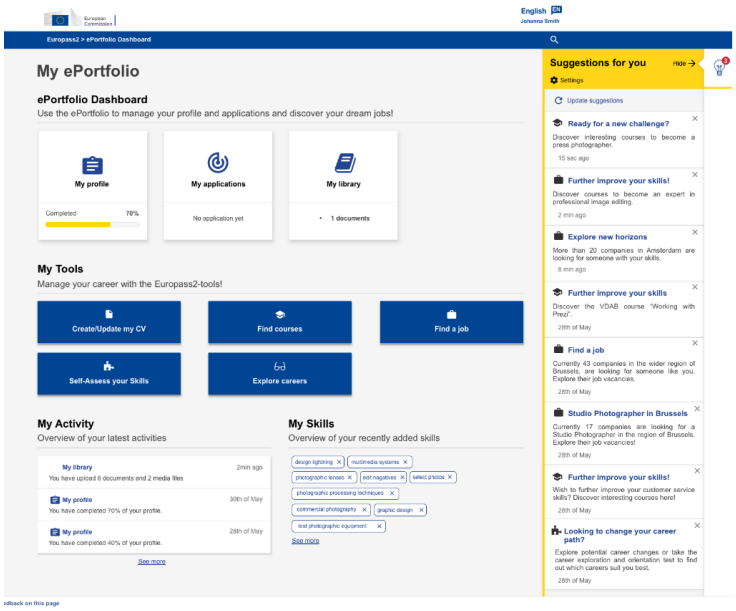

Figure 4: Mockup of the main interface of the EU approach to e-Portfolios[Com21b]

were established. In the document made public in 2018[Com21b] we can read a detailed description of what is intended for the future platform and an outline of what the main panels layout would be (see Figure 4).

Currently we can find attempts to integrate the concept on the *EuroPass* page, whose platform has been re-modeled and in a transition stage, but the process is still incomplete. *EuroPass* has been traditionally associated with the Curriculum Vitae paradigm, initially with great success, but it has gone out of flavour and it is slowly trying to integrate and transition to the e-Portfolio format. An initial attempt by EU to integrate this e-Portfolio paradigm can be verified at the current main website of *EuroPass*, where both the context of education and the labor market are interconnected with the concept of community and e-Portfolio strategies (in the documents of its future platform, there are mockup interfaces that try to offer job proposals and place the individual and the companies in contact via the e-Portfolio, while at the same time sharing it with a community).

Besides this approach conducted by the EU towards e-Portfolios, we also underline their work towards ontologies, Knowledge representation and semantic methodologies on this same area (job market and Curriculum Vitae), mainly through the project European Skills, Competences, qualifications and Occupations Ontology (ESCO)[Com17, Com, Eura, Eur21, Eurb], in which a classification system has been developed for describing, in the format of an open-source standardized ontology, skills, competences, qualifications and occupations in the context of the European Union (one should also note that it is made up of dozens of languages common to Europe, so it is also an interesting translation/localization tool, besides being an incredible standardization effort for establishing a common language in the context of the job market and education diplomas). BEsides being an open-source effort, it supports the European Union inner structures and projects related with this thematic[Com] (see Figure 5). In one of the most recent reports regarding this tool[Eur21], one can read as a summary that it "is currently employed by public and private implementers as the reference language for employment and education,

being widely recognised as the dictionary for the European labour market". In the State of the Art section related with Semantic Web (2.3.1, and it the first chapter related with the theoretical approaches of this project (4) one can read an extension of this subject.

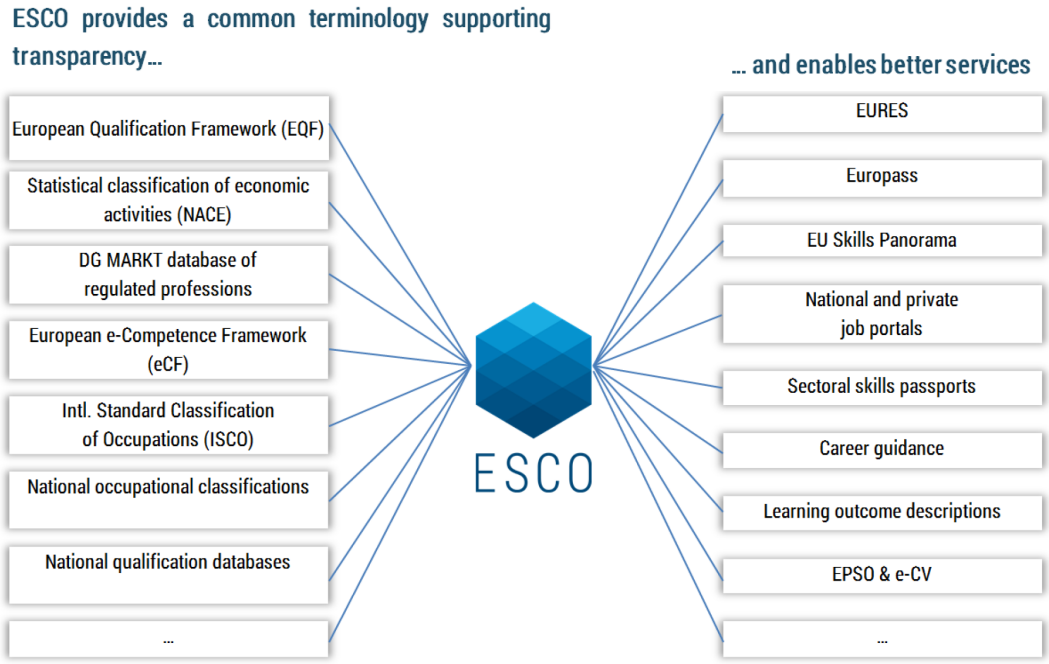

Figure 5: ESCO common terminology[Com]

Similarly, several modeling exercises have been tried to structure the e-Portfolios in a structured data representation, including ontologies, but none has become mainstream. In particular, we highlight several attempts that have been tried in the context of ontologies, for example Nguyen and Ikeda (2014)[NI14], Rezgui et al. (2017)[RMG17b], Shouhong Wang (2019)[Wan09], Hornung-Prähauser et al. (2005)[HPBB05], Rezgui et al. (2018)[RMSG18] (etc.), just to name a few. In the context of the current project, we studied these past approaches and conceptualizations before redefining our own.

This project intends to offer an approach to e-Portfolios more focused on the labor market than on education, but we consider that both fields will always be interconnected, because any both complement each other. In particular, we intend to expand and generalize the concept by integrating it into a social network context, integrating past methodologies into a simplified more user-friendly context target for the general public (we concluded that the current approaches failed to go mainstream because they aimed over-engineered something that should be by its nature more intuitive). Regarding labor market approaches, it should be noted that *LinkedIn* is the most popular social networking platform that exists at the moment, for that reason they are the main model we are competing against, but they have an approach only centered on the traditional Curriculum Vitae (CV) - we intend to create an alternative enriched product incremented by the e-Portfolios methodologies.

2.2 MICROSERVICES

In the following sections, based on literature review, we'll summarize the origins of the concept of Microsservices, and relevant concepts and technologies related to their application. Microservices are the main architecture for the implementation of the current project.

2.2.1 *From Monoliths to Microservices*

The term microservices was first discussed in Venice in May 2011 at an event between Software Architects, to describe "what the participants saw as a common architectural style that many of them had been recently exploring"[FL14]. In 2012 the same working group made the term "micrservices" official. This term does not have a sole complete definition, it has been expanded since then by several researchers such as James (2012)[Lew12], Fred George (2012)[Geo12], (etc.) and have been evolving. It is a emergent and expanding new approach to software architecture, even though it has been solidifying and gaining mainstream acceptance and an increasingly common ground, it is contemporary and for that reason a work in progress. For Johannes Thönes (2015)[Thö15], for example, a microservice is "a small application that can be deployed independently, scaled independently, and tested independently and that has a single responsibility".

This paradigm is an evolution of Service Oriented Architecture (SOA) methodologies, in which features and functions of an application are broken down into smaller services (microsservices), with a single responsibility and total autonomy, but retaining the app full functionality when working as whole in communication with each other, composing a decentralized model.

The concept refers to a particular type of approach to software development, focused on an architecture that promotes product segmentation and modularity into its basic and individual components, with totally independent execution, promoting the development and operation of these gears in isolation from each other. This segmentation facilitates the division of tasks between different teams; provides complete independence and autonomy regarding the technologies to be adopted by different components, since its execution is independent; facilitates the debugging process since execution is circumscribed and delimited to specific and smaller environments, etc. On the other hand, with regard to the disadvantages, it introduces a considerable increase in the complexity of the systems and the necessary resources (division by components implies an increase in complexity in execution environments), in addition to introducing new challenges and difficulties that did not exist before (for example, at the level of communication between different components or deployment exponentially more difficult due to a single application now breaking down into tens, hundreds or thousands of individualized sub-applications with particular requirements and different technologies).

Microservices Architecture can be considered the opposite of Monolithic Applications (a term adopted by the *UNIX* community to describe systems that become too large[Ray03]), in which the product is concentrated in a single development platform with responsibility for all functions. In conceptions based on microservices, the division of responsibilities is limited and reduced to the minimum common denominators, with well-defined limits, with the components being compartmentalized according to this criterion, promoting a decentralization

of data and a single responsibility per component (for example, one microsservice per database responsibility is a common approach). According to Thönes (2015)[Thö15] "An example of a single thing might be a single responsibility in terms of a functional requirement, or it might be in terms of a nonfunctional requirement or, as we've started talking about them, cross-functional requirements".

This is not an unprecedented approach, in fact the industrial revolution has long before introduced the concept of the assembly line, with production segmentation into isolated and independent components and functions. This approach has revolutionized many industries since Henry Ford popularized it in the 19th century. The concept of microservices can be seen, therefore, as nothing more than an adaptation of the industrial assembly line concept to the context of software development. In recent years we have seen a huge demand for this type of systems due to issues of high availability, scalability and considerable increase in the complexity of systems (mainly the distributed ones) - companies that face these challenges have found notable advantages in adhering to this type of architecture. Traditional monolithic applications continue to have their niche when systems do not face the aforementioned challenges and have more humble requirements, but in the context of distributed applications with exposure to high load volumes, concurrency and complexity the simplicity of the systems monoliths fail to compensate.

It should be noted that the entity Cloud Native Interactive Landscape (CNCF)[CNC20] is a recent project that promotes the architecture based on microservices and related web technologies, offering in its website an updated map with all existing tools and promoting new ones[2].

This type of architecture was popularized by *Netflix*, which was one of the main drivers through several articles and communications promoted by members of its development team, with Thönes (2015)[Thö15] being one of the pioneers in this broadcast.

After analysing several sources and reference bibliography, we compiled a timeline of the main milestones in the evolution of the microservice concept (see Figure 6).

The concept of microservice was an adaptation and progressive evolution of several paradigms, contextualized to particular needs and challenges introduced over time. *Web 2.0*, for example, has faced exponentially increased challenges in terms of high availability and scalability - now, in a digital age with mass access of the world's population to internet and platforms, never before have computer systems been exposed to such a large amount of accesses per second and heavy data exchange, the era of Big Data introduced immense challenges. This exposure and incremental evolution of challenges and systems led to the milestones exposed in the previously mentioned Figure 6. Figure 7 is an interesting summary of the evolution of this concept in the context of distributed systems methodologies.

Since the first official definition of the term microservice in 2011, several new challenges have emerged and with them new responses and concepts. Containers, for example, were popularized in response to the difficulties of isolating and deployment of the various components of a system based on a microservices architecture (both technologies are independent, but their popularization and evolution is strongly linked to that need). The same applies to orchestration systems (Kubernets and Docker Swarm, for example), which came to be to respond to mass containers management needs in a distributed environment, etc. These and other concepts (Service

2 CNCF microservices landscape map: https://landscape.cncf.io/

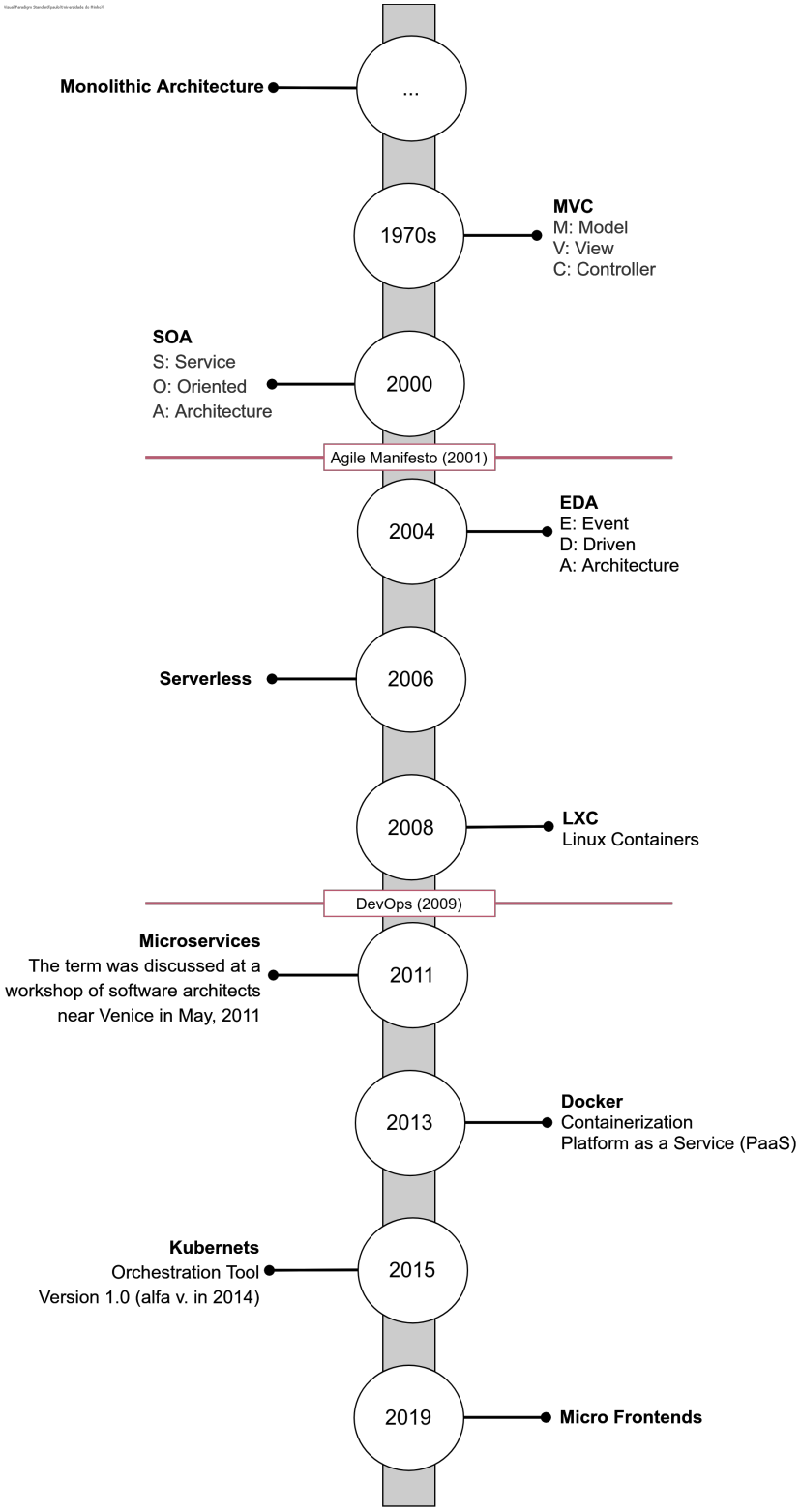

Figure 6: Timeline of the evolution of the concept of microservice

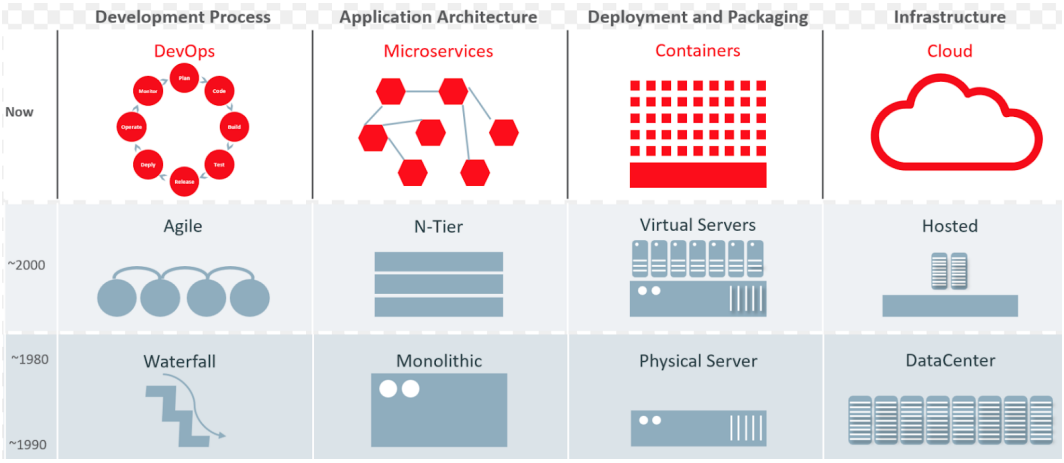

Figure 7: Evolution of distributed systems paradigms[Bel19]

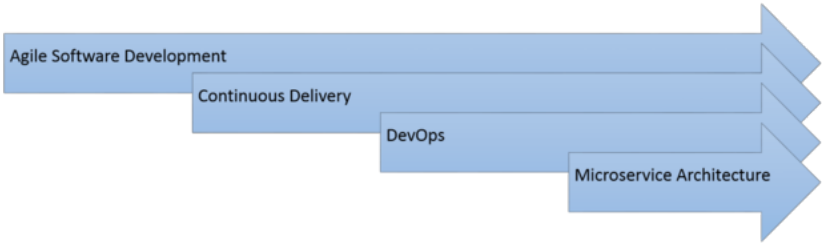

Figure 8: Progression and integration of microservices in agile development[McL16]

Meshes arised to help on microsservices discoverability and management; Message Brokers evolved to help on services async communication, etc.) will be covered very briefly in the following sections.

Very briefly, it is also worth mentioning that there is a relationship between Agile Development and Microservices Architectures. Matt McLarty (2016)[McL16], integrates the emergence of microservices in a natural evolution of the Agile Development of *Software* (Agile Software Development (AGILE)): "Microservices are the architectural phase of the agile progression". Figure 8, taken from this same source, summarizes this progression. Microsservices can be considered therefore a natural evolution in Agile Development.

On the current project, microsservices architecture and containerization where deeply used in order to allow high availability and scalability in the context of very high traffic. In the following sections we will expand on these architecture and emergent tools that support it.

2.2.2 *Containers & Orchestration*

Containers avoid the common exclamation "but on my machine it worked!". Between different development teams, in particular in a microsservice context, it is common to develop components separately initially, later integrating for debugging and deployment. This separation of responsibilities makes the management of work, time and resources more efficient. However, the scenario in which everyone develops on systems with exactly the same environments is rare - one of the biggest advantages of microsservices is the autonomy and independence of decision, allowing teams to select the best technological stack for their task, without depending on the decisions of other team members. But with great power comes great responsibility, this technological heterogeneity introduces added complexity: all it takes is a different version of the operation system, a different driver that introduces a bug, a missing library, or even a different version of the programming language used or configuration (etc.) so that when transposing the product to run on another machine, being it for integration tests or deployment, it fails or, even if working, produces unusual results or errors not seen so far. In the monolithic context one needed to replicate only one tech stack and environment, but now, with dozens or thousands of different microssrvices using different tech stacks each, all the team members need to replicate dozens of system/tech environments in order to be able to run, test or deploy the complete product. Containerization solves this problem, by encapsulating each service in a controlled and replicable environment[Sar20].

The goal of containers is to standardize the product development and production environment. This technology, at least in the format closer to what we know today, originated in 2008 with the Linux Container (LXC)s. Docker, based on this technology, has expanded and popularized the concept (there are several competitors and alternatives today, but Docker is clearly the most popular). However, the origin of this type of technology can be traced back to 1979, with the *Unix V7* and the introduction of the *chroot system*, which made it possible to change the privileges of processes regarding the directory and thus isolate its execution[RWC+15, Sar20, Jan18].

The history of microservices is based on a progression of recursive new needs, their resolution, and new needs introduced by them.

The various components produced in a single system with Microservices Architecture stimulated the need for deployment facilities in standardized ways with isolated environments, thus emerging containers as an answer. At the other hand, the high volume of containers now in use, introduced new challenges: the need for systems to manage them. In this context emerged orchestration systems such as *Kubernets* (the technology implemented in the current project) or *Docker Swarm*. For instance: to run a single product it was sometimes necessary to connect dozens or hundreds of services and subsystems divided into components - activating and maintaining one by one is time consuming, and some could crash during execution and amidst so many the administration would need tools to monitor that; to complicate the equation these components are not usually all on the same machine, being distributed by several in different networks and replicated in order to scale or help in high availability, complicating the management process. Tools like Kubernets were developed to solve this kind of necessities, facilitating development and maintenance of microsservices distributed architectures[Say17, BBH19, BBH18, Luk17].

Brendan Burns et at. (2019)[BBH19], started their book "Kubernetes: up and running: dive into the future of infrastructure" by making a delicious ironical dedicatory, that gives insight into this thematic:

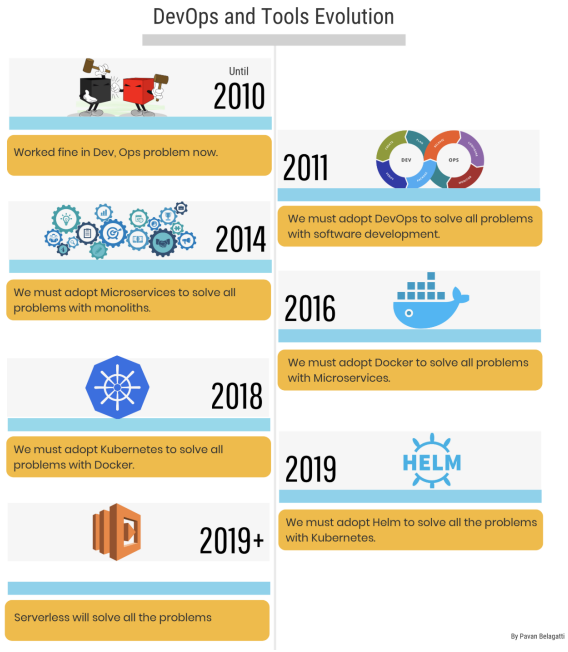

Figure 9: Evolution of Microservices in the context of DevOps[Bel19]

> "Kubernetes would like to thank every sysadmin who has woken up at 3 a.m. to restart a process. Every developer who pushed code to production only to find it didn't run like it did on their laptop. Every systems architect who mistakenly pointed a load test at the production service because of a leftover hostname that they hadn't updated. It was the pain, the weird hours, and the weird errors that inspired the development of Kubernetes. In a single sentence: Kubernetes intends to radically simplify the task of building, deploying, and maintaining distributed systems".[BBH19]

Figure 9 is illustrative of this evolution of necessities and solutions in the context of *DevOps*. For contextualization of the figure, Helm Charts is a package manager for Kubernetes, and Serverless is a ramification of the microsservices paradigm in which the architecture is more focused on events and the microsservices are sleeper functions that wake up on call (this results in better cost management, but also in many new limitations related to cold starts and latency of the functions - AWS popularized this concept, but it is still in its infancy limited to particular contexts). The atual web landscape is evolving very fast, since 2019 (last milestone on the figure) many new paradigms and experiments have been emerging like Micro Frontends, WASM containerization, etc.

In this same context, one should mention that different methodologies and best practices paradigms have been suggested, due to the complexity and immense variation of practises that have been arising from the new approaches introduced by microsservices architectures and variety of cloud dedicated tools (from Orchestration Tools interaction to telemetry and logging practices, or how should services communicate, etc.), with the objective of guiding development into good practices. One of the earliest and most popular best practices methodologies is the "Twelve-factor App", introduced in 2011 by Adam Wiggins[Wig21] (and drafted by developers of the Heroku platform) and included in many products architecture. Since then the technology panorama has evolved, includ-

ing the microsservices architecture itself, and this methodology no longer reflects completely all the needs of a 2021 developer (even though it is still relevant)[Hof16], for that reason many variations and alternatives have emerged, from the "3factor App", proposed by Hasura[3fa20] (suggestion the use of GraphQL based APIs, events for communication and async sleeveless functions), or the "Ten-Factor App" methodology suggested by Peltotalo (2021)[Pel21], adapting the the later methodology to a modern panorama (with a focus on API, Telemetry and Automation). In the current project, we followed a mix of these three best practices guidelines. The original "12Factor App" guidelines can be summarized, according to Reselman (2021)[Res21] (he did an amazing work of illustrating each factor with diagrams), in the following manner:

1. Codebase: One codebase tracked in revision control, many deploys.
2. Dependencies: Explicitly declare and isolate dependencies.
3. Config: Store config in the environment.
4. Backing Services: Treat backing services as attached resources.
5. Build, Release, Run: Strictly separate build and run stages.
6. Processes: Execute the app as one or more stateless processes.
7. Port Binding: Export services via port binding.
8. Concurrency: Scale-out via the process mode.
9. Disposability: Maximize robustness with fast startup and graceful shutdown.
10. Dev/Prod Parity: Keep development, staging, and production as similar as possible.
11. Logs: Treat logs as event streams.
12. Admin Processes: Run admin/management tasks as one-off processes.

One should also note that various types of tools abstract or build upon the Kubernetes technology since then, orchestration tools influenced a lot the microsservices architecture. We explored some of them in the implementation of this project, from telemtry, to communication, but regarding K8s we'll focus mainly *Rancher* and *Microk8s*, both being abstractions and interfaces that help in the management of a Kubernetes clusters. In chapter 5 we will further expand upon them.

New needs and solutions followed during this process, related to this same thematic of evolution of the concept, introducing new concepts and tools. This paradigm is still evolving today, new tools and methodologies are emerging from this collective brainstorm. We will discuss some of them in the following sections, focusing the ones that we included in the development of the current project.

2.2.3 *Service Meshes*

With the emergence of orchestration services, new challenges arose: the layers of abstraction under the components made it difficult to understand what was happening within the system under the hood, amidst dozens/thousands of replicated services distributed in a cluster of dozens of servers, so it was necessary to introduce monitoring and discoverability tools interconnected with those of orchestration, in order to better understand what happened during the execution, log the state and traffic of each service, what was their path, nodes and communication networks throughout the services distribution, to understand how the system components are related and how were they behaving - thus came into existence the Service Meshs, in order to solve these necessities[LLG+19, EMZ19, CB+20, Cal20]. This technology is therefore a kind of proxy in the communication between microservices, logging and helping keeping track of the sate and location of each service in the cluster network.

According to Charles Pretzer (2021)[Pre21] "a service mesh is an abstraction layer that binds together microservices-based distributed systems by providing observability, security, and reliability".

Service Meshs also started to function as a kind of API Gateway (in the sense of discoverability of the services within the network) in some contexts, because the way of communicating also changed with the introduction of this type of systems - with systems distributed over different containers, in machines and different networks, often replicated to ensure high availability, it was useful to have an aggregator point that ensured the path of each component in the network was monitored and abstracted, each address and node in the internal intricacies of the system, even when these were restarted or changed during execution one could communicate to a service and the orchestration tools would take care of selecting the replica with and distribute the load. Service Meshs help providing this coherence, offering real-time information on the state and address of each component and the flow of the communication between them.

Kammer et. all (2021)[Kea21] compiled an excellent source of information about this type of technology and what are the advantages and disadvantages of each technological offer on the market at the moment. Figure 10 is taken from this reference, summarizing the context of Service Meshs in microservices.

One of the most popular and used examples of a Service Mesh tool is *Istio*[CB19], sponsored by Google. It is multi-paradigm (not restricted to any particular type of orchestration tool), well-tested solution, but it is also one of the less intuitive ones, and least performative according to benchmarks.

At the other hand *Linkerd* is a competitor of *Istio*, another example of a Service Mesh tool that has grown enormously, being the second most popular at the moment. Its 1st version was one of the first to pioneer this type of technology, but it was obsolete. However, more recently, the tool was completely rewritten in the Rust language, having been released as a new rebranded tool totally independent and different from the 1st one - this new version is now the Service Mesh with the best performance on the market, having twice the performance of *Istio* according to benchmarks. One downside is that it only supports systems based on Kubernets orchestration technology, but it has been noted to be more intuitive, easy to deploy besides being the most performative one on the market at the moment[Pre21, KK20].

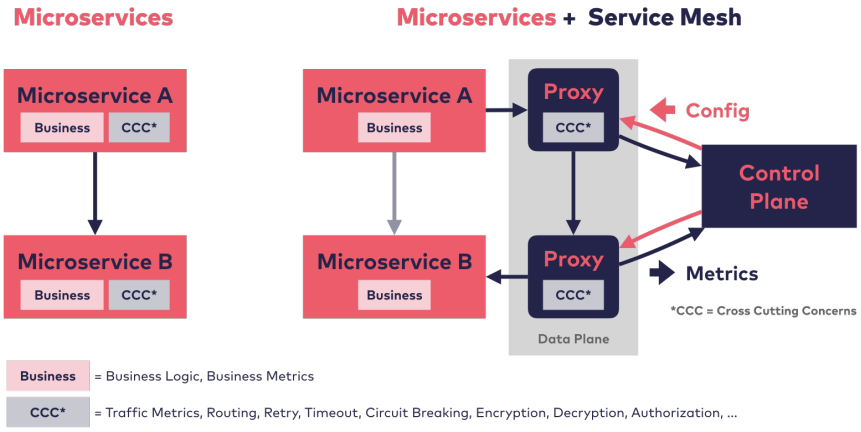

Figure 10: Service Meshs in the context of microservices[Kea21]

Even though *Istio* and *Linkerd* are the most popular choices, other ones exist that we will not focus[KK20]. In third place of popularity, for example, we could briefly mention *Consul*, that has been important historically in the evolution of this technology. We should also note that the popular API Gateway *Kong* (implemented during this project) is developing an extension of their tool with Service Mesh features called *Kuma*, and building a new ecossystem around them, however at the moment it is in beta phase and its use is not yet the most desirable in production in comparison with the other well-tested and established tools. But it is an interesting solution to take note.

In the context of the current project, and because it was the most performative options and we also wanted to give focus to Rust programming language and test it as an emergent force in the Web/Cloud panorama (in the following sections and implementation chapters we will expand upon it), *Linkerd* was the chosen Service Mesh for the current project implementation. We also included and mantained a traditional API Gateway layer as an entry point, in order to serve as an aggregator, authentication management and common entry point, and besides providing a traditional SWAGGER APIs RESTful interface, we also created and included a GraphQL based service for the API component, as suggested by the "3Factor App" best practices methodology[3fa20]. Also, we included reliable events as a communication bridge between the services, further explained in the next section "Service Broker".

2.2.4 *Message Brokers*

RESTful communication continues to be common in systems based on microservices, but synchronous communication between services is discouraged in many scenarios, due to creating performance bottlenecks, or loss of messages due to increased communication between services. Figure 11, from Nishaninl (2021)[Mic21] encourages asynchronous communication in this regard and showcases synchronous as an anti-pattern. Therefore,

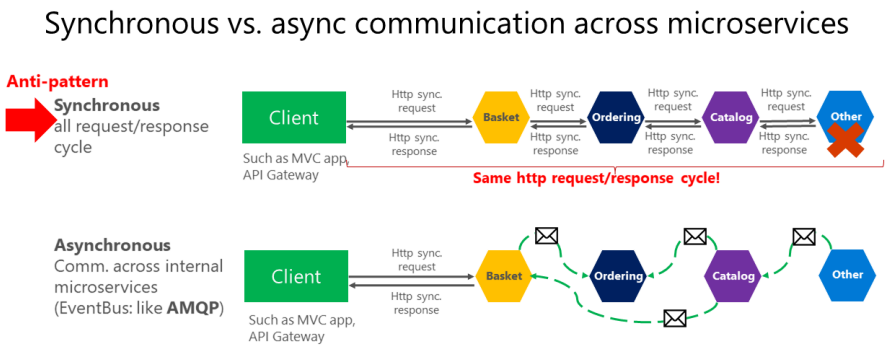

Figure 11: Good and bad approaches to communication between microservices[Mic21]

new forms were introduced and encouraged, namely Message Brokers (for example based on the AMQP protocol), serving the base for systems based on asynchronous events - communication would no longer be carried out directly only through endpoints and direct calls via HTTP RESTFull protocol, but rather by event queue managers, who served to manage asynchronously endpoints, functions or components in an orderly queue, managed by dedicated workers or event systems who manage the execution priority of each, ensuring that no requests are lost by unavailability of some component or failure, as they keep logs of each execution or repeat a certain number of times until it finishes.

In a system composed of hundreds of components, delays or failures in processing one in a sequential execution could mean interruptions throughout the system, for this reason systems based on asynchronous messages and event managers were an important evolution to ensure effective communication. Examples of related technologies include *RabbitMQ*, *Apache Kafka*, *Celery*, *Py-Rq*, etc. There are many protocols options, from MQTT to STOMP, but Advanced Message Queuing Protocol (AMQP) is one of the most popular common choice[Mic21, Miń18, Bib19] (it offers reliability and interoperability, being a good choice for reliable queuing, topic-based publish-and-subscribe messaging, flexible routing, transactions, security, etc.).

This type of technologies introduced new communication paradigms between different components of a system, promoting decoupling, decentralization and asynchronous structures. In Figure 12, from Minkowski (2018)[Miń18], we can see examples of these paradigm differences regarding the communication of the components - the synchronous protocols open up performance bottlenecks and scalability/reliability issues, but on the other hand a Message Broker based pattern (being it a queue system or a publish-subscribe model) highly increase a microsservice based architecture efficiency.

In the context of the current project, we opted for asynchronous message systems based on AMQP protocols, implemented using *RabbitMQ* and redis based queuing systems (Celery and Py-Redis for instance), that received each request in a queue of messages, and workers distributed them to each microsservice asynchronously, better managing loads or failure, repeating and logging each failure in a controlled way or activating replication mechanisms through the Kunernetes pods if needed. This approache allowed us to better manage communication with high availability, reliability and scalability, allowing us to better manage traffic and load balancing for each request, monitoring each queue with telemetry, replication and repetition in case of failure.

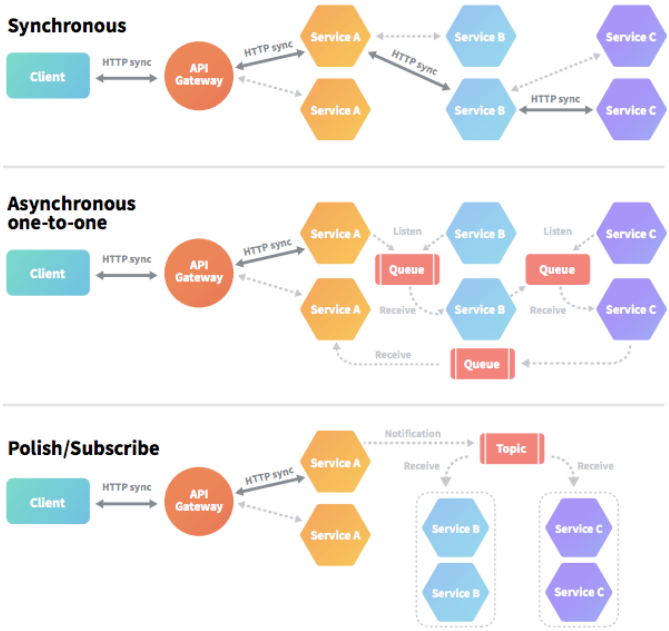

Figure 12: Different communication patterns between components[Miń18]

2.3 EMERGING WEB TECHNOLOGIES

In the following sections, based on literature review, we'll summarize relevant emergent technologies in the Web Development landscape, recent but with great impact for its future, from Semantic Web, to Rust, Micro Frontends and WebAssembly. These technologies were exploratory implemented during the current project.

2.3.1 *Web Semantic & Knowledge Graphs - From Web 1.0 to Web 3.0*

Tim Berners-Lee, director of the World Wide Web Consortium, established the term Semantic Web and promotes the idea of converting the Web into a big collection of semantic databases: "People keep asking what Web 3.0 is. I think maybe when you've got an overlay of scalable vector graphics - everything rippling and folding and looking misty-on Web 2.0 and access to a semantic Web integrated across a huge space of data, you'll have access to an unbelievable data resource" [Sha06].

"Web 1.0 started as a Read only medium; the next version Web 2.0 established itself as a Read/Write medium. Now the currently evolving version of web, Web 3.0, is said to be a technologically advanced medium which allows the users to Read/Write/Execute and also allows the machines to carry out some of the thinking, so far, expected only from the humans" [RL11], introducing the concept of Semantic Web and Linked Open Data. One of the main technologies supporting this vision of the Web 3.0 is ontologies. Semantics, with origin in ancient Greece, is the study of meaning, reference, or truth. It is a subfield of Philosophy, Linguistics, Neuroscience and Computer Science. We believe it is the key for advancing the human-machine interaction.

This new vision for the Web reduces the man-machine gap, by giving semantic and structural meaning to complex Big Data. SPARQL, RDF, and OWL (expressed, amongst other ways, through the Turtle format), are central technologies for this stage, both recommended by W3C for implementing Web Ontologies, trying to revolutionize the way we interact with the Web, data and computers. Because of these characteristics, ontologies are one of the key technologies for the implementation of the new generation of the World Wide Web.

Ontologies, the backbone of the Semantic Web 3.0, which contain the vocabulary, semantic relationships, and simple rules of inference and logic for a specific domain, are accessed by software agents. These agents locate and combine data from many sources to deliver meaningful information to the user[Mor11].

An ontology is a "specification of a conceptualization", "a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents"[Gru09]. Essentially, in a broad sense, it is a type of graph database based on triplestores for each fact (Subject, Predicate, Object), built around a semantic representation of the relationships between them. Guarino et al. (2009)[GOS09] expanded on what is a conceptualization and the history of the formal definition of ontologies, suggesting more precise ones.

Knowledge Graphs are clusters of ontologies with common ramifications and ways to efficiently interact with them. They are important because "are critical to many enterprises: they provide the structured data and factual knowledge that drive many products and make them more intelligent and magical"[NGJ+19].

Google popularized the concept of Knowledge Graph with the launch of their project in 2012, aimed at improving their search engine feedback[Sin12]. Since then, many projects have been trying to translate Big Data into comprehensive data through similar methodologies, for example the ones related with the Linked Open Data initiative, WordNet, YAGO[PTWS20], DBPedia, etc. These approaches are expanding the concept of what we understand to be the World Wide Web, building the basis of the Web 3.0, also known as the Semantic Web.

This project gets inspiration from the mentioned ones, implementing personalized methodologies based upon entities identification: a graph of articles and metadata, expanded with relationships of entities identified inside them. It explores this concept of Semantic Web, by developing tools target at building a web of relational semantic data related to a subject, instead of a traditional web of documents. These approach served as a tool to build better search interfaces, enriched with semantic data. We ended up with ontologies, dynamically generated, composing a Knowledge Graph of exploitable data.

Besides the development of new tools (EntiGraph) and active integration of graph databases and methodologies supporting OWL and SPARQL, we also integrated already developed standards and ontological datasets. Mainly, we integrated the European Skills, Competences, qualifications and Occupations Ontology (ESCO) project.

ESCO - European Skills, Competences, qualifications and Occupations Ontology

The European Union release an ontology aimed at standardizing skills, competences, qualifications and occupations/jobs in its continent, including 27 languages. Figure 13 quantifies this result.

ESCO was built as an extension of the International Standard Classification of Occupations (ISCO)[Eurb] - Figure 14 showcases this integration. "The labor market is a system that is complex and difficult to manage. To

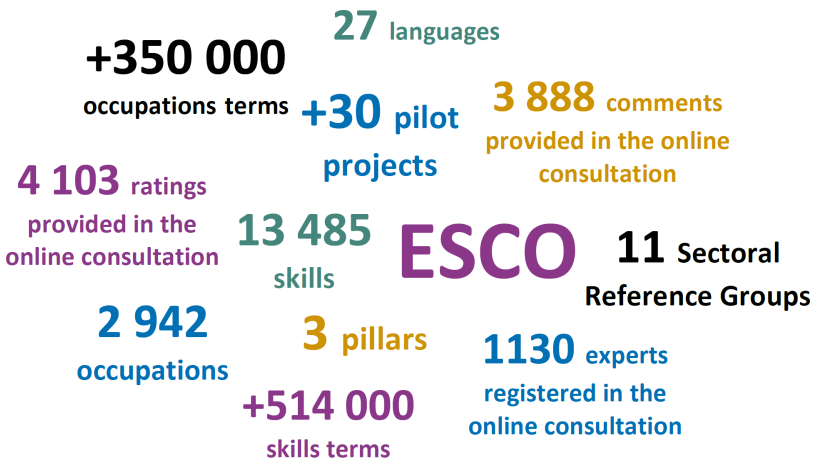

Figure 13: ESCO in numbers

overcome this challenge, the European Union has launched the ESCO project which is a language that aims to describe it"[Kah20], Kahlawi (2020).

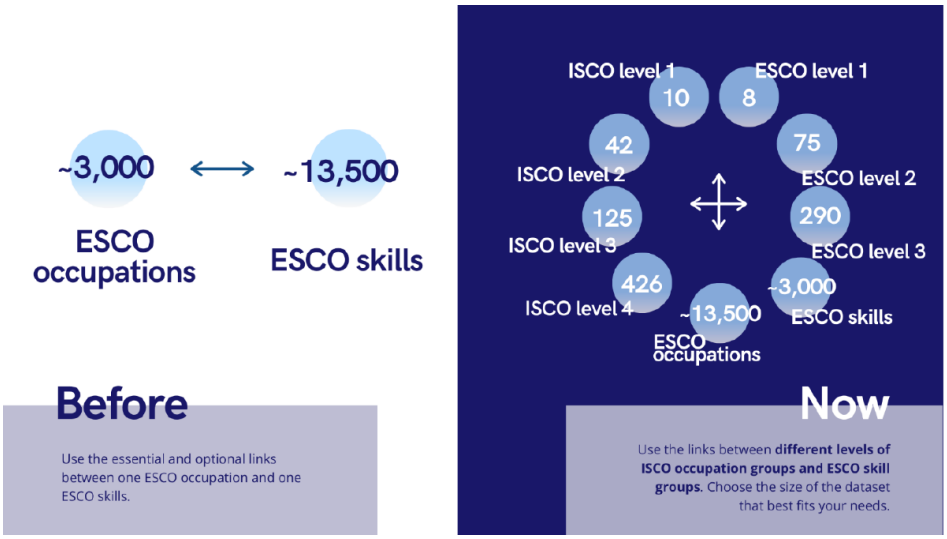

Figure 14: ESCO and ISCO integration[Eur21]

ESCO could be, for example, integrated in employment services, to ease the process of classifying and managing employees, or to improve matching algorithms; integrate in education institutions for defining students career paths linked to study areas, or to conduct graduate tracking; it could also be integrated in research bodies to connect the ESCO classification with data gathered externally or to build surveys and investigate results, etc. The possibilities are unlimited.

In the context of the current work, we integrated this ontology as support for a better Semantic Web infrastructure, to classify and structure information, to guide users in the interfaces and search queries, or to integrate with

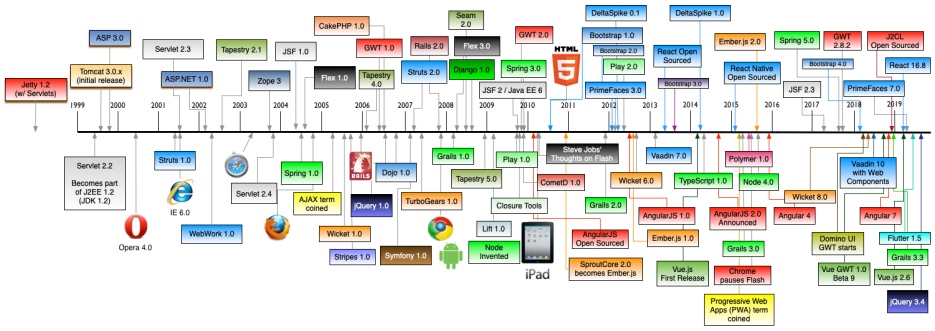

Figure 15: "A Brief History of Web Frameworks"[Rai19]

our inner Knowledge Graphs to create richer semantic ramifications. We downloaded this open-source ontology in RDF format, and integrate it using triplestore databases in our servers (Stardog, GraphDB and Oxigraph).

2.3.2 *Micro Frontends*

There are lots of frameworks for web development and every year new ones are born. It is a very rich ecosystem, but also highly competitive and complex, which has evolved at enormous speed, leaving many technologies obsolete to make way for new ones. For example, in Figure 15, we can see a timeline compiled by Matt Raible (2019)[Rai19] with some of the major web frameworks of the past decade.

Amidst such variety of tools, different developers have been specializing in different approaches, there are many paths that lead to the Rome of web development. This fact sometimes makes the composition and integration of teams difficult, because different members have different preferences regarding the solution of problems, and when a rigid course is chosen the heterogeneity of valences is a challenge. Nor is a tool always the right answer to all problems, sometimes projects opt for the least of evils, where a section of a tool would benefit more from the use of a certain technology, while for other secondary components of the same product other ones would be preferable, but for maintaining consistency between the development a common one is chosen.

An example: what if a development team, highly specialized in Vue, concluded that it would be more advantageous to use this technology for a certain isolated section of a product, being possible to achieve double the performance and development time of that sub-component in particular, but the entire frontend of the product was already initially integrated in a reactive ecosystem based on React or Angular making it impossible to mix both approaches? - currently this obstacle is about to no longer be a problem, as the concept of microservices has also migrated to the frontend, making it possible to develop individual portions of the same page in different technologies, allowing these to be compiled on a single page without clash, with autonomy and independent environment of execution, with individualized development teams and with freedom of choice regarding technology environment.

This innovative concept is called *Micro Frontends*. It has evolved through different approaches. Cam Jackson (2019)[Jac19] defines the *Micro Frontend* approach as "an architectural style where independently deliverable

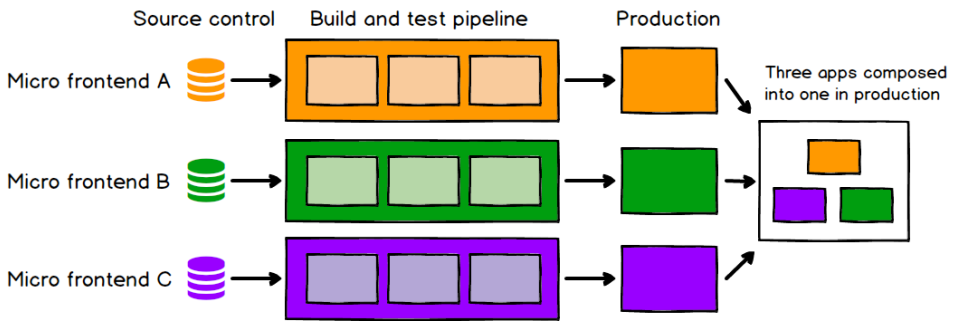

Figure 16: Micro Frontends autonomy: composition of different apps into one[Jac19]

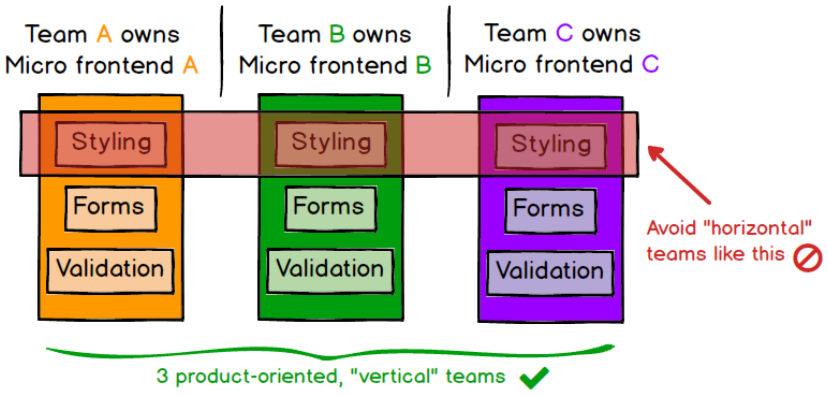

Figure 17: Micro Frontends: vertical vs horizontal teams for decentralized development[Jac19]

frontend applications are composed into a greater whole". Figure 16, from the same author, summarizes this concept.

Cam Jackson points out benefits regarding the integration of this methodology:

- Smaller, more cohesive and maintainable codebases.
- More scalable organisations with decoupled, autonomous teams. Figure 17 showcases this concept, and advocates a vertical independent development for teams instead of horizontal, focusing individualization of responsibilities in a decentralized environment.
- The ability to upgrade, update, or even rewrite parts of the frontend in a more incremental fashion than was previously possible.

However, after analysing bibliography[Gee17, Gee20, Jac19], we compiled some cons that are also important to take notice regarding this kind of emergent approach:

- In the context of frontends, when teams are completely independent of the technology to be used, it can be difficult to maintain a product's visual coherence .

- Application performance may be lower due to loading a larger number of tools (for example React and Angular on the same page doubles the import of additional libraries).
- In independent division of responsibilities between different teams makes it more difficult to manage the consistency of the SEO of the product.
- Product complexity can increase exponentially due to challenges introduced in the inter-communication of components (although new tools are being released to minimize this difficulty, it is still a challenged).
- If left unchecked, it can greatly increase the amount of data needed in the first loading of the website.

It is worth mentioning that new tools have been released with great frequency to minimize these problems and expand the concept of Micro Frontends, from communication challenges between services to encapsulation, the concept is very recent but in fast evolution.

One of the simplest initial ways of implementing this new concept was based on the use of iframes: the page was divided into individual components, and each one was embedded as a whole page in a borderless iframe, creating an illusion of an individual page, when in fact it had many sub-pages as its components. However, this approach is rudimentary and limited, as it introduces performance problems, code duplication, and added difficulties in managing page intercommunication, because instead of a single DOM we would now have several ones spread across different iframes in camouflage - changes in the state of a component would be difficult to share and manage at the global level of the page. This approach is still useful in simple contexts, but on a complex scale dependent on scalability it could face serious problems.

Another example of a popular hybrid alternative approach is, for example, *External App Bootstrapping* promoted by some reactive frameworks, where the components would be developed separately, to later be compiled into a single application and the inter-communication between them set via Window Object or Event Bus for example. This approach usually works at the level of a single framework for example Angularallows this compilation approach by individualized chunks, as if each chunk of the frontend were a kind of microservice, however it is remains limited and offers little freedom as it continues to restrict development with different technologies and methodologies.

Frameworks like *Single-SPA*[3], *Bit*[4] or the *Project Mosaic*[5] expanded the concept and introduced a real division of the responsibilities of the components, with execution in run-time as if they were individualized microservices with full independence. It is possible to have *Vue*, *React* and *Angular* running on the same page! Of course, the fascination of this possibility is only theoretical, in practice it would increase the consumption of resources, but the possibilities become more and more unlimited, as technologies such as *Webpack*[6], in particular version 5, has been improving this aspect and integrating with *Micro Frontend* technologies.

Currently, the most popular techniques applied to this concept have been based on Javascript, which load the components individually at the initial load of the page, or through Web Components (set of standards introduced

3 https://single-spa.js.org/
4 https://bit.dev/
5 https://www.mosaic9.org/
6 https://webpack.js.org /

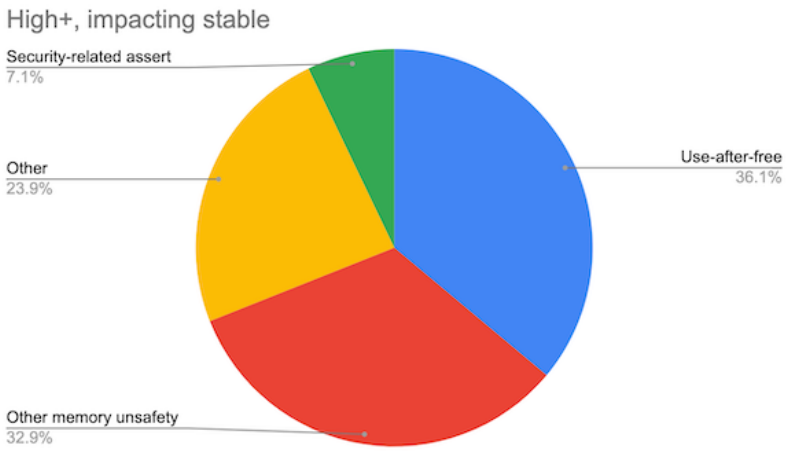

Figure 18: 70% of system vulnerabilities are memory management issues [Pro21]

by W3C[7], which split the DOM into a *Shadow DOM* and a *Light DOM*, allowing the execution of individual components through their manipulation). These emergent techniques are bringing the microsservices architecture to the frontend, even though still facing many challenges it is evolving very fast.

2.3.3 *Rust*

In this section the recent programming language *Rust* will be contextualized: the new paradigms it introduce and its symbiosis with WebAssembly and Cloud technologies.

Rust[8] is a very recent programming language that aims to be a modern alternative to *C/C++*. It is a low-level programming language that introduces very particular new paradigms, in particular with regard to the Ownership concept.

Its proposal is to solve a problem that intersects with this one disclosed by the browser Chromium development team:

> "Around 70% of our high severity security bugs are memory unsafety problems (that is, mistakes with C/C++ pointers). Half of those are use-after-free bugs." Source: "Chromium Project - Security Report"[Pro21][9]

In Figure 18 we can see a summary of this report.

In fact, this report is similar to several others, 70% of vulnerabilities in most systems have preventable causes with better memory management mechanisms. In this regard, the same source discussed ways to approach the problem, as they aim to reduce it, in short they compiled a comparative Figure (19) with an analysis of programming languages paradigms that could solve this problem, being Rust the best option.

7 https://www.w3.org/TR/components-intro/
8 https://www.rust-lang.org/
9 https://www.chromium.org/Home/chromium-security/memory-safety

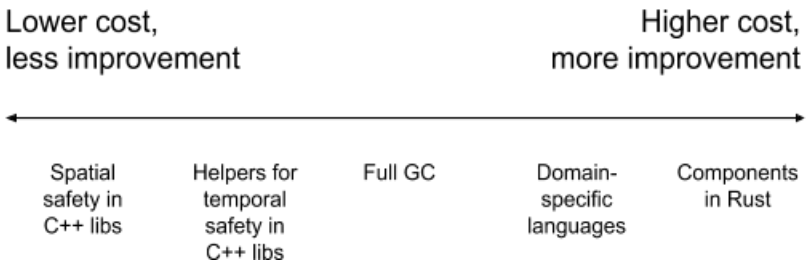

Figure 19: Options for solving memory vulnerabilities[Pro21]

Rust is a multiparadigm language designed to deal with this problem[BB20]. In fact, this is precisely the problem that this new language initially set out to solve, and it is on this objective that its entire philosophy is based. This is clearly displayed on the official website triple motto: "speed" (can be as fast or faster than *C*[10]), "safety" (automatically avoids 70% of the most common vulnerabilities in *C/C++* based systems) and "concurrency" (aims to solve modern scalability issues). The issue of security is revolutionized with a new approach to memory, introducing the Ownership model: "the ownership model is basically a memory management contract that Rust upholds in order to avoid memory leaks and null pointers"[Ort20]. Also Abhiram et al. (2017) explore this issue in the article "System Programming in Rust: Beyond Safety"[BBB+17] which can be consulted for further details. In this regard, we encourage also to see the Nichols (2019) communication[Nic19] that explores the security issues that Rust tries to solve[11]. Furthermore, the language has a very verbose compiler, with strict rules regarding the structuring of certain aspects, serving as brakes in certain scenarios of bad code architecture (it should be noted that the language has commands to completely ignore these verbose aspects of the compiler if needed, working in full bare metal without brakes).

This emerging programming language debuted as a stable release in 2015, released by Mozilla Firefox. From the beginning it had a context similar in some parts to *Haskell*, having sometimes been compared to it, in that it started early on as a high-level theoretical exercise, promising many (but Haskell fell short in popularity, and on the other hand Rust is steadily gaining it). But Rust didnt fell out of flavor like Haskell, because this theoretical exercise resulted in real solutions: Mozilla was facing the same type of difficulties enunciated by Chromium, while delepoing their new browser engine Servo, for that reason they wanted an alternative to *C/C++* without the 70% of vulnerabilities open door - that's how Rust was born as an answer and the Servo Engine was fully written with it and incorporated in Firefox [Kla18][12].

Since then several companies have adopted Rust for exactly the same reasons. For example: NPM, the NodeJs repository, was completely rewritten in Rust in an attempt to improve performance. According to official sources the mission was successful, not so much for the pure performance part (as it was already highly optimized), but more because it was possible to solve several obscure bugs and security issues during this process.

10 Source: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust.html
11 https://www.youtube.com/watch?v=A3AdN7U24iU
12 https://research.mozilla.org/servo-engines/

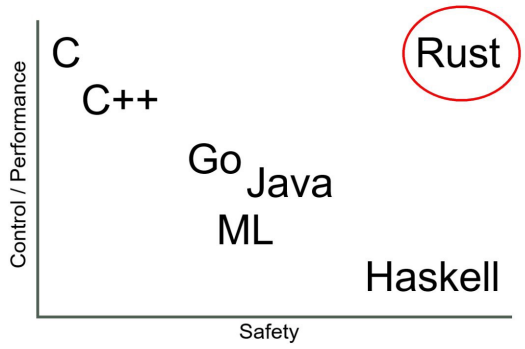

Figure 20: Options for solving memory vulnerabilities[Mat20]

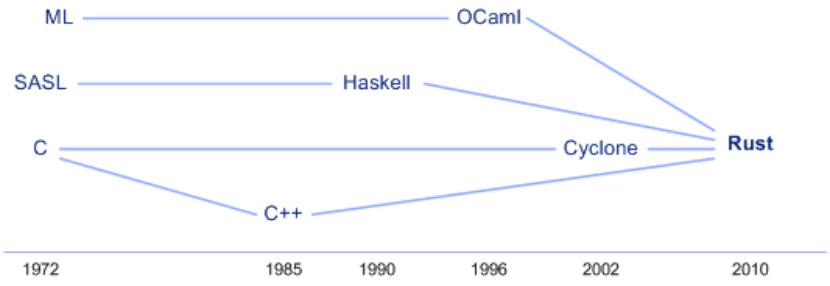

Figure 21: Origins of Rust Language[Jon18]

Also the creator of *NodeJS* fell in love with Rust and recreated a more modern alternative to his original framework in a project now called *Deno*[13] (note: the project is still very recent, in beta stage and an ecosystem still young, but it has a lot of potential and attention, the author wants it to be everything NodeJS couldn't be when he created it).

In a context of comparative framing of Rust with other programming languages, the Figure 20 is an example of its paradigm[Mat20].

The comparison with Haskell in the previous diagram is not unreasonable, in fact several authors consider it an ideological successor, as we can read in Chris Allen (2018)[All18] "Haskell and Rust have shared goals and design priorities".

In fact Rust is a multiparadigm language with different branches of origin, as one can read in M. Tim Jones (2018)[Jon18] "First, Rust is heavily influenced by Cyclone (a safe dialect of C and an imperative language), with some aspects of object-oriented features from C++. But, it also includes functional features from languages like Haskell and OCaml. The result is a C-like language that supports multiparadigm programming (imperative, functional, and object oriented)". The same author summarizes these origins Figure 21.

Despite all the attention surrounding the language (even Microsoft has recently created a development team focused on Rust) and galloping rises in the indexes of popularity, what makes this emerging technology most fit

13 https://deno.land/

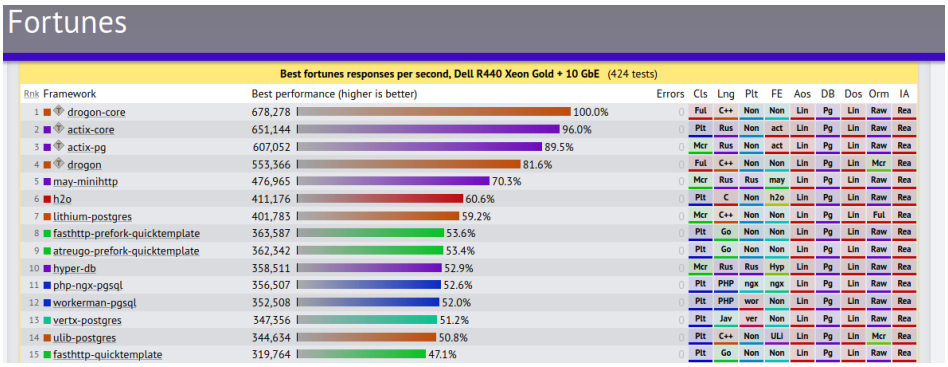

Figure 22: Benchmark: top 15 web frameworks with best performance according to Fortunes 2020 report

for this project is its web ecosystem, and the fact that it was one of the pioneers to adopt WebAssembly (incidentally, Mozilla herself was the founder of both projects) and because it is being adopted with huge adherence by Cloud infrastructures (for example, *AWS* itself has been promoting Rust)[Asa20]. It has therefore become a powerful weapon in the web development arsenal. Some examples of its versatility and integration in Cloud environment (and the Web ecosystem in general) will be summarized at the end of this section with examples of interesting emerging tools developed with Rust, target for Cloud and microservices environments. Many of them are supported by Cloud Native Interactive Landscape. Several other examples exist, in particular technologies that partially use *Rust* for specific components, not in their entirety (for example *Envoy*[14] among dozens of others).

For instance, Figure 22 showcases an interesting benchmark, where one of the Rust web frameworks (ActixWeb) has one of the highest performances in the market, as we can see in the image taken from *Fortunes* at the present date (2020) - note that 3 web frameworks based on Rust appear in the top 5 most performative, and 4 in the top 10.

Another interesting fact is that Rust also has several emerging reactive frameworks (frontend) based on WebAssembly that have stood out as being the fastest on the market in several benchmarks (for example *Sauron*, *Yew*, *Seed*, *Mogwai*, etc. - all ahead of React, Angular and Vue in terms of raw performance). In Figure 23 benchmark we can see that these Rust frameworks target for WASM are at the top in performance matters, above popular frameworks like React, Angular and Vue:[15]

Of course, the issue of performance is relative and of little interest in most cases when the results are already acceptable. This project does not focus on that aspect. These examples are just to attest to the potential that the interconnection of Rust with WebAssembly can bring to the web development industry, as it turns the browser into a low-level execution machine. In the present project Rust stands out as a tool due to its versatility, security, and in particular the fact that it allows compiling for almost all types of systems, including mobile, with reduced use of resources and space (it's a low-level language, the goal is not to replace current high-level production environments, slower in terms of performance but much faster in terms of development time, but rather to inter-

14 Website: https://www.envoyproxy.io/
15 Source: https://github.com/schell/mogwai

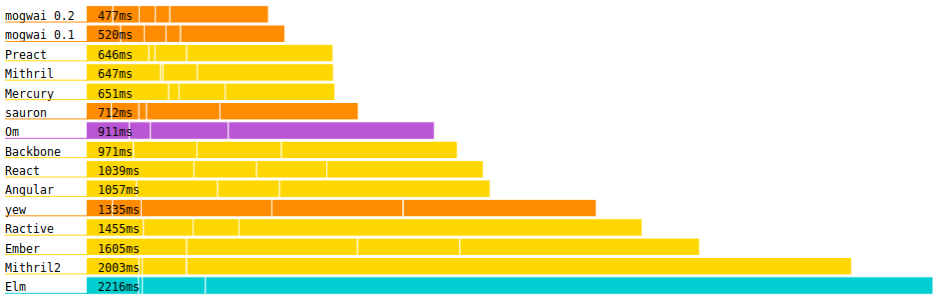

Figure 23: Benchmark of reactive frontend frameworks

connect and cover the flaws of those environments when necessary). See, for example, the communication by Amir Yasin (2017)[Yas17] who talks about his experience in improving the performance of Javascript applications with the integration of Rust, or the interesting lecture by Ashley Williams (2017)[Wil17] about how he managed to convince the world's biggest Package Manager (*NMP*) to adopt Rust in its restructuring.

The most interesting aspect of Rust for the current project, its Rust rich ecosystem of Web/Cloud tools, with a special focus for Semantic Web[VHP+21] and data tools, that we used for developing and exposing RDF and SPARQL based interfaces and open-source applications. Rust was also used in integration with WASM to increment performance in some aspects of his project. It was also the main programming language adopted for the mobile application that resulted from this project, that explored the integration of Rust in mobile systems through exploratory methodologies. These approaches are further discussed in the architectural and implementation chapters 5

During our research about Rust, we compiled some lists of interesting projects, collaborations and tools based on this emerging programming language. Next, we will summarize them.

For a language with only 5 years of existence, Rust as grown incredibly fast. Since release, it has won the "Most loved programming language" poll at Stack Overflow for 5 years in a row[rus21], and it was the most trending language in the TIOBE index in 2020, jumping the most positions in popularity for that year. Besides its own native ecosystem with increasingly great variety and quality for many areas, Rust has been used to improved existing popular tools and other companies code-bases in different ways, because it integrates very well with a variety of systems. Some examples of notable mediatic examples:

- **NPM:** it is the package manager for the Node JavaScript platform, and one of the most used heavy traffic in the world. Due to performance bottlenecks and bugs, its engineering team decided to entirely rewritten it using Rust with great success. A 2019 white paper[npm19] documents this process. Also, at the conference Rust Fest 2017, Ashley Williams NPM engineer, presented the paper "How I Convinced the World's Largest Package Manager to Use Rust, and So Can You"[Rus17], relating to this matter.

- **Next.js:**[Kri21] it is one of the most popular React based frameworks, that integrate it with Node.js server-side. Recently, in version 12, it was improved with a new rewritten Rust based compiler, improving greatly

its build time. This approach will certainly be replicated to other JS frameworks, because it has one of the best results right now, improving build times to 30% or more faster in many cases.

- **Microsoft:** many reports and teams have announced that Microsoft is adopting Rust both for their Azure infrastructure, and even to rewrite some Windows components. In some reports one can read "Microsoft determined that 70% of security patches pushed to computers are to fix memory-related bugs and believes that Rust would've been able to catch these bugs during the development phase"[Cae20].
- **Google:** recently we could read in many sources that besides bringing Rust support to the Android system[and21], Google's Android Team is encouraging the adopting of Rust for the Linux Kernel development[Tun21].
- **Redox:** Philipp Oppermann started the project of developing an operation system from scratch using Rust. At the moment it is experimental but working. "Redox is a Unix-like Operating System written in Rust, aiming to bring the innovations of Rust to a modern microkernel and full set of applications"[Opp21a]. In a series of blog posts he describes each and every developed component[Opp21b] regularly.
- **Kerla:** it is a clone of the Linux Kernel, it is a monolithic operating system kernel written from scratch in Rust which aims to be compatible with the Linux ABI, that is, it runs Linux binaries without any modifications[Nut21b, Nut21a]. The project was started just for fun by Seiya but has interesting results.
- **Redshirt:** Tomaka is building an experimental operating-system-like environment with Rust, where executables are all in WebAssembly and are loaded from an IPFS-like decentralized network [Tom21].
- **Linux:** Linus Torvalds, the Linux creator, has demonstrated approval of the intention of including Rust code in its kernel[16], at least he is waiting to see where it goes, and the first pull request is on the way.
- **Cloud Native:** Rust is gaining great relevance in the Cloud Native environment[17], being it from its own libraries, or by adoptions and integration from Azure, AWS, Google, etc.

Besides the interest and adherence of Rust by many popular companies, developers have been releasing very interesting libraries and tools totally developed natively using Rust. Next, we will summarize some interesting examples of these Rust ecosystem in the context of modern web development.

Browsers

- **Servo:** web browser engine written in the Rust language. It is currently developed on 64-bit macOS, 64-bit Linux, 64-bit Windows, and Android. Mozilla Firefox is based on Servo, in fact, Rust was created originally by Mozilla. Website: `https://github.com/servo/servo`
- **Thirtyfour:** Selenium WebDriver library for Rust, for automated website UI testing. It supports the full W3C WebDriver spec. Website: `https://crates.io/crates/thirtyfour`

16 `https://lkml.org/lkml/2020/7/10/1261`
17 Rust Cloud Native: `https://rust-cloud-native.github.io/`

- **Headless-Chrome:** control Chrome programatically. Website: https://crates.io/crates/headless_chrome

Web Frameworks

- **Deno:** modern and secure runtime for JavaScript and TypeScript that uses V8 and is built in Rust. It is being developed by the same creator of Node.js, in an early stage but stable, as an improved alternative (the creator of Node.js states that he is designing it without the errors he originally made with Node), and the name itself (Deno) is an anagram for Node. Website: https://deno.land/
- **Rocket:** web framework with a focus on usability, security, extensibility, and speed. Website: https://rocket.rs/
- **Actix-web:** powerful, pragmatic, and extremely fast web framework for Rust. Its has one of the top benchmarks performances amongst the major frameworks in existence. Website: https://actix.rs/
- **Gotham:** flexible web framework that promotes stability, safety, security and speed. Website: https://gotham.rs/
- **Nickel:** an express.js inspired web framework. Website: http://nickel-org.github.io/

Interfaces

- **Seed:** Rust front-end reactive framework for creating fast and reliable web apps with an Elm-like architecture. Website: https://seed-rs.org/
- **Yew:** modern Rust framework for creating multi-threaded front-end reactive web apps with WebAssembly. https://yew.rs/
- **Mogwai:** frontend DOM library for creating web applications. It is written in Rust and runs in your browser and has enough functionality server-side to do rendering. It is an alternative to React, Backbone, Ember, Elm, Purescript, etc. Website: https://crates.io/crates/mogwai
- **Sauron:** versatile web framework and library for building client-side and/or server-side web applications with strong focus on simplicity. It is suited for developing web application which uses progressive rendering. Website: https://crates.io/crates/sauron
- **Iced:** cross-platform GUI library for Rust focused on simplicity and type-safety. Inspired by Elm. Website: https://crates.io/crates/iced
- **SixtyFPS:** toolkit to efficiently develop fluid graphical user interfaces for any display: embedded devices and desktop applications. Originally built in Rust, it also allows integration with Node.js, C, C#, etc. It can compile to WebAssembly and run in the browser and android. Website: https://sixtyfps.io/

- **Percy:** collection of libraries for building interactive frontend browser apps with Rust + WebAssembly. Website: https://chinedufn.github.io/percy/

Cloud tools

- **GraphScope:** a One-Stop Large-Scale Graph Computing System from Alibaba. The majority of its codebase is Rust. Website: https://graphscope.io/
- **Databend:** modern Real-Time Data Processing & Analytics DBMS with Cloud-Native Architecture. Databend aimes to be an open source elastic and reliable cloud warehouse, it offers blazing fast query and combines elasticity, simplicity, low cost of the cloud, built to make the Data Cloud easy. Website: https://databend.rs/
- **FireCracker:** open-source virtualization technology that is purpose-built for creating and managing secure, multi-tenant container and function-based services that provide serverless operational models. Firecracker runs workloads in lightweight virtual machines, called microVMs, which combine the security and isolation properties provided by hardware virtualization technology with the speed and flexibility of containers. It was initially built by developers at Amazon Web Services to enable services such as AWS Lambda and AWS Fargate. Website: https://firecracker-microvm.github.io/
- **Tremor:** event processing system for unstructured data with rich support for structural pattern-matching, filtering and transformation. Website: https://www.tremor.rs/
- **Vector:** high-performance observability data pipeline. Website: https://vector.dev/
- **Chef Habitat:** an open source automation solution for defining, packaging, and delivering applications to almost any environment regardless of operating system or platform. Website: https://community.chef.io/tools/chef-habitat
- **Akri:** Kubernetes Resource Interface that lets you easily expose heterogeneous leaf devices (such as IP cameras and USB devices) as resources in a Kubernetes cluster, while also supporting the exposure of embedded hardware resources such as GPUs and FPGAs. Website: https://docs.akri.sh/
- **Parsec:** Platform AbstRaction for SECurity, an open-source initiative to provide a common API to hardware security and cryptographic services in a platform-agnostic way. This abstraction layer keeps workloads decoupled from physical platform details, enabling cloud-native delivery flows within the data center and at the edge Website: https://github.com/parallaxsecond/parsec
- **Krustlet:** it acts as a Kubelet by listening on the event stream for new pods that the scheduler assigns to it based on specific Kubernetes tolerations. Website: https://docs.krustlet.dev/
- **TiKV:** open-source, distributed, and transactional key-value database. Unlike other traditional NoSQL systems, TiKV not only provides classical key-value APIs, but also transactional APIs with ACID compliance.

Built in Rust and powered by Raft, TiKV was originally created to complement TiDB, a distributed HTAP database compatible with the MySQL protocol. The design of TiKV ('Ti' stands for titanium) is inspired by some great distributed systems from Google, such as BigTable, Spanner, and Percolator, and some of the latest achievements in academia in recent years, such as the Raft consensus algorithm. Website: https://github.com/tikv/tikv

Web tools

- **Tokio:** event-driven, non-blocking I/O platform for writing asynchronous applications with the Rust programming language. Website: https://crates.io/crates/tokio
- **Stork:** library for creating beautiful, fast, and accurate full-text search interfaces on the web. It is an indexer for text, and a WebAssembly search interface. Website: https://crates.io/crates/stork-search
- **Tinysearch:** lightweight, fast, full-text search engine. It is designed for static websites. It is created with Rust but compiled to WebAssembly to run in the browser. Website: https://github.com/tinysearch/tinysearch
- **Linkerd:** popular Service Mesh for k8s and Cloud Computing, developed using Rust (the fastest on the market according to benchmarks). Website: https://linkerd.io/

WebAssembly

- **Wasmer:** the leading WebAssembly Runtime supporting WASI and Emscripten. Wasmer is a fast and secure WebAssembly runtime that enables super lightweight containers to run anywhere: from Desktop to the Cloud, Edge and IoT devices. Website: https://github.com/wasmerio/wasmer
- **Wasm-Pack:** one-stop shop for building and working with rust-generated WebAssembly that you would like to interop with JavaScript, in the browser or with Node.js. Wasm-pack helps you build rust-generated WebAssembly packages that you could publish to the npm registry, or otherwise use alongside any javascript packages in workflows that you already use, such as webpack. Website: https://crates.io/crates/wasm-pack
- **RustPython:** Python interpreter written in Rust. It can be used to integrate Python with Rust or be compiled from Rust to WebAssembly to run Python in the browser. Webiste: https://rustpython.github.io/demo/
- **Photon:** high-performance Rust image processing library, which compiles to WebAssembly, allowing for safe, blazing-fast image processing both natively and on the web. Website: https://crates.io/crates/photon_rs

Semantic Web

- **Oxigraph:** graph database library based on Rust that implements the SPARQL and RDF standards. It comes with a local server, API and some basic read/write functionalities to interact with RDF, and has three store implementations: one on memory implementation (faster, but limited to RAM memory) and two different local file-systems based stores (one compiled from C++, and another that is native but in development and for that reason less tested): https://docs.rs/oxigraph
- **Sophia:** a comprehensive toolkit for working with RDF and Linked Data in Rust. Website: `https://docs.rs/sophia`
- **Horned OWL:** Rust library for manipulating Ontology Web Language (OWL) data. While there are a number of libraries that manipulate this form of data such as the OWL API, they can be quite slow. Horned-OWL is aimed at allowing ontologies with millions of terms. The library now implements all of OWL2, and they are working on further parser functionality. It is being tested with real world tasks, such as parsing the Gene Ontology, which is does in 2-3 seconds which is 20-40 times faster than the OWL API. Website: `https://docs.rs/horned-owl`
- **Russoto crate:** Rust interface for the Amazon Neptune graph database (supports RDF and SPARQL standards, optimized for billions of relationships). Website: `https://rusoto.github.io/rusoto/rusoto_neptune/index.html`
- **Rio:** Rio is a low level library which provides conformant and fast parsers and formatters for RDF related file formats. It currently provides N-Triples, N-Quads, Turtle, TriG and RDF/XML parsers and formatters. Website: `https://crates.io/crates/rio_api`
- **Rome:** Rust library for working with RDF By implementing graph::Graph, one can make any data source available as RDF. Ontology wrappers can be generated from RDF Schema. Website: `https://docs.rs/rome`
- **Atomic Lib:** Rust library for managing and (de)serializaing Atomic Data, a strict subset of RDF. It powers atomic-cli and atomic-server. Atomic-server is a graph database server for storing and sharing typed linked data. Demo on `https://atomicdata.dev` Website: `https://crates.io/crates/atomic-lib`
- **Spargebra:** SPARQL parser. Website: `https://crates.io/crates/spargebra`

2.3.4 *WebAssembly*

A common practice for building more efficient Javascript runtimes is to use Just-in-time compilation (JIT) approaches, which greatly increases its performance. For many time it was the only option for most intensive Javascript based applications, but this changed since 2019, when WebAssembly was standardized. Now, it is

possible to run Assembly like code, near metal speeds, directly in the browser (but not only that, one can also run it in the desktop with virtual machines that interpret it). According to Bruyat et all. (2021)[VHP+21] "WASM is a low-level binary language which, alongside JS, is executed in most JS runtimes. WASM files are much smaller than equivalent JS files, and are compiled ahead of time, saving time at execution and opening the way to more aggressive optimizations". WebAssembly is an Assembly like format, very close to machine code, able to achieve near-native performance.

In 2019, WebAssembly became officially the fourth language for the web, according to the World Wide Web Consortium (W3C)[W3C19], alongside HTML, CSS and Javascript. It is machine code, similar to Assembly, but designed for executing with near metal speed in the web, surpassing greatly Javascript when performance matter. Besides that, it incorporates very well with the major programming languages, allowing to easily compile and integrate any tech stack with it.

Currently, WebAssembly can be compiled from a dozen of popular languages (C, C++, C#, Go, Kotlin, Swift, etc.), and Rust is one of them. In fact, Rust was one of the very first languages that allowed it, since both Rust and WebAssembly were initially idealized by Mozilla (but currently independent foundations maintain them) - for this reason, Rust, from the start, were on the frontline of WebAssembly innovation, having one of the best library ecosystems, projects, tools and compilers supporting it. The previous section explored some of these libraries that intersected Rust with WebAssembly, mainly the ones target at the Semantic Web and Reactive Interfaces.

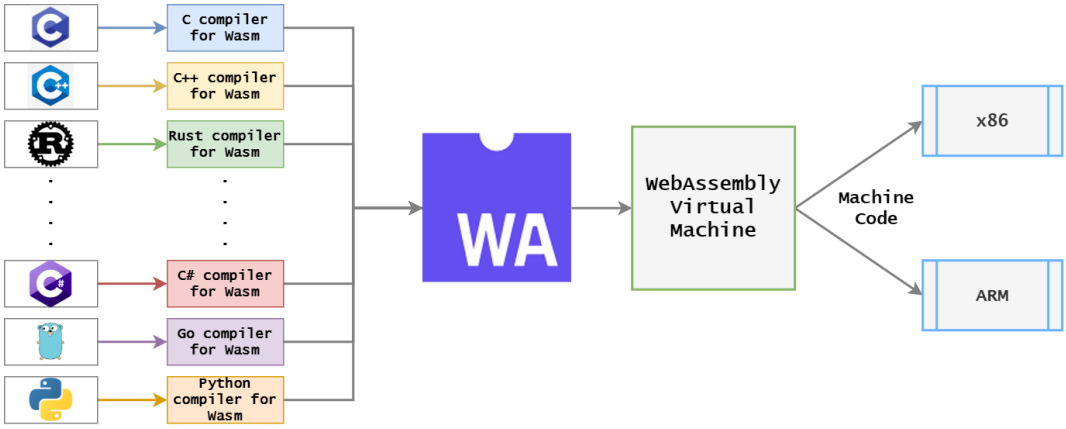

Figure 24: WASM: run pre-compiled code on the web or VM, written in any language[C21]

But the significance of WebAssembly should no be reduced to the browsers, it possibilities go well beyond that, like Watt et al. (2019)[WRPP19] stated "WebAssembly is an abstraction over modern hardware, making it language-, hardware-, and platform-independent, with use cases beyond just the Web". Figure 24 showcases this malleability. This technology is very recent, its exploration is still in its infancy and its best demos have been directed at running performative scenarios (graphics for example) or exporting desktop applications to run in the browser, created entirely in other programming languages but compile to WebAssembly (Rust has been using this method in many scenarios, for running Python interpreters in the browser, to full videogames and GUIs exported from native applications). But virtual machines allow to execute WebAssembly independently of the platform, in fact Rust has many good examples of these kind of runtimes, and for that reason, we could be witnessing the rise

of the universal intermediate language binary instruction set, like the C# CLI (Common Language Infrastructure) or the Java bytecode. Theoretically it could allow to easily glue and interface all other major programming languages with each other, or to run in any machine able to run WebAssembly virtual machines runtimes, since they already compile to it with great success.

In the context of context of the current project, WASM was integrated in conjugation with Rust, to increase performance of key interfaces and data exploration tools.

2.4 SUMMARY

This chapter analyzed the most relevant concepts to the development of the current project, summarizing them, their origins and most relevant bibliography.

In particular, we have dissected subjects like e-Portfolios *versus* Curriculum Vitae (their origins and relevant projects in the digital context); Microsservices origins and technologies supporting them (from Orchestration to Service Meshes, Message Brokers, etc.); and new and emergent Web technologies with relevance for redefining the future of Web Engineering: Rust is being adopted in the Cloud context with much success, bringing performance and security to new levels (but also being the first and one of the best WASM interfaces); WebAssembly opened a Pandora's box bringing near native performance to web browsers but also, because it promises to be an universal glue for programming languages, it will allow web apps to run easily outside of the cloud, on the desktop (and vice versa), abolishing this gap; Micro Frontends promise to bring the concept of Microsservices to the frontend realm, but are they really useful?; Semantic Web approaches promise to bring Web 3.0 to a new age, where we jump from a Linked Open Web of Documents to a Linked Open Web of Data.

3

METHODOLOGY

This chapter aims to describe the methodologies applied to the current work, in order to maintain scientific integrity and consistency during its development, and approaches taken to reach and validate a solution. The following sections summarizes the main applied methodologies. Intermediary steps, like for example Bibliography Analysis, were not described in detail because they are more commonplace, but it was essential at all stages of this project to research and build networks of bibliography and citations, in order to conduct research, following best practices and contribute solutions regarding the current subjects in study: e-Portfolios in an high-availability context and emergent web technologies for better supporting them.

In summary, we identified a problem ("why e-Portfolios applications fail to go mainstream and obsolete CV based applications prevail?"), and aimed to solve it by developing a product (Proof of Concept: "A e-Portfolio Social Network applying emergent web technologies and best practices") - this product was evaluated and analysed (using Survey Research, SWOT and Technological Acceptance Model methodologies), with collected empirical data and different methodologies, in order to study the initial questions through its solution (applying the Design Science Research methodology). The following sections expand these applied methodologies.

3.1 DESIGN SCIENCE RESEARCH METHODOLOGY

This project focus the development of a product in order to study and solve a problem. Nevertheless, the development of computer systems and other technological artifacts *per se* as the only output, is not necessary characterized as scientific research. The confusion between developing an artifact and to do scientific research is still very prevalent, mainly in the the engineering fields of research, were technology and science have a common ground[MFS20]. In order to avoid an imprecise approach between technology and science, we needed to find methodologies and fundamentals that legitimized the development of artifacts as a means for the production of scientific knowledge from the epistemological point of view, not as a means in itself[PTR08, MFS20].

After carefully analysing different approaches, we opted for the Design Science Research[PTR08] in order to guide the present work. This term can be traced back to the 70s, when Nunamaker et al. (1971)[NJ71, NJCP90] used it to describe and defend the legitimacy of "Systems Development in I.S. Research" as a legitimate research methodology. A methodology is "a system of principles, practices, and procedures applied to a specific branch of knowledge"[PTRC07]. It aims at producing high-quality and rigorous research, with clear reproduciblity and processes[PTRC07]. Design Science can be described as a means to "create and evaluate IT artifacts

intended to solve identified organizational problems" (...) "it involves a rigorous process to design artifacts to solve observed problems, to make research contributions, to evaluate the designs, and to communicate the results to appropriate audiences"[HMPR04, HC10].

DSR focus two main cycles: a) to solve a practical problem in a specific context through an artifact (design phase); b) to generate new scientific knowledge by elaborating theoretical conjectures based on the resulted artifact and its human/organizational impact (knowledge phase). Both phases provide scientific output: one by researching a problem and developing the best possible methodologies to solve it; the other by analysing the impact of the resulting solutions[MFS20, HC10, PTR08].

In figure 25, adapted from Peffers 2008[PTR08], one can find a summary of this approach, composed of 6 steps.

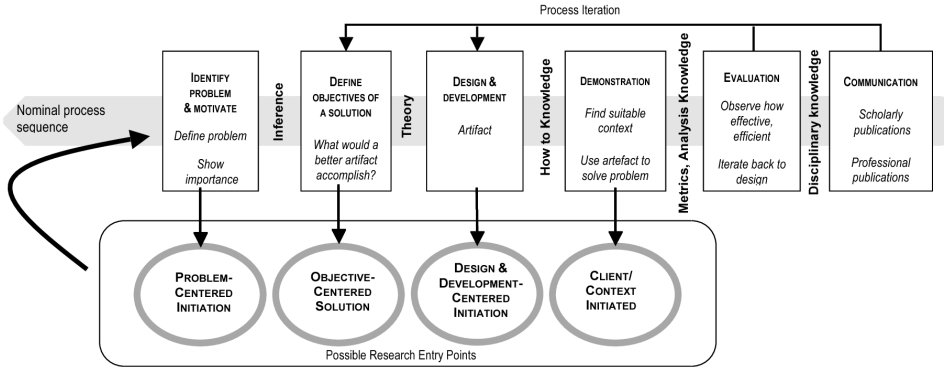

Figure 25: Cycle of the Design Science Research methodology[PTR08]

This work accomplishes these steps in the following sections:

1. **Identify problem and motivate:** this step objective is to identify the research problems that we aim to solve, in order to motivate to the development of a solution that solves them. Thus, the solution and the original questions are connected, one justifies the other. The chapter "Context & Motivation" (1.1) present these questions and proposes solutions, motivating the developed product.

2. **Define objectives of a solution:** this step aims at exploring the defined solution in the previous one, analysing it and identifying its objectives. The chapters "Work Plan (1.2) and "State of the Art" (2) explores the objectives and context of the proposed solutions. We conducted surveys in order to better analyze the target audience and product needs, its results can be found in the first sections of the "Implementation" chapter (5) and a contextualization in the last section of the current chapter (5.2.1).

3. **Design and development:** this step focus the development of the solution, from its design to architecture and production. The chapter "Implementation" (5) showcases different phases of this design and development process and adopted strategies.

4. **Demonstration:** this phase tests the developed solution, by running it in its context, simulating different scenarios. The sections "Implementation" (5) and "Resulting Materials" (A) showcase these results, tests and source-code developed during this project (see also the Appendix section at iv).

5. **Evaluation:** this step aim is to evaluate the previous phase and analyze whether it meets the stipulated objectives. The objective is to repeat the process and go back to design step until the evaluation is satisfactory. We conducted tests and methodological surveys (e.g. Technological Acceptance Model) in order to assess the impact of the application on target audiences. A Proff of Concept and SWOT analysis were also conducted. These findings are analysed in the chapter "Data Analysis" (6).

6. **Communication:** This step aims at disclosing the results and debating them with the scientific community. Besides this dissertation, during its development (2021) we published 3 papers (2 published and 1 *in press*), and participated in 4 scientific events. The main focus of these outputs were to disclose research related with Knowledge Graphs and Semantic Web, showcasing different solutions developed towards this problem, also approach and integrated in the present work. We'll also focus the participation in a national competition called "Arquivo.pt", were we won the 1st prize with the project Major Minors, delivered at the "Encontro Ciência 2021"[1] - part of this Knowledge Graph and tools were integrated in the present product, and some of the published findings and research were also expanded in the present dissertation. In the chapter "Scientific Outputs" (B) in the "Appendices", one can find these references.

3.2 SURVEY RESEARCH & TECHNOLOGICAL ACCEPTANCE MODEL

a **Before development:** before beginning the design phase of the DSR, it was important to base its development in a well defined strategy aimed at a target audience: we needed empirical data. For this reason, it was important to collect data and feedback regarding the current closest competitors in the market right now, user feedback regarding expectations and current products, its strengths and weakness's, expected approaches and methodologies, and generic statistics regarding the target audience. Survey research is a common and accepted method to perform this empirical research[For16], for that reason we built a survey and disclosed it to various mailing lists, obtaining 101 results. Survey research can be described as "the collection of information from a sample of individuals through their responses to questions"[CS11].

The obtained data were compiled and its statistical analysis gave us important insights regarding strategies to follow and best approaches planning the design phase - "Statistics provide a systematic way of summarizing what has been learned from the data"[Vas19]. In the chapter "Data Analysis" (6) we discuss these results and the survey structure, including an analysis of the pros and cons of this methodology. This approach was important for empirical insight before starting the design phase.

b **After development:** after the design phase we started a recursive cycle of evaluation the proof of concept design, improving it, and back to the evaluation cycle until the solution was satisfactory. Again, empirical

1 Arquivo.pt 1st prize press clippings: http://minors.ilch.uminho.pt/press

data were important, in order to access the accuracy and state of the solution. Survey data could be an interesting source of information, but at this stage, by itself, it could be a very broad and imprecise method if without an objective structure and proven methodologies to access the quality of the results - it was important to minimize error at the final stages of evaluation and get precise empirical data. As stated by Ponto J. (2015) "Survey research, like all research, has the potential for a variety of sources of error, but several strategies exist to reduce the potential for error"[Pon15]. For this reason, we opted for an already established and proven survey methodology, aimed at product and interfaces evaluation, with a clear strategy and researched approached, in order to get preciser and universal results.

For this reason, we opted for the Technological Acceptance Model methodology. It was developed by Davis (1986)[Dav85], inspired by the Planned Behavior Theory (PBT) from Ajzen (1985)[DBW89]. Davis concluded that the use of a system is a behavior, and for that reason PBT would be a suitable model for explaining and predicting such behavior, mainly, analysing Perceived Ease of Use, Perceived Usefulness, and Attitude Toward Using - he expanded on these models during his career.

Besides the general industry, this methodology has been tested and researched already in the particular context of e-Portfolios, where Technological Acceptance Model (TAM) was adopted in various studies for the empirical analysis of e-learning[MTTMG+08, SHO13, WW05, LCAS19, GMNHSR20]. In one of these studies, Martínez-Torres, M. (2008) states that "user acceptance is determined by two key beliefs: perceived usefulness; and perceived ease of use"[MTTMG+08], for this reason we aimed this two key variables in our initial design phase as important non-functional requirements: perceived usefulness and perceived ease of use, transmitted through strategical interface planning.

During the design and evaluation cycles TAM gathered data was used to redefine the developed solution according to its detected strengths and weaknesses. Besides the proof of concept, we also conducted a complementary SWOT analysis of the final product. This process, strategies and results were described in the "Data Analysis" chapter (6).

3.3 PROOF OF CONCEPT METHODOLOGY & SWOT ANALYSIS

Proof of Concept (POC) is a "step towards innovation and value creation, a learning step that is often decisive if it is well managed"[Ric]. Jobin, C. et al. (2020) bibliography analysis, conclude that the term origins date back to the 60s, from the U.S. aerospace, aeronautics industries and later NASA, in the context of research contracts with the objective of taking mankind to the moon[JLMH20]. Jobin, C. et al. (2020) describes that this methodology was adopted, because the partners demanded evidence achieved by the technology, proof that it worked in the expected scenarios and empirical data analysing these demonstrations were needed. One should note that this methodology does not require a fully functional product, a partial prototype, with results and empirical data proving the application of the theory were enough.

In this early context, POC was used to level of evidence assessment. Later, this methodology was also applied in the context of Biomedical and Pharmaceutics research, with the objective of future-failure risk reduction. From 1970s to 1900s it started also to become common practice in Public Research, because it allowed for a better

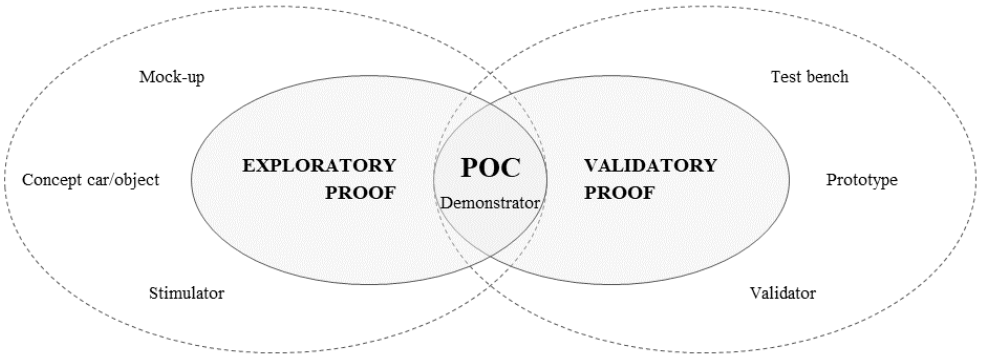

Figure 26: Proof of Concept as a double double piece of proof[JLMH20]

fund allocation and economic evaluation, bringing universities closer to industries. Entrepreneurship also adopted POC, making it part of the startup phenomenon, because it allowed to better organize the development and to attract investors. Also, in the fields of Information Technology, this methodology were integrated in different scenarios of research and industrial context, improving development cycles, tests, security analysis, (etc.) and a means to verify potential of a new technology and use cases[WSBH16, JLMH20, YPB18].

POC is a "powerful tool for decision-making" and clarifies "intermediate stages of knowledge and relationships"[JLMH20, WSBH16]. It was expanded in the "Implementation" chapter (5). This methodology is in the frontier of Exploratory Proof and Validatory Proof, dialoguing between both, encompass various levels like "Mock-ups", "Concept analysis", "Tests", "Prototypes", "Validators", "Simulator" for an idea (etc.), but at the same time being more than the sums of these parts, being a "double piece of proof"[JLMH20]. This is summarized in figure 26.

POC complements the Design Science Research design phase, by providing a demonstration. In the evaluation phase, besides the already mentioned strategies (e.g. TAM) we opted also for a SWOT analysis of the final product, at the last cycle. This approach will allow us to better understand our solution, and to study its Strengths, Weaknesses, Opportunities, and Threats, for better analysing the resulting solution and data and if it achieves the objectives initially proposed. SWOT serves the purpose of "identifying and analyzing the internal and external factors that have an impact on the viability of a project, product, place or person entities"[GUR17, Vla19]. As concluded by Gurl, E. (2017) SWOT "has more advantages than shortcomings" (...) "it is capable of analysing multiple domains and systems quickly through multidimensional modelling and analysis"[GUR17].

3.4 SUMMARY

This chapter focused the methodologies applied to the current work, in order to keep scientific integrity and consistency during its development, and the approaches taken to reach and validate a solution. Namely, we applied a Design Science Research methodology, documenting and expanding on 6 steps to accomplish it. By iterating between cycles of design, implementation and evaluation, until we reached a satisfactory product solution, we would be able to document new results and from them new scientific resources.

Supporting this approach, we also targeted a Proof of Concept, with the objective of providing a prototype demonstration to complement and testify the initial objectives and design solutions reached to solve them.

Before the project cycles started with conducted surveys, for market prospecting and decision making regarding the best strategies to adopt during the initial design phases. During the last cycles of the project, after having a prototype, we conducted Technological Acceptance Model surveys, in order to obtain data related to the acceptance and quality of the final product, and what we could improve before a public release.

Part II

HYPERFOLIO - THEORY & IMPLEMENTATION

4

E-PORTFOLIO SOCIAL NETWORK

The first part of this dissertation introduced the working context (state of the art, methodologies and goals). The current second part main objective will be to describe the strategies and design decisions regarding the end product (present chapter), and analyse the architectural and technological implementations regarding the final Proof of Concept (see chapter 5 "Implementation"). On the the third and last part, the last chapters ("Data Analysis" 6 and "Conclusion" 7) will summarize and evaluate the results. Before that, this introductory chapter (4) will summarize our design decisions and theorizing regarding the object of this study (e-Portfolios), without focusing the technological aspect (chapter 5 will cover all the architectural decisions), contextualizing the problem and the chosen solutions.

As described in chapter 1.1, the main objective is to develop an e-Portfolio product, augmented by Social Networks paradigms, and supported by high-end emerging technologies and methodologies, aiming high availability and scalability. In summary, we will evaluate three main vectors: e-Portfolios, Social Networks and emergent web technologies.

As stated before, the decision of approaching this problem through the exploration of emergent technologies (one of the base vectors of the project) are connected with the non-functional requirements that we aim to solve (a product in the context of high availability and scalability of a modern Social Network with immense traffic load). Also, the chosen methodology Design Science Research, in which a product is a means of achieving new scientific knowledge, not a means in itself, encourages this exploratory approach to technology, because the use of already common technologies would provide less significant data and results than the exploration and evaluation of new approaches lesser known. The following chapters (and the State of the Art 2) will expand on this vector: emerging web technologies that are impacting the future of web development. At the other hand, the current chapter will expand on the first two vectors: analysing the importance of the concept of e-Portfolio, current solutions (or lack of them), and how Social Networks can help in augmenting them. This analysis will serve as a base for designing the product and everything else.

As stated by Teixeira et al. (2020), Curriculum Vitae is a very imprecise medium for such important mission and information, decision makers that receive them face many problems like "incomplete, outdated, biased, private, as well as falsified and fabricated information"[TdSDAKT20]. No only that, but the focus of the problem are also in the interpretation from the decision makers themselves, because even with well prepared and objective

CVs it is common to make biased decisions, subjective interpretation, or worse, discriminatory analysis. In fact, regarding the CV there is "weak consistency or standardization in implementation internationally, and little verification"[TdSDAKT20, CB09].

From our research and bibliography analysis[HLT02, CB09, TdSDAKT20], we concluded and compiled the main problems underlying the Curriculum Vitae paradigm:

- **Imprecision:** a milestone (for milestone we mean an individual item in a CV, because it defines key events in a timeline) can tell the decision maker where someone has worked, but nothing regarding the quality of his work. Quantity does not equal quality, but the milestone paradigm in some situations exudes the first, overshadowing the latter (for instance, someone having 5 different jobs as milestones, but being fired from all of them after one year for incompetence, has not the same value has having only one job during 5 years and very positive outputs for the company - but the CV paradigm would privilege the first case, because it don't showcases the outputs).

- **Incompleteness:** a milestone can tell the decision maker which degree and score one got from a Master Degree (for example), but tells nothing regarding the human value, technical quality and soft skills (e.g. social skills, friendliness and manners, curiosity, adaptation skills and speed of acquisition of new information). For instance, a student from a rich family that paid for extracurricular tutoring and had all the time in the world, with the best books and support material and ended the degree with a score of 18, will be less interesting than a student that ended up with 16 but worked part-time at a grocery store with the objective of paying the degree fees, while his parents were divorcing with instability at home and no individual space for studying, but producing some creative and interesting projects. The same applies in a bigger scale to milestones related with job positions (the milestone only gives a timeline and a place of work, it not includes any kind of information regarding the competence of that work or quality of projects).

- **Subjectivity:** a milestone can mean different things to different decision makers. For instance, a certification in a minor tool, that someone got during the night after the work shift, even though with low technical value, could mean a lot for a decision maker that values the spirit of sacrifice, or nothing for someone that only values the technical utility of the training.

- **Bias:** the decision makers themselves influence the milestones importance. For instance, one could have worked at Fiat, but if the decision maker don't like that particular brand because he had one Italian car that broke down, or have a biased opinion that Italian companies are not as organized as the German ones like Volkswagen, he could be biased towards candidates with milestones at German companies, even though he never worked for one and has no idea of the inner structure of each company and qualitative innervation of each worker.

- **Consistency:** there is nonexistence standardization. There are many proposals, but not a universal one internationally. The adopted format deeply influences the decision making process. Europass tried to solve it in the context of Europe, but is in decline, the European Commission has been experimenting

with e-Portfolio thought the EU Lifelong Learning Program[Age16] and the Europortfolio project[Age16, Com18].

- **Verification:** a milestone can be false, or imprecise, but most of the time the CVs don't give insight regarding that. Usually, there is a verification step *a posteriori* by the companies, where they ask for certificates of degrees, but this verification rarely inspect all the milestones.

In contrast, we find that the e-Portfolio paradigm could positively contribute to solve each one of these problematic points related with CVs. Since it expands on the milestones concept by including artifacts (produced work) and assets (media evidence of this work) and evaluation (by the user and/or a community), we find that it would solve the majority of this problems.

In practice, the decision makers already don't just use CV for employee or student selection (for example), because they know the limitations of this paradigm. They also use other methods in order to secure their decision and the shortcomings of the CV, like interviews, exams, personality tests, I.Q. tests, temporary experimental phases, certifications delivery, recommendation letters, past employer contact, etc. But in an initial phase, the fact is that the CV is the solely main criteria, the major tool for getting the attention of the decision makers, but an imprecise one. For instance, Mark Zuckerberg, before Facebook became a global success, would never have had success great success applying for a job using a CV, because he drooped out of University before completing any degree, he never graduated, neither worked for a previous company. But if had used an e-Portfolio paradigm he would have immediately distinguished himself from his colleagues, because he had already developed interesting and creative tools at the beginning of his graduation, for example the software CourseMatch, that helped his colleagues to form study groups, while at same time allowed them to play an Atari Asteroids 1968 clone; later he developed Facemash (that became the base idea for Facebook). Even though he never ended a graduation, he had produced artifacts/products possibly more interesting than his colleagues, and those skills showcased more of this merit than any CV could. In fact, any company will miss on the opportunity of hiring one of the most successful entrepreneurs of the XXI century, if Mark Zuckerberg showed them his CV before the Facebook became popular, but a company could (potentially) successfully hire and identify this early developer and his ideas in an early stage if they used an e-Portfolio paradigm - companies can be missing on great minds otherwise.

With the current work, we propose a new approach for the job market and education, improving this paradigm by encouraging the inclusion and use of e-Portfolio methodologies instead or together with the CV. In particular, with the current project, we developed a concept and early prototype that propose a digital solution and Proof of Concept product, target at this niche. For that, we incorporated the concept of Social Networks, because they heavily complement each other: e-Portfolios depend on a cycle of artifact creation, evaluation, feedback and reflection (and repeat), to stimulate the evolution of the user - evaluation and feedback depend on the existence of a community, and what other concept best fits and encourages this communication process than a Social Network? We find it to be the perfect environment to stimulate the evolution of the user, by allowing him to share his artefacts, products and skills to a community with the same interests, whom would mutually evaluate and give feedback. Besides that, it would be a perfect environment to reconcile the concept of Job Market and Education, bridging both together through dedicated communities/users for each interest.

As defined by Brandes (2013)[BFW13] "in general, a social network consists of actors (e.g. persons, organizations) and some form of (often, but not necessarily: social) relation among them". He continues "the fundamental assumption underlying social network theory is the idea that seemingly autonomous individuals and organizations are in fact embedded in social relations and interactions". This environment is the perfect mix for what we aim to achieve with e-Portfolios: bridge users, companies, education entities, and thematic communities, allowing them to evaluate and share their creations (skills/artifacts). This is the reason we focused Social Networks as an important vector for this project, because it fits perfectly our needs. But in a modern, highly connected world, Social Networks are one of the most demanding systems to produce with success, because requirements like high-availability and scalability become problematic with very intensive traffic loads and user constant interaction. In a prototype context we could have built a monolith and very simple project, but that would not be scientifically interesting: we opted to idealize the technological stack of a Social Network for the century XXI, using recent and emerging techniques. But, in an early stage of design, the real challenge is how to join both concepts in a coherent and intuitive product.

As stated in the State of Art chapter (2), there exists many theories and implementations of the e-Portfolio paradigm, with some examples of open-source tools and commercial projects, but none had much success to this day, the concept never really took off outside the education sector. From our analysis and tests to already developed products, we find that the majority of them are over-engineered, developed for a research context but not practical, bombing the user with too much forms and complicated work flows. See again Figure 3 at the initial chapters (1.1), with the most popular diagram about e-Portfolios done by Barrett (2010)[Bar10]: it is very interesting and was important for the academic analysis of this thematic, but any product that would try to incorporate literally all these concepts and flow would have difficulties making the interface intuitive for the user. UI/UX studies indicate that it takes only 0.05 seconds for users to form an opinion about about a product (website) and determine whether they'll stay or leave - before transcribing the e-Portfolios theory to the mainstream public, one would need to simplify this flow and conceptualization to the maximum, to its atomic components, taking into account UI/UX concepts, for not overloading the user.

After studying e-Portfolio bibliography, current developed applications and prototypes, and also conducting surveys (see chapter 5.2.1 for a summary of these survey results) we've identified the atomic components of our product as: users, interaction and community:

- **Users:** client users, companies seeking them and/or education institutions.
- **Communities:** users seek other persons with commons interests and skills, for evaluation and sharing. For this reason, mechanisms for creation and joining communities would be important, as well as communication tools.
- **Interaction:** this is the more crucial aspect of the product: users want to share and create, the product is a medium for that. What will the users create/share? How will they do that? This is what we will analyse next.

For the "Interaction" vector, an e-Portfolio paradigm would aspect artefacts. By artefacts we refer to user creations in different formats (a painting, a software developed, a published paper, the architecture plan of a

house, a poem, etc.). We don't wish to restrict these creativity, but we need to structure and organize it, for that reason we opted to organize it by the medium format of the file (an image, a video, a text, a 3D mesh, an audio file, etc.) and to allow the user to categorize it accordingly to communities (these communities are categories in the mentioned ESCO ontology, that we used to categorize and structure all the data). Optionally we allowed the user to define skill trees (also, accordingly to the ESCO ontology skills ramification; but our objective is not overextend the forms and tire the user with too much input, for that reason skills are optional; with the ESCO ontology we can also derive automatically many of the skills from the main job classes that will me renamed to the communities - the form will automatically show a preview of skills ramifications, for the inner search/structuring and export of the portfolio and CV).

But we didn't want to focus the platform only on artefacts, because we find that diminishing in some testes products - e-Portfolios don't need to exclude entirely the CV paradigm, both can complement each other. For that reason, we opted to sub-divide the "Interaction" vector in two more types: one focused on events (for instance: graduated, got this new job, obtained a certified, attended this event, etc.) and free text to create a blog like interaction (for example: sketches, thinking about this, planning that, etc.). In summary, we divided the "Interaction" vector of the e-Portfolio paradigm in three types:

- **"Did this":** artefacts that the user created.
- **"Lived this":** events/milestones that the user lived.
- **"Said this":** things that the user thought.

With this system, we were able to condense all the major aspects of an e-Portfolio, without compromising usability, maintaining an intuitive and familiar interface for the user. Each "interaction" output (from now one we will call it "post") will interconnect with the community and users vectors, because each post done by the user will also have ramifications with communities (in its inner ontology graph) and will be publicly available to all users in a main page. The "Did this" type of post is the central type to the portfolio development, since it will include all the artifacts created by the user. The "Lived this" vector will be an hibrid CV format, based on milestones, allowing to easily exporting automatic portfolios/CV like formats in paper format for instance. The third vector ("Said this") will allow to create a blog like experience, and this will be a useful post type for companies or education entities too, to freely share content. These three post types will share the same generic form interface, we could add additional contextualized fields but we opted for coherence and an intuitive interface for the user. We reduced this form to its basic most important components, because complex forms would only discourage users from daily posting content. These common mandatory atomic fields of the form would be: a title, associated communities and the post type (did, lived or said this) but "did this" would be selected by default already to make it faster since it is the most common and important post type. Optional fields would be: a descriptive text field, a "when?" date type field, a "with?" field that would allow to select other users, a "for whom?" field that would allow to select companies or educational entities (registered or not in the platform), and a "categories/skills" field that would allow to add and better refine associated skills (the ESCO ontology, accordingly to the community would already show suggested ones). Then we would have hibrid form fields, that would be mandatory for some post types and

optional for others: the "Add Assets" field, which would allow to add various attachments and would be mandatory for the "did this" post type (for example, if the user published a paper, he could add/upload as assets the original document in PDF format, the source document in LaTeX and an external link for the persistent doi URL of the published paper; optionally he could also add assets with images of relevant diagrams/statistics that he created for the paper, etc.). Besides the assets, the users can upload or select one of these to illustrate the post in the front page (an image or GIF animation), for this reason we also included a fieldcalled "add illustration". Figure 33 (in the Implementation chapter and Mockups section) showcases a sketch of this intended form structure, this mockup image was created during this initial study and analysis.

Artifacts, in the context of e-Portfolios, are the most important concept: they are digital evidence of progress, achievements and experience experience by the user over time. Each artifact is like a creation, a production, a work, etc. We subdivided this artifacts with the concept of "assets": artifacts are the general work/creation, and assets are the media types that showcase and prove these creations. There are different theoretical conceptualizations for defining e-Portfolios as we observed in the State of the Art chapter (2, but the majority of them use the concept of artifact to define each atomic work - we added the concept of "asset", to sub-divide the media types produced in the context of each "artifact" (in fact, in the context of our product, the concept of post is the artifact, and the attached media documents are the assets, and all else is meta-information to define and structure it). In other words, artifacts are examples of work created by each user, and this work is proved and showcased through media documents (digital artifacts, or in our words "assets"), organizing and combining different media types into cohesive units that communicate a narrative.

One should note that we also planned that the main front-page were the post will be displayed and interacted with, will use hybrid functionalities common to "web content rating" platforms (for example Reddit, 9Gag, Imgur, etc.) in which users can vote content up and down, and each post will go higher in the ranking the more the votes, appearing for that day in the front-page or top most positions of the wall of posts accordingly to the votes - it is also common to structure into sub-pages like "trending", "hot" or "fresh" accordingly to the votes (maximum votes, most voted in the past 5 minutes, or fresh content without votes yet, etc.). Usually this kind of systems are generic and daily based, but we will divide the front-page votes in "Daily", "Month", "Annual", "Forever", and include filters for the country and community (show by country or show post rankings by community). This way, we intend to create a gamification[KSLB18] like sentiment, in which users will compete to appear in the top rankings (the filters like "country" and "community", in addition to contribute to this gamification competitive sentiment by focusing in in local communities familiar to the user, they were implemented also to allow for equality, and to help companies find the most skilled individuals in their zone or field). This system also helps companies to more easily find skilled individuals, because someone winning the best post position for a year, with the most community votes, will be a better target for companies, because if his peers loved his creations it means his public acceptance is good - this way, communities themselves could influence on who will be more easily hired by a company. One should note, that only the first vector "Did this" will be part of ranking system; the events are target for automatically exporting a CV like format, and free text are only meant for communication.

We concluded that it would also be important to not fully discard the concept of Curriculum Vitae, because the "Lived this" component is an hybrid milestone format, for that reason we opted to maintain some parts of it. Any

user using this product, would be interested, besides the community and evaluation aspect, in the Job Market, and for that reason would wish to be able to export their portfolio to other mediums in order to share it with companies, including a CV. For this reason we opted to include tools and interfaces for automatic compilation of results, in an intuitive way, that would allow any user to select filters (areas, skills, years, etc.) to include in an hybrid e-Portfolio and CV format, that would automatically build a timeline of each artifact and event on the life of the user. For this system to work effectively , users would need to fill the field "when?" in the form for each artifact/post, in order for the system to produce an accurate timeline of milestones and works. Our plan is to in the future provide and allow various types of templates to be upload by the users (in HTML format), so that users can choose different visual layouts for their CVs/portfolios export (we use the therm portfolio instead of e-Portfolio, because when one downloads it in static/paper format it looses the media assets functionalities, it is not digital anymore, only showing persistent links and meta-information for each artifact). We also plan on giving the users various filtering options, allowing them to adjust what type of post (did/lived), community, timeline, posts with a specific skill (etc.) they wish to add to the export. If users wish, they could download only a traditional CV (if they only activated the "lived this" post types based on milestones) or only a portfolio type document with only developed software or paintings if selectiong the "did this" type of post and the corresponding community/skill set.

In conclusion, Figure 27 summarizes our design strategy for the product and workflow, mixing e-Portfolio, CV and Social Network concepts. It will be centered around users (divided in common users, companies and education entities), communities (for feedback and belonging sense) and interactions. The later will be centered around three types of Posts: "Did this" (portfolio artifacts), "Lived this" (CV milestones) and "Said this" (free brainstorming blog like interaction). This interaction will be associated with communities and graph ramifications of skills, allowing interaction through comments, sharing and voting (votes will be part of a ranking system, that will create a gamification feeling in the users, and a filtering tool for companies seeking the most talented and voted individuals for each skill set). This will be the target base for our product, reducing all the e-Portfolio concepts to an intuitive minimum, so the users can feel familiarity with the platform. To aid that familiarity, we will also maintain some CV basic functionality disguised as the "Lived this" type of post.

We believe that this new idealized product, augmenting the e-Portfolios methodology through Social Networks and emerging web technologies, could positively impact social, educational, and economic vectors (besides the scientific one that this dissertation directly tries to explore).

Kahlawi (2020)[Kah20] stated that "Labor market governance is one of Europe's top priorities", he continues saying that "market governance is an important challenge because the job market is a complex network involving many diverse actors", concluding that "therefore, the European Commission has proposed ESCO". European Skills, Competences, qualifications and Occupations Ontology (ESCO) is precisely one of the tools (related with the new emergent web technologies vector of this dissertation) that we strategically incorporated on the current product, in order to tackle its many design challenges. We hope that through it, we can prove the importance of this new paradigm and make it more accessible to the general public.

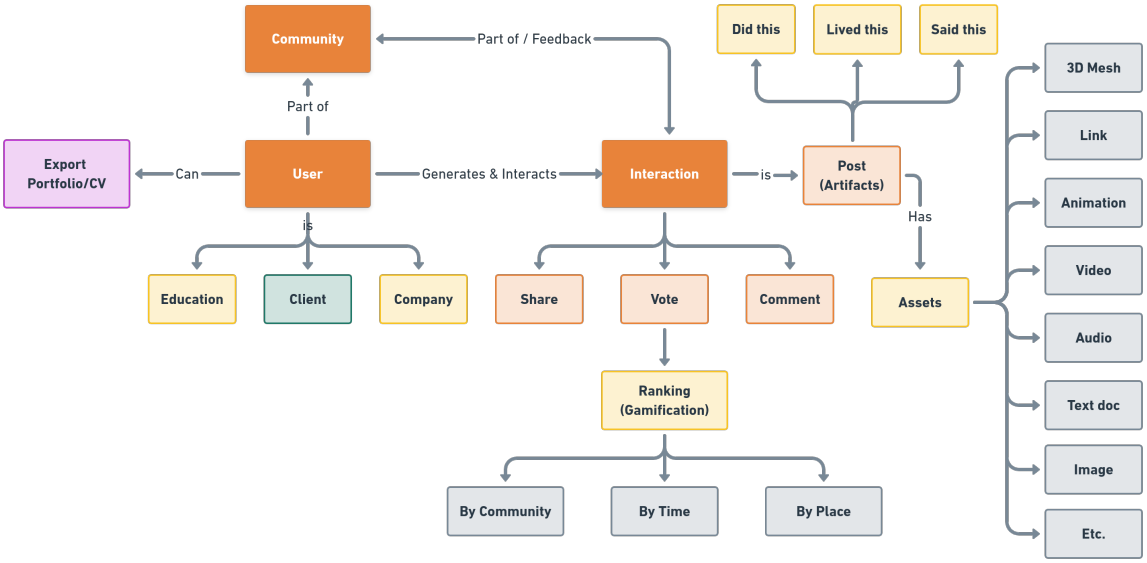

Figure 27: Product design strategy and workflow

4.1 SUMMARY

In this introductory chapter, we summarized the roadmap for Part II and presented the main theory and design decisions that will conduct the implementation of the Proof of Concept. We started by analysing our design decisions and theorizing regarding the main object of this study (transposition of the e-Portfolio theory to a mainstream product). For that, we analyzed the differences between the Curriculum Vitae paradigm and the e-Portfolios, compiling 6 problems associated with the Curriculum Vitae paradigm. We then proceeded to define how an e-Portfolio approach could solve them, and outlined a solution in the context of the current product.

We concluded that Social Networks were a catalyst that could improve our objectives, because it provided a community, evaluation and reflection mechanism needed in a e-Portfolio ecosystem. We started by defining the atomic components we should target, and methodologies to transpose them to an e-Portfolio context: we design an hybrid solution, between Social Networks, e-Portfolios and Curriculum Vitae, where artifacts were reduced to their most atomic aspects, categorizing them into three types ("did", "lived", "said"), based on a triangle centered around interaction, users and communities. We analysed past projects on this area and concluded that they failed because they over-engineered the concept, and were not intuitive enough, so we opted to target these vectors, reducing the e-Portfolios to its most elemental concepts while stimulating familiarity. In the end, we defined and built a workflow diagram with an overview, further documenting the final design strategy for the end product.

5

DESIGN AND DEVELOPMENT

The main methodology chosen to guide the project was the Design Science Research, as described in chapter "Methodology" (3), in which a cycle of problem-solution generates new knowledge: one defines a problem, designs solutions to solve it, evaluates the final product and repeats, generating new knowledge until satisfactory results are obtained. The development of a Proof of Concept is not the objective by itself, but it is a means to explore the problem, to research new approaches and develop new technologies and theories - it is a process target at generating new technological value and knowledge by analysing the process itself. For this reason, besides solving a problem with a new product, we aimed at innovative and alternative technological implementations for our solution, in order to generate new knowledge along the design phase cycles.

As the title of this dissertation implies, the base problem that we seek solution through a product is the "Development of a e-Portfolio Social Network", but the second part of the title imply the chosen design phase focus, aimed at generating new knowledge: "through emergent web technologies". We could solve the base problem and develope a POC with obsolete common technology, like for instance MVC architecture and static pages, without adventuring through WebAssembly, microsservices, micro frontends (etc.), using easier abstractions for mobile development (e.g. React Native, Flutter, VUE Native, Progressive Web App, etc.) or focusing old languages with tons of templates instead of emergent ones like RUST without so much documentation and use cases due to being most recent - but this path, even though easier, would not produce as much new knowledge. For this reason we opted for the exploratory route, we aiming not only at producing a solution (product), but also to explore new and better ways to accomplish it, by exploring emergent technologies and architectures and publishing the results. We also aimed at producing an enterprise level product, with the objective of incubating it and deployment in a real-world scenario after the dissertation in a Startup context. Since we were working in the context of Social Networks and targeting future professional deployment, in order to achieve our objectives we followed to main non-functional requirements during all design phases: horizontal scalability and high-availability.

This chapter describes the development of the final product, and documents the design research. It was divided in sequential sections of the development stages.

5.1 PREVIEW

Before describing in detail the major aspects of the product, we will make a brief state point summarizing the current state.

We designed and developed a website, a mobile app, and configured a physical dedicated server with a Kubernetes network of microsservices and NGINX routing. Currently we have functional (but incomplete for public deployment) prototype of the product website (made using microsservice architecture and various technological stacks from React, Django, NodeJS to Rust, Triplestore Databses, etc.) tested in a cluster of three dedicated servers (using Kubernetes, Rancher, Cluster Database based on CockroachDB to distribute the load between the three servers, etc.), and a functional mobile app (made using Rust, Kotlin with webviews). All the source-material of the project is available in GitHub at `https://github.com/Paulo-Jorge-PM/hiperfolio`, alongside the source-code and prototyped material not used and discraded during the project (e.g. the initially React Native demos for the first mobile app builds; the Baton Rust app initially created to be proof of concept for the WASM as a microsservice concept, but discarded due to lack of time, etc.). Our objective is to market this product in a real commercial context, so we over-engineered each detail so that it would be full functionally product ready for a real scenario, but at the moment it is still in evaluation and debugging phase in an early alpha prototype phase, we plan on adding new functionalities and study business models before releasing a public version.

Our objective was to design an e-Portfolio product able to compete in the job market segment with an alternative concept, improving the traditional Curriculum Vitae approach. For that we researched e-Portfolios State of the Art and current best implementations, and based on these concusions we opted to reconcile it with the concept of the Social Networks, improving on the LinkedIn formula (which is the most known application in this market segment). For that we started by conducting surveys, that helped us identify user preferences, expectations and vectors that LinkedIn should improve upon. This data was useful to define the early design mockups, objectives and requirements.

We opted to implement a microsservices architecture, supported by Kubernetes, because it would facilitate future scalability and high availability (project's main non-functional objectives), and at same time, it would improve cost control, maintainability and future expansions in the context of a big team. This architecture also allowed us to experiment freely with new technologies, because each microsservice is modular and independent from the rest, allowing to experiment with WebAssembly, Rust, and new technological ecosystems different from the traditional monolithic most common approaches and tech stacks. One of these technological new approaches was the introduction of Knowledge Graphs, based upon clusters of ontologies, to improve the search mechanisms and introduce Semantic Web techniques - for that, we used known open-source ontologies (e.g. ESCO), and built an independent tool, integrated through microsservices, that allowed to build graphs of entities from indexed texts, and endpoints exposing them through WASM interfaces. In order to maintain the high availability objective, we also introduced techniques and data models adequate for that context, for instance we used distributed databases, for instance CockroacheDB, which is a cluster of scalable distributed databases, supported by communication protocols like Service Brokers based upon Redis, Graph Databases and ElasticSearch. For the mobile app we opted for an experimental hybrid architecture as proof of concept, in order to explore the Rust programming language potential for the mobile market.

In the following sections we will describe in more detail each major design considerations.

5.2 MOCKUPS & SURVEY

We are building a platform to compete in three main markets: job market/networking (e.g. LinkedIn, Xing, Jobcase, Recruitment Agencies, etc.), resume builders (e.g. Europass, DeGóis, Canva, Zety, Novoresumé, Ciência Vitae, etc.) and Social Networks with creativity oriented communities (DeviantArt, Pinterest, Reddit, Quora, Tumbler, etc.). The selling point of our product in these three markets is a new paradigm: e-Portfolios. The Social Networks oriented to creative communities has also the Knowledge Graphs, WASM and Microsservices approaches as a differentiating selling point. In summary, our product should be able to integrate these three different markets in an homogeneous result, focusing two central differentiating points in all its design processes: e-Portfolio paradigm and new/emerging web technological stacks.

As analyesed in the previous chapter (4), our main competitor in this market segment is the LinkedIn platform. In a second degree, we have the LinkedIn alternatives (lesser known), the creative communities (e.g. DevianArt) and Social Networks related with knowledge and creativity (Quora, Pinterest, Tumblr, Reddit, etc.). Before initiating the design phase, we conducted surveys to analyze the current user perception of LinkedIn, competition, and the CV *versus* e-Portfolio public receptivity. We then used this data to design mockups and strategies, regarding the integration of the three target markets with the two differentiating vectors. We will start by summarizing the surveys, and next the design decisions.

5.2.1 *Survey - Market Prospection*

The complete market prospection survey questionnair can be found at Appendix C. We will briefly analyze the most important results. But first, we should mention that the accuracy of any survey dependents on the structure of its questions, the sample size, and the target audience. Our sample size is 102 answers, and our target audience where mainly academy staff/students (we used university's mailing lists to disclose the survey) and users of social networks related with technology (we disclosed the survey in two Reddit rooms, one national and other international, and in a forum dedicated to technology). For this reason, our sample audience is very specific, ranging from tech savvy public to graduation students that usually are familiar with the products in study, mainly Portuguese users. This is a strength when it comes to analysing the competition and current active users (our sample is focused on the target audience of the competition), but we could not fully analyse users outside of this bubble, which don't know this market segment options, for this reason we could not analyse how to attract outsiders.

The sample public include 25.5% users between the first graduation age (18-23 years old), 29.4% have the most common age for their first and second jobs after a graduation (24-30 years old), 22.5% (31-40) and 20.6% (41-67 years old) are experienced users which should had some job experience by now, and 2% are possibly retired users (67 or more years old). Only 7.8% don't have a graduation (36.3%) or a postgraduation (55.9%: 35.3% Master; 20.6% PhD or more).

Regarding their job situation, the sample audience only include 13.7% currently unemployed and 16.7% that never worked before, the majority is currently employee (60.8%) or self-employed (8.8%). 13.7% had only one job experience, 69.6% had more than one job, and 19.6% had 5 or more jobs.

Regarding the job market, 51% resorted to the services of dedicated websites or employment agencies. If we exclude the 16.7% that never worked and for that reason answered negatively to this point, this result becomes more significant - the majority of the population seek the services of our product market segment, which is an encouraging result. Also, another positive factor, is that 82.4% of the sample users which to change/find a new job in the near future, if the opportunity arises.

Regarding the services/products used for job seeking, the majority answered LinkedIn by a large margin (61.9%). Other answers, in a more diluted and distributed percentage, include: ITJobs, Manpower, Reditus, Olisipo, NetEmpregos, Kelly Services, Michael Page, Randstadt, Twitter, Indeed, Alerta Emprego, Companies websites, Sapo Empregos, HR websites, Jobs UK, Elempleo (Colombia), Freelancer, Upwork, Select, Akadeus, Catho, Hays, XING, IEFP, Glassdoor, ResearcheGate, Stack Overflow, Teamlyzer, Egor, Eracarreers, BEP, Euraxess, ACICE, Landing Jobs, Google Search. As we suspected by the initial research, LinkedIn is currently, by a large margin, the bigger competitor on this market segment. For this reason, in anticipation, we also focused it on the survey.

Regarding the LinkedIn section in the survey, we highlight the percentage of users that already used or knew it (89.2%), but 99.9% never paid for premium features while using it (interesting data regarding the business model). Only 17.6% have started using LinkedIn in the last 6 months or less, the majority have used it for more than one year, and 29.4% have used it for 5 years or more (see Figure 28).

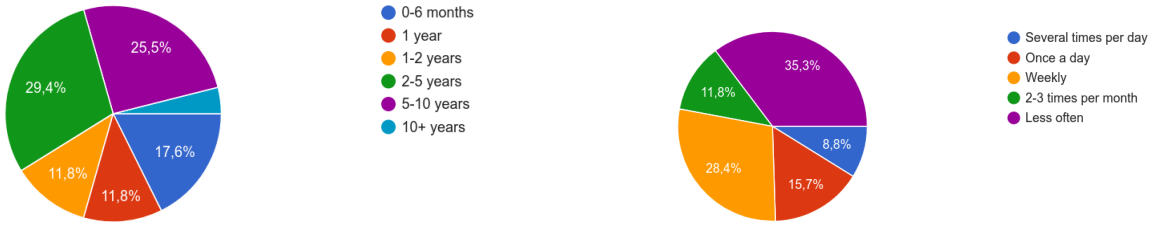

Figure 28: Left: How long have been using LinkedIn? Right: How many times a day visit LinkedIn?

At the other end, LinkedIn popularity seams to not be currently solidified, because our sampled users are not totally satisfied with the service, neither the pragmatic results of its use are very positive. This means that we have a chance to compete in this market: the major competition has not solid legacy. For instance, in Figure 29 we can see that the majority of its user base are neutral or negative regarding their satisfaction (64.2%, in which 49.5% are neutral, and 14.7% are negative).

Also, as Figure 30 indicates, the percentage of jobs successfully started through LinkedIn is very low (only 7.8%), and job interviews that originate on LinkedIn are only 24.5% (the success rate for a sample of 89.2%) users that are familiar with LinkedIn is very low). One should note that the success rate of job interviews is also less than half (31%), so the them employers are not meeting the right employees or the employees are not fully prepared and LinkedIn is not providing quality services in that regard. Our product has margin to work on improving the number of interviews started through the platform and the success rate of those: one of our chosen

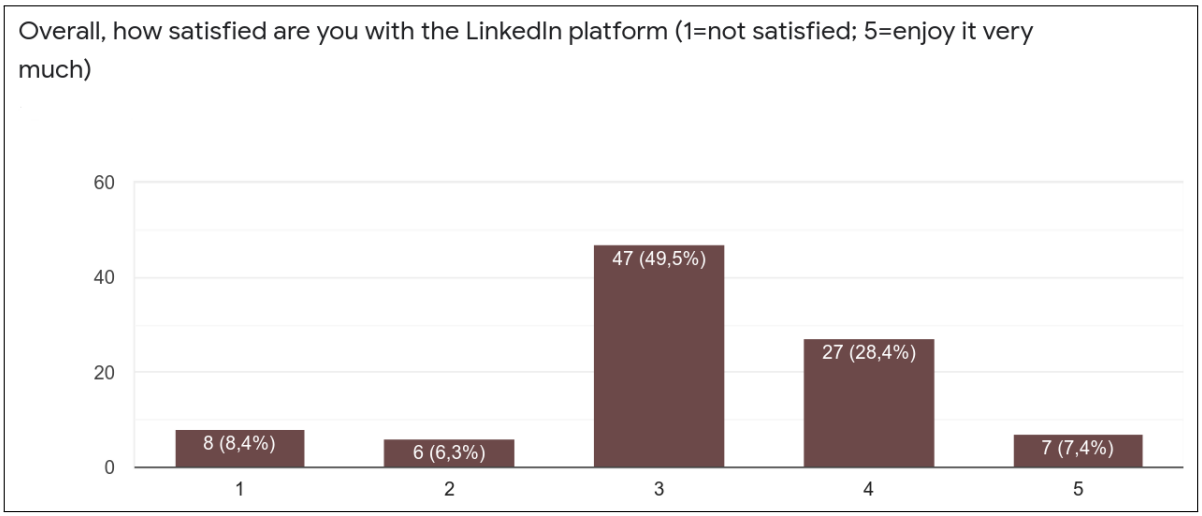

Figure 29: LinkedIn sser satisfaction survey

strategies is the refinement of the search criteria and algorithms, in order to get in contact the right employees with the right employers, for that we applied techniques related with the Semantic Web and Knowledge Graphs, as we will analysed in the following sections. Another interesting results is the percentage of users that are active and post in LinkedIn, 71.6% never posted, and 23.5% posted less than once a month - the user activity is very limited, LinkedIn ins not fully engaging with its user base, which is too passive, only filling their CV and never engaging again with the platform. The e-Portfolio paradigm improves on this problem, because by its nature it demands and encourages constant engagement between the user, his portfolio and the community.

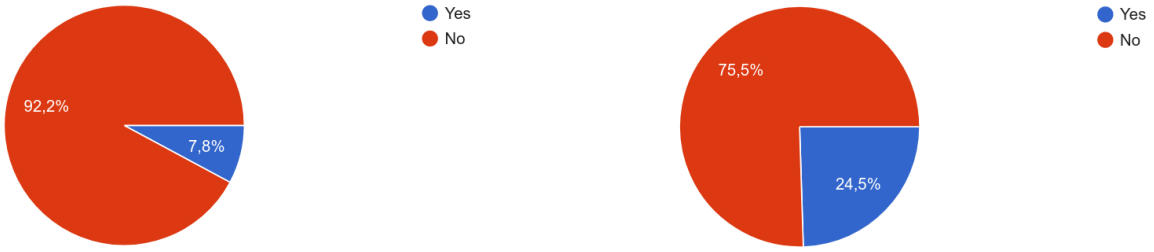

Figure 30: Left: Jobs got through LinkedIn. Right: Job interviews got through LinkedIn.

In order to better perceive vectors in which we could improve, the survey included free text fields asking the user sample what they most liked about LinkedIn and waht they least enjoyed. The most common positive answers focused the job offers and a place to find them, a centralized place to read companies news, the networking aspect and friends connections, ability to find companies and people, reputation, job oriented Social Network instead of trivialities orientation, etc. One should note that some users also answered "nothing", or that it had no utility on his job segment. When it comes to negative aspects regarding LinkedIn, the sampled users most common answers focused the interfaces ("complicated", "ugly"), not being able to export the CV to other formats on not being able to import from other platforms or formats, being owned by microsoft, it is turning into "motivational crap without the professionalism expected", more and more people using it more as a Social

Network for personal life instead business, people postings memes and it being transformed in a Facebook type of product, not being able to showcase soft skills, too much focused timeline events, "posts of people bragging about themselves", "some people acts like they're in Instagram", the job above all else mentality, business oriented without creativity skills, hard to find job opportunities, "it's a Facebook for", "too many questions", etc. The gathered answers were very usefully in designing a strategy for our product. For instance, we gave a lot of attention to the last negative answer "too many questions", because we also feel that, and more, we have found that this is also one of the major problems with the e-Portfolio paradigm: users don't like long forms and to waste time posting, for this reason our main strategy focused the simplification of the e-Portfolio format, as we will describe in the next section regarding the Mockups.

Regarding the traditional Curriculum Vitae paradigm *versus* new approaches more oriented to e-Portfolios, the survey included targeted questions to analysed user predisposition to change. The results were satisfactory, 70.6% of the user sample feel that the traditional CV model is not enough to showcase their full potential to a contractor. When asked if they would like to have more alternatives to LinkedIn, less focused on CVs and milestones but more oriented towards skills and creativity, 78% of the user sample wished for those alternatives - this results confirmed that there is a growing need for alternatives in this market segment, as we can confirm in Figure 31.

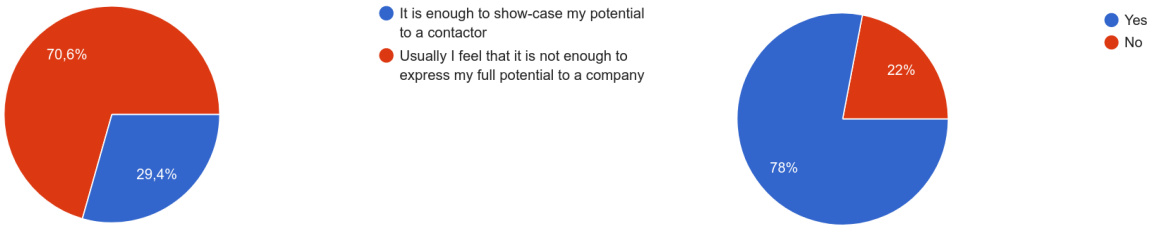

Figure 31: Left: Feeling regarding CVs. Right: Wish for LinkedIn alternatives not focused on CVs?

5.2.2 *Mockups and Strategies*

As stated before, one of the most common problems pointed to the LinkedIn (and CV builders in general, starting by Europass and why it is failing), is related with "too many questions", excessive forms. Users don't like to wast time on long forms, posting should be fast, too many questions destroy user motivation to use the application in a daily basis: these are general UI/UX known strategies. This is also one of the major problems with the e-Portfolio paradigm: the majority of the digital adaptations of this paradigm try to included every aspect of this complex approach, ending up suffering from this UI/UX problem. Solving it was one of our first and most important strategies, we worked on condensing the e-Portfolio format to its basilar aspects, hiding all the complexity from the user.

One common problem with the CV format and CV builder platforms, is that the user needs to complete and/or repeat the tedious process of inputting and compiling his life milestones. Users with many years of career, or some jobs like the academic ones, end up with very long CVs, with dozens of pages. Platforms like Ciência

Vitae, Europass, DeGóis, ORCID (etc.) all suffer from this, and the user base usually are very negative regarding transitioning from one to another or starting the process of building his Curriculum Vitae. Also many e-Portfolio approaches also suffer fro mthe same problem, in a larger degree, because as analysed in the Stated of the Art chapter (2), the e-Portfolio implementation is very complex and its full digital implementation usually results in driving users away.

For this reason, we opted for a system, in which the users don't need to insert all his past information and loose much time in order to start using the platform. The user is asked to engage with the application by posting content, and this content will be used to automatically build his CV and graph of skills. This content can be asynchronous, the user can just publish something he just did, it is not encouraged to post all his milestones and creations in one session. In a first instance, the platform asks for posts, regarding a set of activities, and asks the community to engage with it. This engagement will encourage new publications, and this cycle will slowly but steadily build the user CV totally automatically. The first 5 seconds of a user visiting a new website define all th experience and future engagements: we wanted to make the entry easy and non-time consuming in a first instance, allowing for more complex engagements optionally later - much of our strategies aroused from this premise, following the motto "easy to use but complex to master". We summarized our aims related with the interface as:

- Forms should be reduced to essential information.
- Don't waste users time, publications should be fast.
- e-Portfolio cycle should be simplified to its atomic components.
- The possibility to generate and export the user Curriculum Vitae to different formats. T
- The CV should be automatic, based upon the user posts, we should not make user intervention mandatory (but give the option to widen and complement the information).
- The platform should build a graph of the user skills (both hard and soft skills), user interactions and interests, in order for the algorithms better refine matches with companies, people and jobs.
- This graph should be publicly available and users should be able to interact with it thought visual interfaces.
- Search results and filtering options should be refined by implementing Knowledge Graphs techniques.

To support our strategies for the interface, we built several mockups before the implementation, in order to study the best approach. Figures 32 and 33 showcase some results of this mockup phase (further expanded in the final version, analyzed in the Frontend 5.5 and Mobile 5.6 interfaces sections in the Implementation chapter at 5). The most challenging aspect were the forms related with the user posts, because they are the most time consuming aspect of the engagement, and if they are not intuitive and easy enough users would get tired and avoid publishing new content. For this reason we simplified it to the minimum possible, to its most important aspects needed to represent a portfolio in digital format: we eneded up dividing the concept of an "artifact" (digital portfolio production) in three different options: Did This", "Lived This" and "Said This" - this way, besinds

making the form intuitive and indirectly telling the user what kind of content is expected and he can engage, we also were able to better categorize information and divide portfolio inputs ("did this"), from milestone inputs ("lived this", and from general discussion posts ("said this"). This strategy and methodology was further expanded and described at the section E-Portfolio as a Social Network (4) and the following ones related with the user interface.

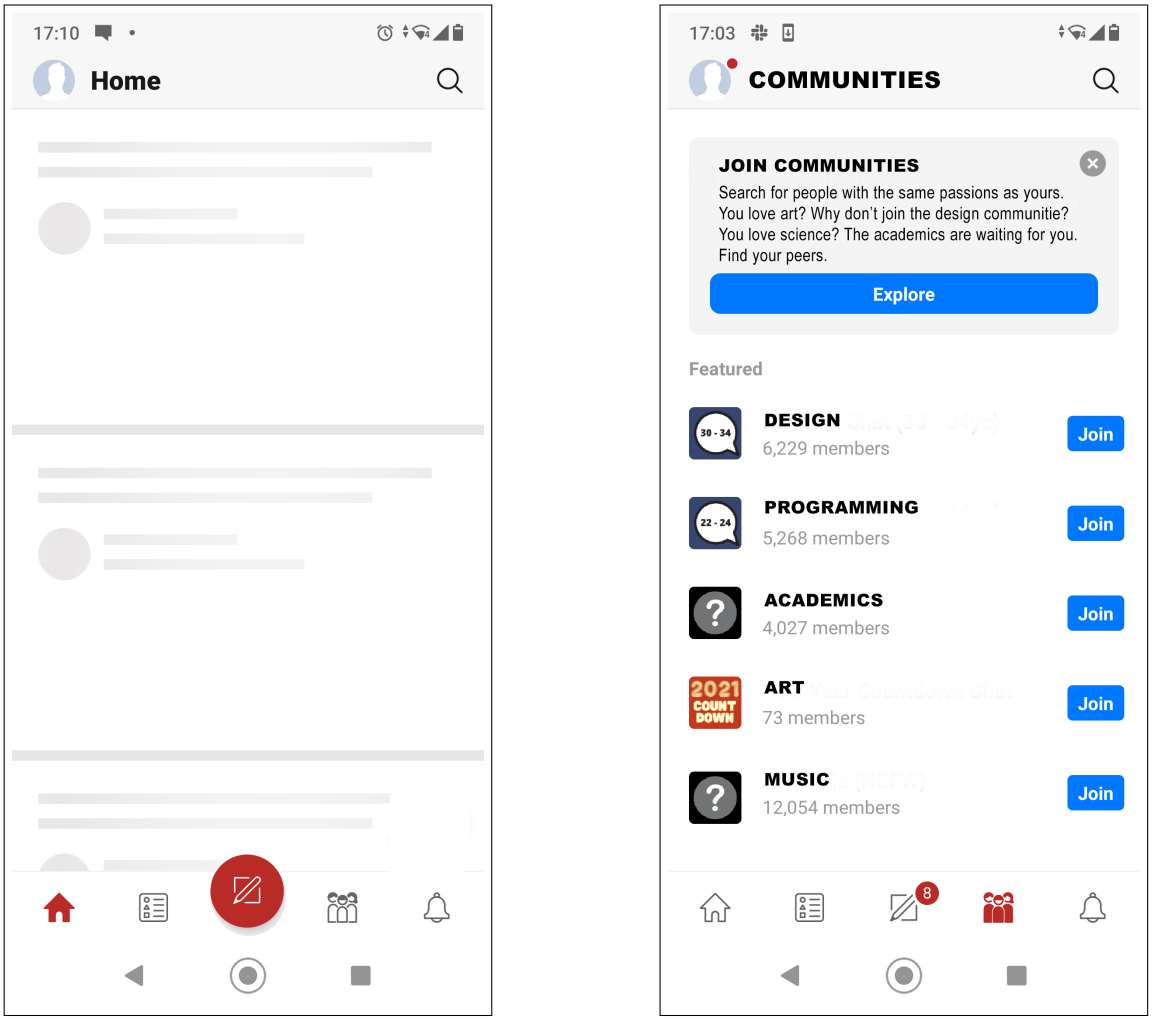

Figure 32: Left: Mockup 1: Homepage infinite scroll. Right: Mockup 2: Communities page

5.3 ARCHITECTURE

In the following sections, we'll summarize the core components of the final product architecture. For that we divided the analysis and components in layers, for an easier exposition: Static Content Layer, API Gateway Layer, Service Mesh Layer, Kubernetes Layer and Message Broker Layer. In general, the architecture was based on microsservices and technologies supporting/augmenting it.

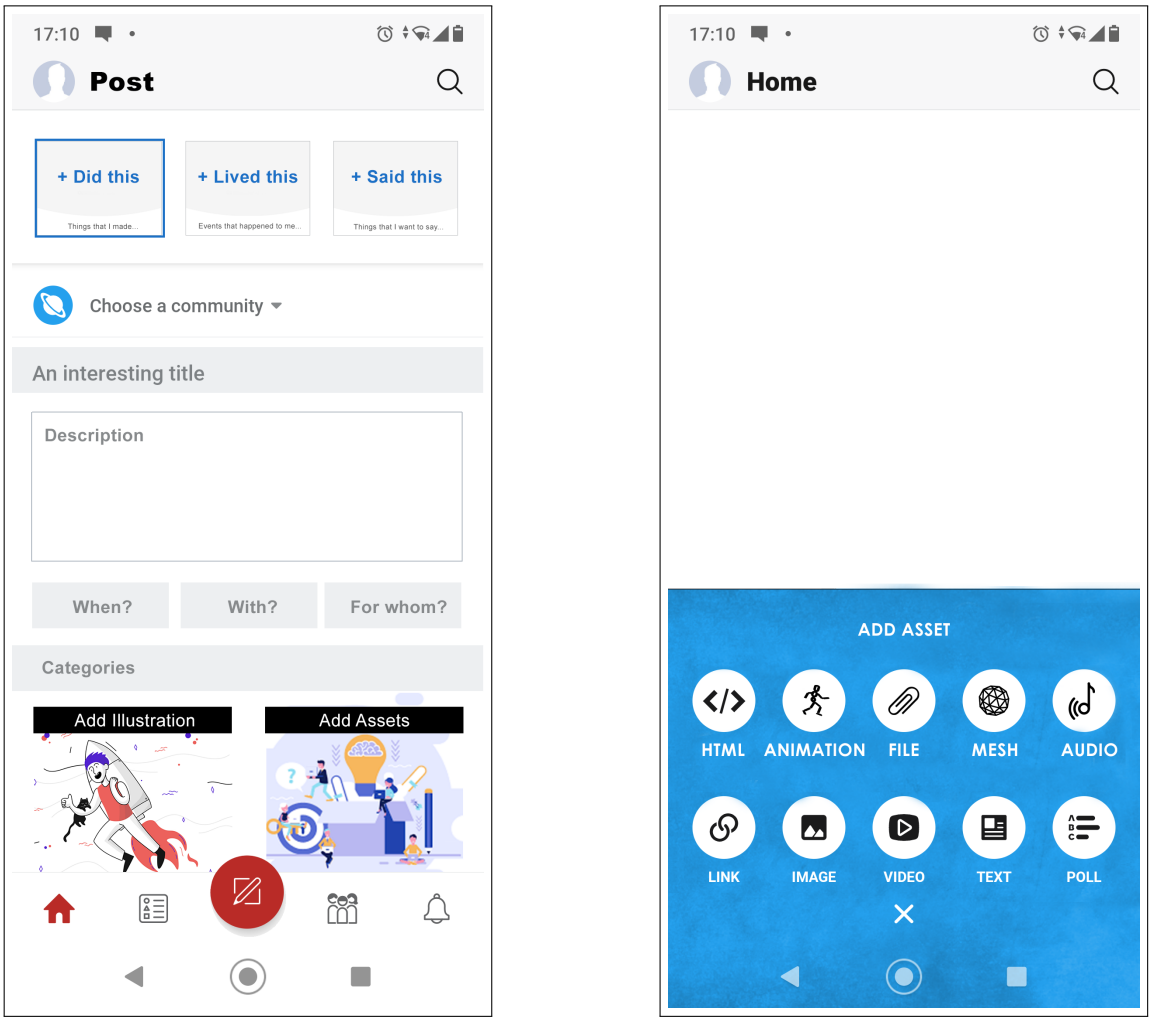

Figure 33: Left: Mockup 3: Publishing form. Right: Mockup 4: Assets form

5.3.1 *System Overview*

Every web application is a technological sandwich, composed of many different but complementary parts. The chosen technological stack and architecture has deep implications on the final product non-functional attributes.

LAMP (Linux, Apache, MySQL, PHP) is an example of one of the oldest (and obsolete) most popular tech stacks for the web, very common in the later nineties and early XXI century. Popular software born originally from LAMP is for instance the Wordpress, Facebook and Joomla. It was composed by a set of four open-source, free, well documented and very stable technologies, with many low-cost options for deployment, for that reason it was very popular during the end of the Web 1.0 and beginning of the Web 2.0. It was one of the most popular choices for developing monolithic web applications during the Web 2.0 early era.

Nowadays this paradigm is totally obsolete. In the later XX century and beginning of the XXI century the connected world was very different from the contemporary one: at the time, only some households had a personal computer and it was the main entry point to the WWW. Nowadays with smartphones, IoT, Tablets, Smart

Watches, Virtual Assistants (etc.) the world is connected 24h. We are closer to a transition to the Web 3.0, in the era of the Big Data, AI and high volumes of traffic, monolithic applications based on the LAMP tech stack can't provide the necessary scalability, maintainability and high-availability needed in a digital interconnected world.

Many new tech stacks and architectures have emerged and evolved since then. For example LEMP (Linux, Nginx, MariaDB/MySQL, Python/PHP/Perl), MEAN (MongoDB, Express, Angular, NodeJS) and its variants like MERN (MongoDB, Express, React, NodeJS) or MEvN (MongoDB, Express, Vue, NodeJS), etc.

Most of these web tech stacks can be divided in three major vectors: Frontend (reactive interfaces, Single Page Application, AJAX, Progressive Web Apps, WASM, React Native, etc.), Backend (API Gateways, Cloud Providers, VPS, thousands of web frameworks and databases paradigms, etc.), and a third layer, wchich we will imprecisely call APIs, that connect the Frontend with the Backend communication (RESTfull APIs, GraphQL, Stripe, Twilio, Message Brokers, Advanced Message Queuing Protocol, etc.).

Nonetheless, this paradigm can't be applied anymore, because it only made sense in the context of the monolithic applications. The Web Engineering is moving very fast to new architectures and approaches based on Microsservices, in which one is not limited to a single tech stack anymore, being possible and recommended to develop the same product with has many tech stacks as needed to better express the needs of specific services. Even the Backend and API component has become increasingly complex with Cloud Providers, Knowledge Graphs, Service Discoverability, Service Meshs, new types of Message Brokers, Orchestration tools like Kubernets or Docker Swarm, etc. The same can be said of the Frontend component, where the concept of Micro Frontends (assimilated from the Microsservices paradigm) is emerging, or new revolutionary technologies like WebAssembly are testing the limits of the web.

The contemporary web ecosystem can't be anymore described solely with simplifications like tech stacks, but rather one should talk about architectural approaches and tech choices for each isolated service.

In opposition to the monolithic applications, a microservice architecture is a form of service-oriented architecture (SOA) in which software applications are built as a collection of loosely coupled services, in which they can communicate or be totally independent from each one or even be built in different programming languages, making development cycles and teams independent, but at the same time better contribute to the final product integrity (modularity equals easier maintenance and selection of the most adequate tools for each job). In the State of the Art chapter (2 we analysed this architecture benefits and particularities in more detail, we will not repeat it here, but will underline its importance for the modern Web Engineering challenges.

We opted for this microsservices architecture as the main design principle for the current product, because it was the best choice for the non-functional requirements and challenges ahead, providing flexibility, granular approach and loosely coupled services[SZH], as discussed before. And also, because this project is exploratory, and if we focused the traditional obsolete monolith tech stacks, even tough would be a lot easier, would not provide relevant new data and conclusions, we aim to experiment and provide new methodological data.

In Figure 34 we can see a macroscopic Data Flow Diagram, summarizing the designed main layers of the final application and their interactions. We target a cross-platform end-product, designed for the Web and Mobile, supporting multiple APIs access levels. Next, we will further explain each layer objective an analyse its inner works.

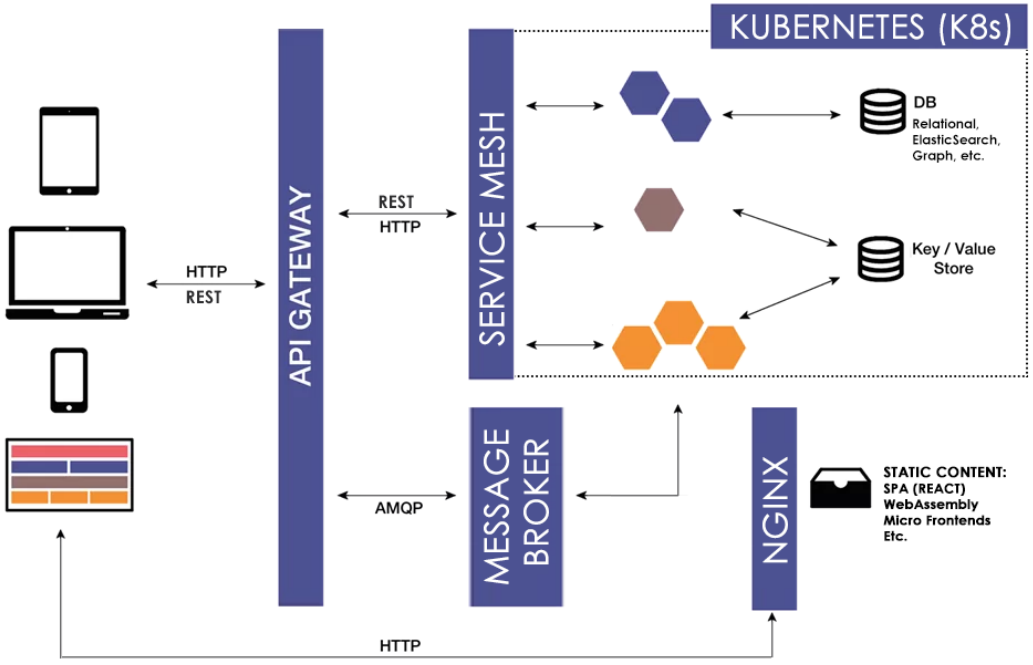

Figure 34: System overview

5.3.2 *Static Content Layer*

On the user side, the first level of communication will be a call to the main application website, through a static content layer, with a NGINX router able to provide a Single Page Application (SPA) (React and VUE based), providing the frontend to the user when a call is made to the main URL entrypoint (e.g. "https://www.website.com"). This layer also provides static content (e.g. images, documents, videos, etc.) when a call is emitted to a relative URL with a specific endpoint using a target key argument (e.g. "https://www.website.com/static/image/img.jpg"), or proxies and routers to other endpoints, or other special endpoints for misc content like the WebAssembly open standard or other reactive interfaces providers for running micro frontends in parallel (e.g. a VUE instance to run in parallel with the main one using micro frontends). This layer will be the first level of access for root domain calls, sending the reactive SPA frontend to the end-user. This layer can also serve as a load balancer and router, helping in replication mechanisms to secure high availability. We replicated each main component, and used techniques do secure its availability in case one fails, for instance, we have a slave replica of this layer, pinging the main one, and in case it goes offline, the slave one becomes the master one, securing high availability in case one master replica fails. In future, in case scalability needs increase, we would also be able to easily load balance the load of traffic between an increase number of replicas (e.g. using AWS Elastic Load Balancer or other methdologies), if needed (being it through orchestration tools and automatic pods or manual methods). Other mechanism that could improve future scalability, according to increasing demand and necessities, would be the integration of caching tools and CDNs on this layer.

The Single Page Application aspect of the application, exposed by this Static Layer, will further be discussed and present at the sections Frontend (5.5) and Mobile (5.6), related with the final interfaces. Initially a client, be it through a desktop or mobile hardware, first asks the server for a SPA version of the application, receiving one based on React (in the case of mobile, this SPA would already be cached in case of offline access, only checking the web and downloading it in case of a new version). This individual interface would reactively interface with the data (exposed through the API Gateway) and adapt. Besides the main React interface, we also integrated VUE into the application, to expose data from the developed tool EntiGraph (section at 5.7.1). Further sections will expand on this topic.

One of our main objectives during the design phase was to avoid single points of failure. Each layer should have mechanisms to secure availability in case one of the nodes fail, and for that replication and load balancing is a common technique. But one common problem when using an external load balancer point, is that it itself could become a single point of failure (e.g. if the server running the load balancer breakdown and goes offline), and incoming connections can't be distributed to the replicas. Containerization and service orchestration tools help minimize this when configured in a distributed cluster, but for this static layer, since it was the entry point with a very specific and simple function (to serve the SPA, proxy and route static content requests, etc.) we opted for a traditional and direct approach. We idealized to transform this layer into a Nginx load balancer, with replicas pinging its state and ready to divert traffic in case this node fails, securing high availability in every moment.

5.3.3 *API Gateway Layer*

The SPA reactive frontend, from the user side, will establish communication with the server through HTTP and/or HTTPS protocols using REST APIs. The first layer of access to these APIs will be through the "API Gateway" layer represented on Figure 34, through an endpoint with the "api" argument and its version as starting arguments (e.g. "https://www.website.com/api/v1/users/1"). At the moment we configured this layer to use a Kong Gateway container as API Gateway, routing all the API calls in persistent endpoints that abstract the internal server communication with the multiple microsservices and clusters. We also provide an endpoint with the REST API documented in SWAGGER OAS 2.0.

An API Gateway design pattern serves as a single entry point and proxy router between the Cloud services and the Clients. Among other things, it encapsulates the specific internal implementation and interfaces of the system[ZJJ18], being useful in the context of a microsservices architecture for avoiding service discoverability difficulties and to avoid direct access to individual services from the external Cloud environment, abstracting the communication and serving as intermediate layer of communication between the Client and the Cloud. It also adds a layer of security, by avoiding direct communication of the users with the microsservices layer, ensuring that these service only communicate internally server-side in a local context, never exposed to the outside world, only accepting calls directly from the API Gateway IP address (monitoring each layer by Firewalls and limiting the flow by specifying IPs and ports ranges and blocks). Lastly, it also contributes to an easier management of authentications and accesses, because in the context of the API Gateway one can define public and private endpoints, and the later requests will first have to pass through an authorization service and token based system

(JSON Web Token (JWT)) in order for the request to proceed - the API Gateway layer can keep track of active sessions and tokens and control this flow of access, including the management of groups access (for instance, some endpoints could only be accessed by users in an administration group, beta tester groups, etc.) or simply redirect the user to a registration, login page or block/message page, in case it doesn't have an active token with privilege of access to certain endpoints.

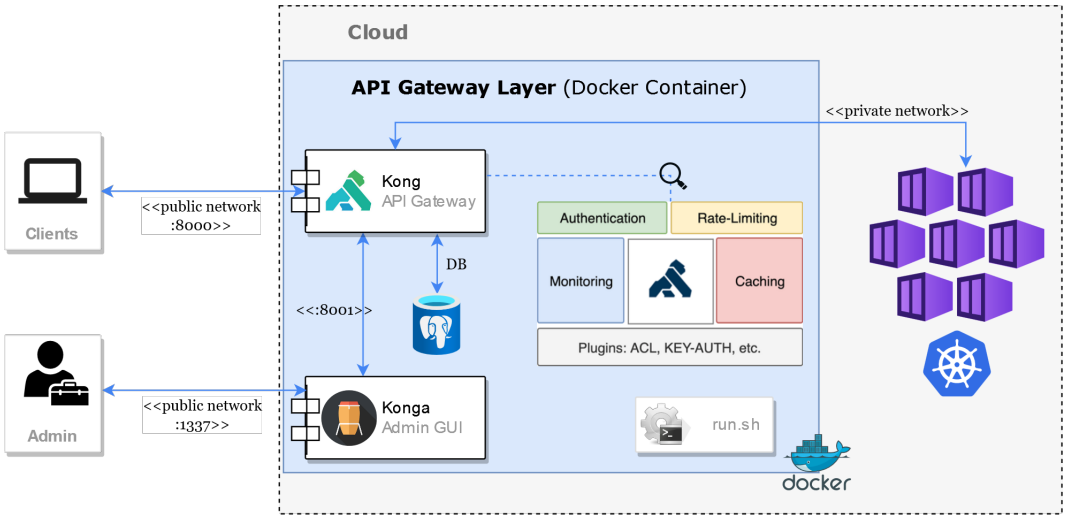

Figure 35: API Gateway overview

The chosen technological stack for this layer was the Kong API Gateway, a popular choice for APIs and Microservices management. There are many different GUI options to expose the management of this technology on the web through an administration page, we opted for the Konga web GUI because it was free, open-source, tested, intuitive and well documented. A PostgreSQL container ws defined alongside this stack to support its needs (configurations, logs, etc.). This stack was encapsulated and configured inside a docker container, using Docker Compose and YAML files to define it, exposing Konga on port 1337, and Kong admin API on port 8001 and its exposed endpoints in port 8000. then we generated configuration files for defining the services, routes, and authentication criteria. Figure 36 shows screenshots of konga's interface running.

During the development of this product, we tried to encapsulate every layer and individual services in docker containers, in order to make them easier to manage, deploy, replicate, test and integrate. Later, in more advanced design cycles, we transitioned from a simple Docker Compose to a Kubernetes in a cluster context, transforming these containers in replicated Pods - there are tools for converting the Composer YAML configuration files to the Kubernetes format (e.g. the Kompose tool), so we applied them to easily migrate from one context to another, ending up using Docker Compose initially in a local testing environment, and Kubernetes in the later stages of development and production.

To facilitate development and testing, we developed bash scripts in the context of each service called "run.sh" and "install.sh" (the GitHub repository is divided in folders by the layer name, and each sub-folder is a service, and inside each one of them we inserted these bash scripts), which respectively allow one to easily start the services (by running it on the terminal or double clicking after making the file executable) or easily install the

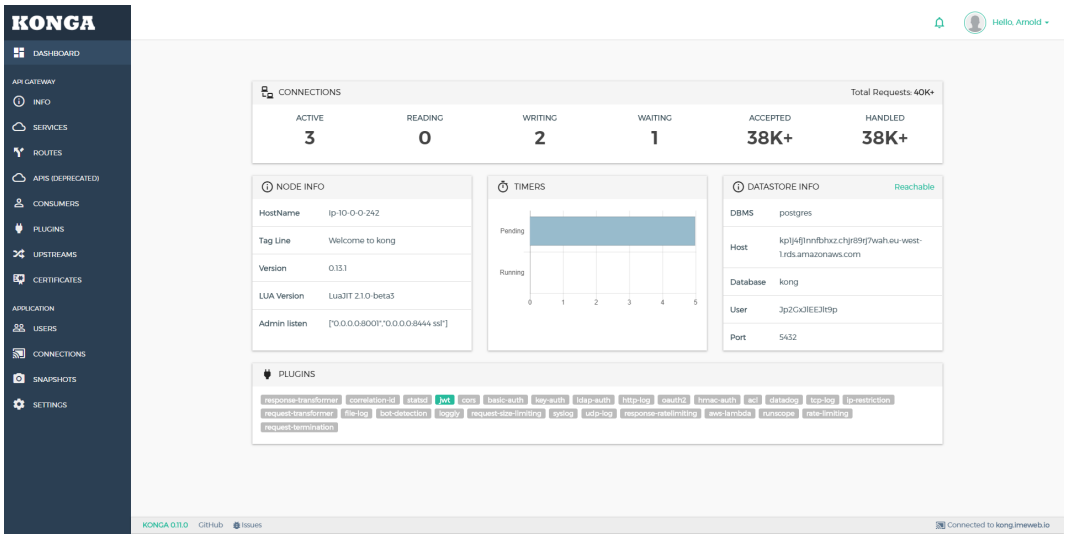

Figure 36: Konga interface - a web based GUI for the the Kong's API administration

container dependencies on the first run (provided that Docker is installed and running on the machine). Figure 35 summaries the tech stack adopted for this layer.

5.3.4 *Service Mesh Layer*

In order to improve the service discoverability, management and communication between the Kubernetes cluster and the requests, we implemented a service discovery design pattern through a Service Mesh. Even though microservices architectures have become increasingly popular, improving speed, agility and scalability of software through service delivery, it also significantly raises operational complexity and introduces many new challenges in the context of modern applications - of Service Meshs try to mitigate this problem by applying an abstraction layer over microservices, dedicated to service discoverability, without interfering with the underlying services implementations[LLG+19, EMZ19]. Kubernetes and similar Container Orchestration tools, allow, for instance amongst many other things, to easily and automatically scale services, by replicating them (pods) and distributing loads - this, even though rises availability, increases complexity by introducing service discoverability difficulties (we are not dealing with a single service with a well-defined network address and name, now we are dealing with an irregular number of replicas in an automatic scaling context). This can also introduce concurrency and communication issues, if the services were designed in interdependence with other instances, like in a traditional monolithic fashion, in a sequential blocking processing - this could lead to many concurrency, replication and/or loss of data, because we now have an irregular number of services competing for the same actions without a common global state[EMZ19].

Indrasiri et al. (2020) summarizes this complexity by stating that "microservices have to talk to each other and inter-service communication is one of the key challenges in realizing the microservices architecture"[IS18]. The Service Oriented Architecture (SOA) used to deal with this kind of complexities, by abstracting this kind of

problem, but nowadays, smart endpoints mandate that the developers need to deal with inter-service communication directly. Service Mesh is a new pattern that solves this complexity by introducing a distributed layer that that encapsulates inter-service communication.

A service discovery design pattern solves this problem, namely in our context through the use of technology called Service Mesh, which is an abstraction layer similar in some degree to an API Gateway, but instead of abstracting endpoints it abstracts microsservices, tracking and monitoring its

sate, health, statistical data like load or uptime, facilitating also its discovery by serving as a single entry point. According to Pretzer (2021) "a service mesh is an abstraction layer that binds together microservices-based distributed systems by providing observability, security, and reliability"[Pre21].

Indrasiri et al. (2020) also summarizes this complexity by stating that "microservices have to talk to each other and inter-service communication is one of the key challenges in realizing the microservices architecture"[IS18]. The Service Oriented Architecture (SOA) used to deal with this kind of complexities, by abstracting this kind of problem, but nowadays, smart endpoints mandate that the developers need to deal with inter-service communication directly. Service Mesh is a new pattern that solves this complexity by introducing a distributed layer that that encapsulates inter-service communication.

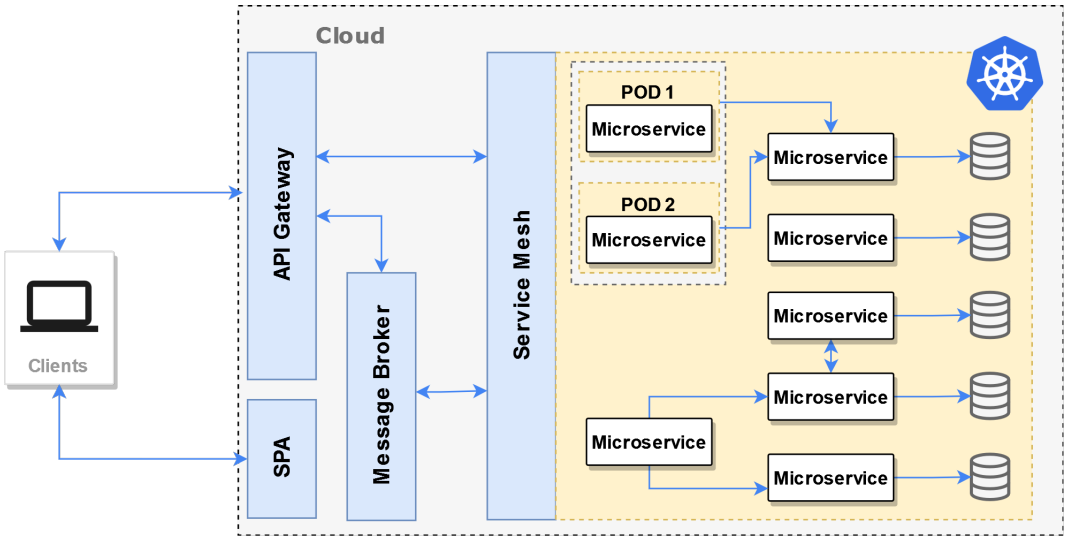

Figure 37: Technology stack with a Service Mesh

API Gateways usually deal with static services and endpoints communication, on the other hand Service Mesh deal with services discoverability and inter-service communication. Nowadays many API Gateways try to also to incorporate and offer Service Mesh like functionalities, in order to expand to better cover and expand their concept to the microservice architerctures, for instance our implement API Gateway, Kong, recently also offers an extra tool called Kuma that serves as an Service Mesh Layer. But it was in beta state until recently, and we think that these kind of hybrid options are not as mature as other enterprise proven solutions like Istio (backed by google, 1st in popularity) or Linkerd (the fastest solution on the market, made using Rust, and the 2nd most popular). For this reason we opted for Linkerd because even though it is the second most popular Service Mesh,

behind Istio, its second version is more recent, performative and easiest to maintain. It was also rewritten in the Rust programming language, making it the most performative in benchmarks, and Rust is one of our core technologies in study for this project, so it was our predominate choice.

Figure 37 showcases an example of our architecture with a Service Mesh Layer, mediating service discoverability (note: the microservices instances are just an example and we didn't specified a pod number like "Pod 1" in the case of a single instance without replicas).

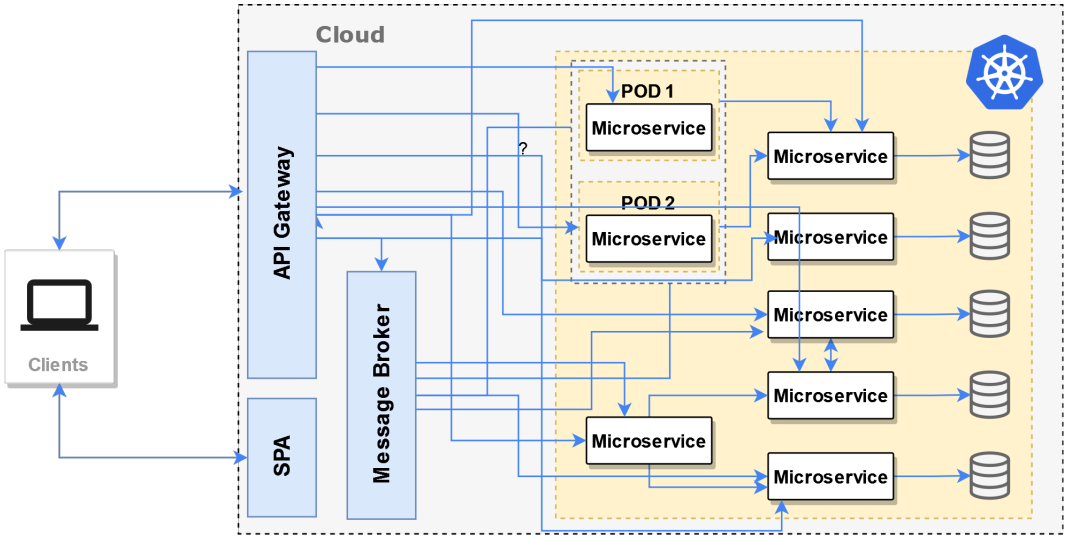

Figure 38: Technology stack without a Service Mesh

At the other hand, figure 38 is an hypothetical hyperbolized example of a scenario without a Service Mesh layer, in which the API Gateway, Service Broker and outside world would need to communicate directly with each microservice, knowing the state and where each one is located at each given time in the distributed cluster network, including each dynamically generated replica (which is the problem the Service Mesh tries to mediate).

5.3.5 *Kubernetes Layer*

A microsservices architecture, amongst other things, allow for loosely coupled, lightweight, agile, reuse oriented services, in opposition to monoliths, eliminating technology lock-ins or teams inter-dependence, while stimulating a cleaner code base, easier to maintain (services Independence from each other) and better scalability between components. For instance, its modular approach avoids the need to recompile complete big monolith products even when simple modifications are introduced, isolating important services from each other, or avoiding blocking team development by other teams/managers technological choices.

There are many different design patterns for applying this kind of architecture. We followed some common principles like:

- **Single-responsibility units:** Individualize databases, and for each one have a dedicated service or sets of services. Each one of this services have an individual IP and instance running.
- **Replication:** Clone services in more than one instance and design them in a way that this cloning doens't generate data races or conflicts, in order to secure high-availability in case of one failing, for allowing scalability and load distribution.
- **Design Principles:** we followed the IDEAL pattern (isolated state, distribution, elasticity, automated management and loose coupling) and the twelve app factors used by the Heroku's method, in conjugation with the 3Factor App proposed by Hasura[3fa20] (see the State of the Art chapter 2 for more details).
- **GraphQL:** following the 3Factor App methodology we included a GraphQL based API.
- **Async Events:** many authors recommend async events for request/response management. There are many patterns, we followed a Message Broker pattern based on AMQP, using RabbitMQ and Redis based queue systems for managing message flow and processing.
- **Serverless Functions:** Serverless Functions are becoming very popular as an alternative microsservices pattern architecture, because they economize many resources and money, but we find that they yet face many challenges (namely cold starts that result in a performance hit). For that reason we avoided them, implementing a more traditional microsservices pattern, only using them if a specific service would not suffer from a start/stop performance penalty.
- **Multiple computing paradigms:** we build the services in a way that they didnt depend on each other technology to communicate, and used the most adequate technology in the context of individual services according to needs, using tech flexibility increases the quality of the product.
- **DevOps and decentralized delivery:** we built each service autonomously, allowing for independent development and delivery, implementing Devps best practices.

We would also like to cite Zimmermann (2017)[Zim17] that made an excellent work collecting the State of the Art of these technology and compiled a list of 7 tenets of good practices, we tried to follow most of them.

A Kubernetes cluster is a a set of nodes that run containerized applications distributed in a network of servers, effectively distributing the processing load and managing smartly nodes failures, or high-availability by using replication mechanisms (example: one service could have 5 replicas in a cluster, if one fails in a part of the cluster the others can continue the work, and a load balancer distributes automatically the traffic load to them). But this distributed replication based architecture introduces many new challenges, mainly at the level of communication (synchronous communication or dependencies, for example, could result in many problems in a replication context). For that reason we also included a Message Broker layer (that we will explore in the following sections).

In the context of Kubernets, we explored two three different flavours: initially we explored and configured Kubernets vanilla in a test server, but for testing environments we found it too complex and time consuming;

next we explored abstractions, that allowed for building easier testing environments, and ended up exploring Microk8s[1], which are a Ubuntu tool integrated through its Advanced Package Tool ("apt-get microk8s"), with a single command all the dependencies needed by Kubernetes and a pre-configured environment where abstracted and available, whit a basic abstraction layer to start and stop it as a service (it was a very interesting choice, but in later stages we needed to work with Operation Systems beside Ubuntu for testing on dedicated physical servers that were available to us (using Fedora for example), so we ended up abandoning Microk8s, but we liked it very much for testing purposes and deployment if using only Ubuntu); finally we explored Rancher[2] and opted to use it with great success: it is the same concept as Micork8s, offering an abstraction layer to manage it, but it runs Kubernets inside Docker itself, allowing for a lot of flexibility for testing and even for deployment, offering also a very good web interface for management with plugins and the most common tools used with Kubernets already pre-configured (telemetry tools for example) that allowed one to get clusters up and running with great interfaces and management abstractions, combining CLI with a GUI. At the moment we are running our Kubernet cluster with Rancher with great success (we tested it locally and with 3 dedicated physical servers with success). Figure 40 showcases an example of the main interface offered by Rancher, and Figure 39 is a summary of Rancher main functionalities and objectives (from their main documentation).

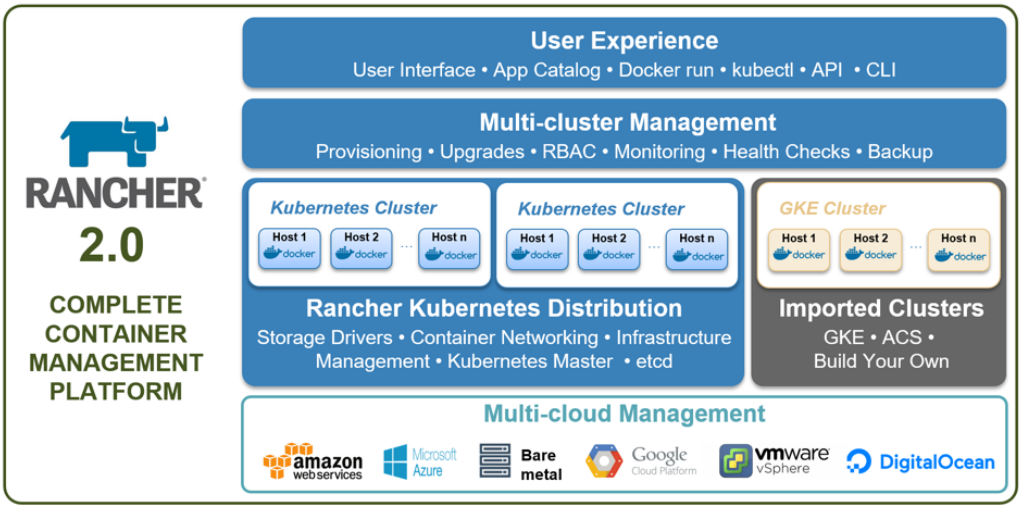

Figure 39: Rancher objectives

This Kubernet layer, beyond the mentioned technological stack (the other layers), had all the microsservices of the product as pods (besides some of the other layers, for example the Service Mesh layer that also ran instances under Rancher K8s).

We opted to have one microsservice per individual database singular responsibility (accordingly to best practices), and secondary microsservices for support tasks.

1 https://microk8s.io/
2 https://rancher.com/

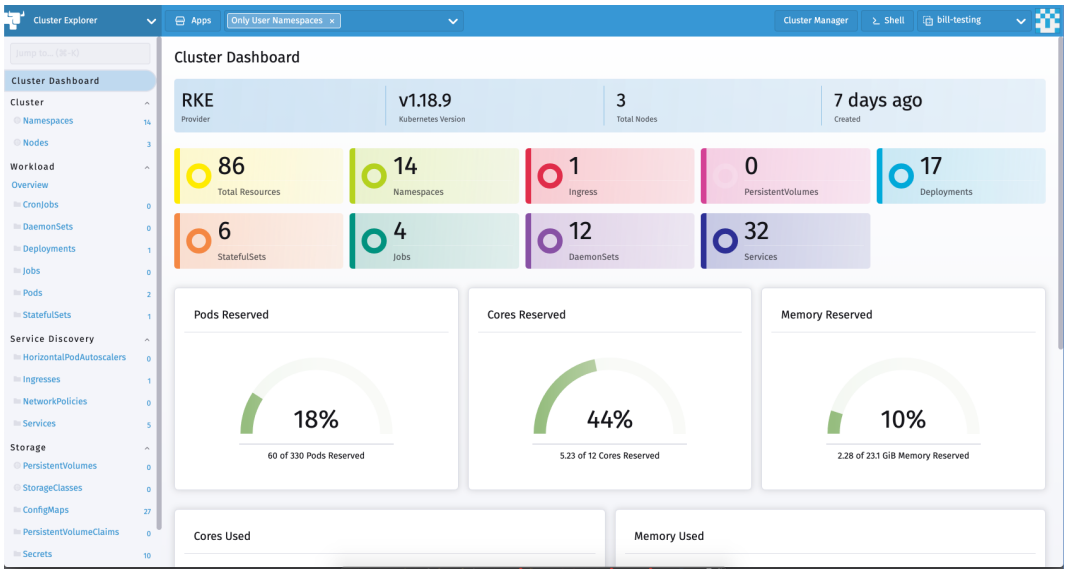

Figure 40: Rancher main user interface

In total, we defined these main microsservices (each one could have secondary supporting ones, but these were the main entry points):

- **Auth:** a service dedicated to authentication.
- **Users:** a service dedicated to management of user data (profile, user name, etc.).
- **Posts:** a service dedicated to management of the posts (read, write, update, interactions etc.).
- **Assets:** a service dedicated to management of the assets (read, write, update, interactions etc.).
- **CV:** a service dedicated to management of the CV/Portfolio export functionality (compilation of a CV/Portfolio for exporting).
- **Communities:** a service dedicated to management of communities data.
- **Comments:** a service dedicated to management of user comments.
- **Followers:** a service dedicated to keep track of users graph of followers.
- **StarDog:** a service dedicated to exposing RDF triplestores and SPARQL APIs.
- **CockroachDB:** a service dedicated to exposing a cluster of databases based on CockroachDB.
- **EntiGraph:** a service dedicated to exposing a tool developed for this project (derived from the Major Minors project and also integrated in the NetLang R&D project), that accepts text as input, outputting identification of entities and RDF lines used to build Knowledge Graphs.

Each one used different backend technologies (complemented with NIGNX when necessary), some used Rust and ActixWeb, others Flask, one tool used NodeJS, but the majority used Django.

The Auth service is the entry point of the system, only after creating a successful authenticated session with a token would a user be able to access the rest of the application. The API Gateway keeps track of active sessions and tokens at any given time, and would give access to the application in case of the token being active.

The other services are contextual to each main product need, and they reflect the database main structure: we avoided relational tables when possible, avoiding interlinked tables like the SQL paradigm, so that each service and his related database could be totally independent from each other. Our first database design was relational, but soon we opted for a NoSQL style of design, to stimulate this independence and avoiding future problems (dependence between services or sharing of databases between more than one service would result in problems in the long run and bad design practices). In the following sections we will better explore the database structure.

Each service has a single responsibility towards a single database. Its objectives are clear in the above descriptions and service name.

Besides this main services, we have other supporting generic microservices running, for example we a service with a Triplestore databse (ontology graph database) running an instance of the European Skills, Competences, qualifications and Occupations Ontology (graph database from the European Union). We used StarDog to expose this ontology (RDF/Turtle), in a Docker as a service, configured to expose an SPARQL API in a specific port - the product web application React frontend uses this port and API to show communities and skill sets in the form fields, with real-time text search and feedback recommending results. This ontology structure is also used for the general search fields used in various points of the platform and to better organize filters and the CV/portfolio exporting tool.

Other supporting services include, for example, the EntiGraph tool that we developed, inspired by the Major Minors project[3] also developed by us, to automatically extract/identify entities in posts and build ontologies based graphs around them. In the following sections we will better describe this tool (it was developed using Rust, REST servers, Python, Regular Expressions, YAGO ontology project, VUE interfaces and the Oxygraph Rust RDF interpreter. A local server instance is always running as a containerized microsservice in a specific port, receiving input as text (from posts) and outputting new RDF segments updating the ontology.

Each service is configure with replication, health check, load balancers and telemetry mechanisms, to secure high-availabilty and scalability. In the following sections we will further expand on supporting layers alongside this one.

5.3.6 *Message Broker Layer*

In a system composed of dozens of components/services that continuously communicate with each other, delays or failures in processing one call in a sequential execution could mean interruptions throughout the system, for this reason idealizing our system based on asynchronous messages and event managers was important

3 http://minors.ilch.uminho.pt

to ensure effective communication, availability and even future scalability. For this reason we implemented a Message Broker layer to our system, to give it asynchronous qualities and better communication management. In the State of the Art section (2) we already covered the concept of a Message Broker, in the current on we will briefly explain how we achieved and implemented it.

Mainly, we opted for implementing this layer with RabbitMQ[4] and Redis based Python libraries targeted at job queues (mainly Py-RQ[5]. Basically, RabbitMQ worked as a Broker, receiving, saving and managing each request, and the Celery and/or Py-RQ worked as the consumers/workers processing them (dividing/managing tasks by they priority and time consuming level, or scheduling non priority events for low traffic moments for example). Our technological choice was inspired by the Instagram architecture (see Rick Branson 2013 presentation [Bra13a] and his slides [Bra13b]), in which they also maintain a Message Broker layer with RabbitMQ supported by Celery (Rick Branson describes that prior to that they used Gearman as Message Broker but migrated to this new paradigm). They also use a Python stack similar to ours, with Django as backend, like most of our backend services, for that reason we got inspired by this choice. Celery and/or Py-RQ are great tools to easily give Python services asynchronous qualities and better manage heavy loads of traffic, without loosing any request - besides working as job queues, they also can serve as loggers and replication mechanisms, maintaining data consistency.

There are many options on this segment, for example Kafka is one of the most popular choices in higher tier businesses, incorporating Broker and Worker services, but after testing we found it to complex for its benefits, when RabbitMQ and Celery integrate so well and accomplish the same in most of cases with much more flexibility (Instagram deals efficiently with millions of requests each day, that was enough benchmark to us). Besides, with a Python based backend stack based on Django and Flask (some services use Rust based backends, but the majority are Python based), interoperability becomes more efficient. If, for example authentication, list was not important and direct RESTfull access was enough, this layer was immense important mostly for dealing with I/O and database intensive operations, better scaling, logging and scaling each process, better distributing the load through different workers and keeping track of each failed attempt and scheduling retries. Figure 41 showcases an example of this kind of job queues and service distribution tasks. Last, but not least, we end this initial roadmap of the main core architecture with an overview of the most important technologies used (Figure 42). In the following sections we will further explore the interfaces (frontend and mobile), databases, and semantic tools developed.

5.4 DATABASES

As mentioned before, we opted for a pattern where each microsservice had a single responsibility over a single database (a service per database). Following good practices, we stimulated async event communication between the outside world and the microsservices, but avoided inter-service communication and dependency when possible. For that, we opted to design the tables of the databases without much dependence between

4 RabbitMQ: https://www.rabbitmq.com/
5 Py-RQ: https://python-rq.org/ and Celery[6]

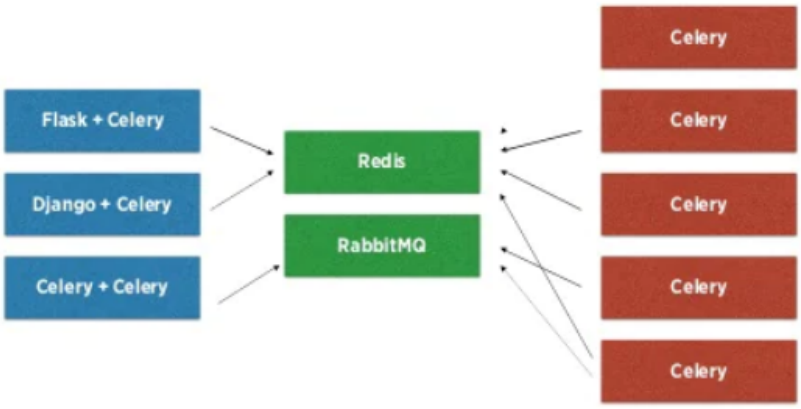

Figure 41: Message Broker: example of job queues and distribution of tasks through RabbitMQ/Redis as Message Broker and Celery as workers in a Cloud infrastructure

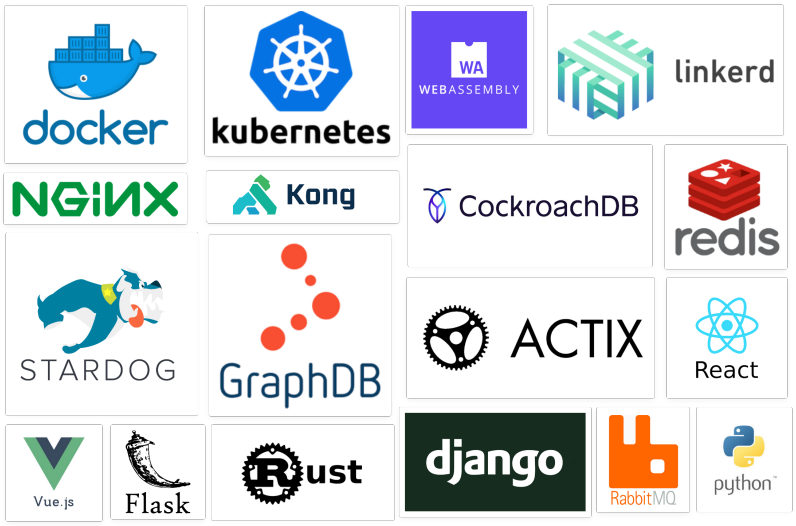

Figure 42: Tech stack: some of the main technologies implemented

each other, and for that we opted to not used relation databases design paradigms, and opted for an agnostic one, close to a NoSQL style. This way, initially we designed the database as one, but later in implementation separated each cluster of tables accordingly to a single responsibility rule, transforming each one in a single isolated database, ending up designing the microsservices accordingly to these databases and both influencing each other design. At the moment, these are the main microsservices that we created and, accordingly, each one represents an individual databases with the same name:

- **Auth:** a service dedicated to authentication.
- **Users:** a service dedicated to management of user data (profile, user name, etc.).
- **Posts:** a service dedicated to management of the posts (read, write, update, interactions etc.).
- **Assets:** a service dedicated to management of the assets (read, write, update, interactions etc.).
- **CV:** a service dedicated to management of the CV/Portfolio export functionality (compilation of a CV/Portfolio for exporting).
- **Communities:** a service dedicated to management of communities data.
- **Comments:** a service dedicated to management of user comments.
- **Followers:** a service dedicated to keep track of users graph of followers.

Each service has CRUD functionalities over its database and dedicated support microssevrices for individual actions (for example: to increment the vote rank of a post, or to reduce it by one value, etc.). Figure 43 showcases a Data Structure Diagram of the generic database idealized, later separated into individual databases accordingly to the microsservices and single responsibilities.

Note that the design stimulates an independence between each table, allowing for easily dividing into autonomous databases. In the initial design cycles our model was inefficient at that (see Figure 44 for an abandoned design), so we had to redesigned it. In this initial abandoned model, it is worth noting the inefficiency in the possibilities for join of data regarding the three types of user publications/posts. Current models have been refined to meet demands more efficiently via a centralized "Post" abstract aggregator table that relates to its three variants.

For each database we used different technologies, accordingly to its necessities. We used in many cases the CockroachDB[7] (distributed database for clusters, SQL based, heavily inspired by Google's Spanner) for extensive I/O services, and NoSQL paradigms (MongoDB for example) for particular contexts (but aiming to transition soon to the ArangoDB cluster multiparadigm databse system), and in particular contexts like graph databses for ontologies we used StarDog (for exposing the ESCO ontology), Oxygraph for the EntiGraph tool service, and GraphDB triplestore for inner graphs related with user data.

We used in many cases Djago as the main backend technology, and it has a very good Object Relational Mapping (ORM), for that reason and to make it easy to conduct migrations and testing for other database paradigms,

7 https://www.cockroachlabs.com/

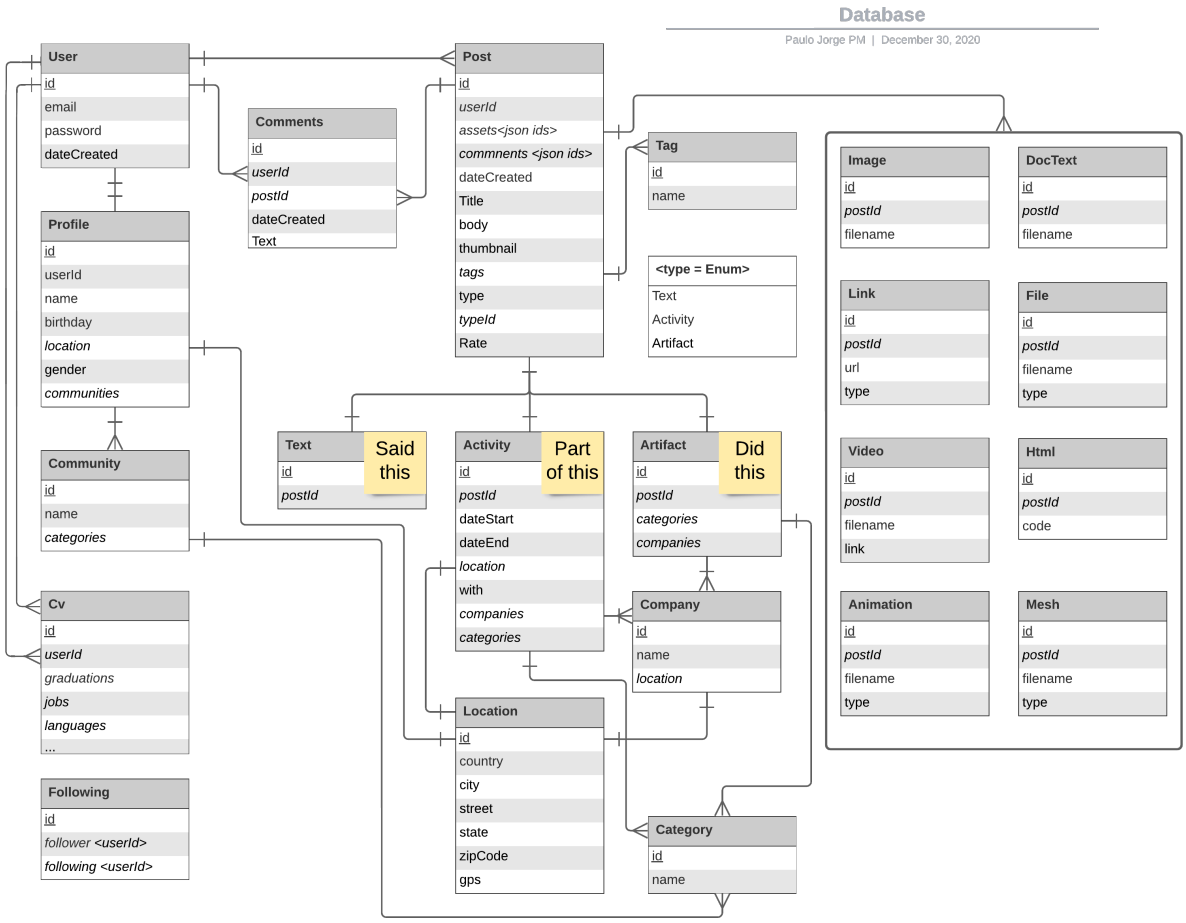

Figure 43: Final Data Structure Diagram (DSD)

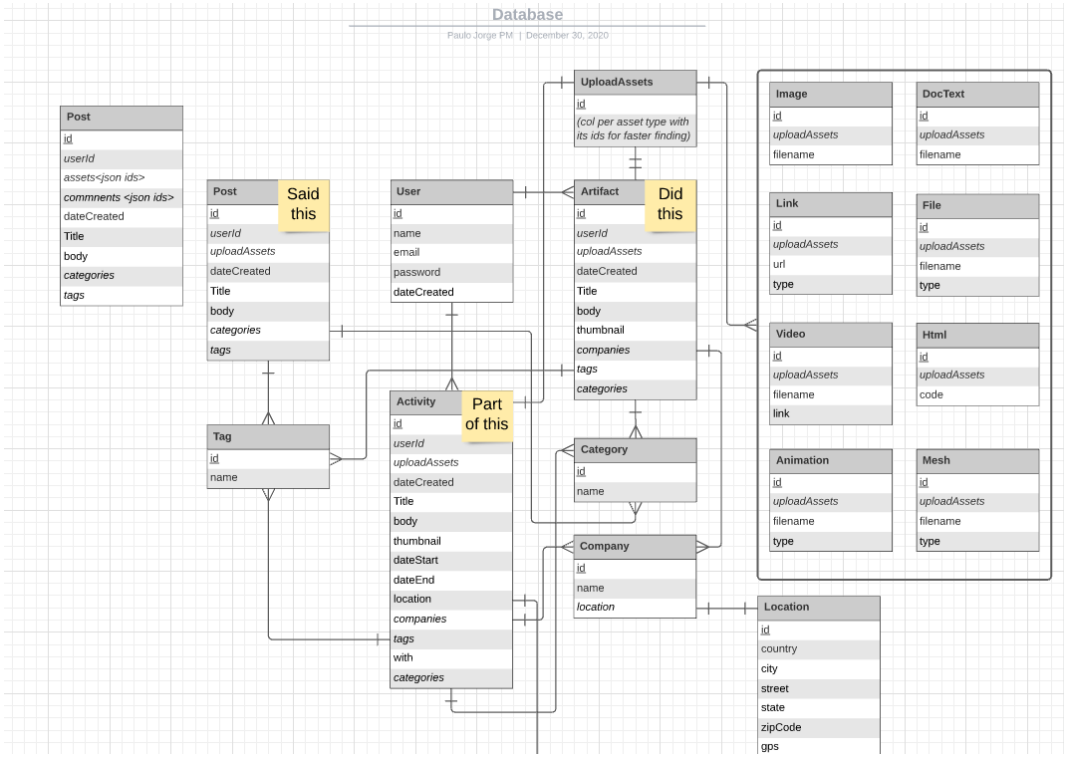

Figure 44: Initial Data structure diagram with limitations (abandoned) (DSD)

we used it extensively to abstract database operability. CockroachDB had a stable and tested plugin for Django, to make its ORM compatible with it, this encouraged us to use CockroachDB (initially we were considering Apache Cassandra, but the better compatibility of CockroachDB encrouraged us to experiment with it - in the future the ORM makes it very easy to migrate to other paradigms). Besides thsi compatibility, what encouraged us to use CockroachDB was his cluster paradigm (it was used to be a distributed database across several servers), and it easy itnegration with Kubernets (Rancher supports it natively and has compatible docker instances pre-configured) - with it we had our data secure in a high-availability and high-integrity context, if a server failed we had it distributed and available in other place of the world. This database was extensively tested and used for popular companies, and benchmarks were very positive, Cassandra had better results overall but they were very close, and CockroachDB surpassed it in some I/O operations.So we opted to experiment with it. Figure 45 showcases an example of the interface (provided by CockroachDB) that we have available in our Kubernetes cluster to manage CockroachDB and check its health status across the cluster.

Regarding the used Triplesotre graph databases used, Figure 46 showcases an example of graph view of the interface of the StarDog database running we the ESCO database.

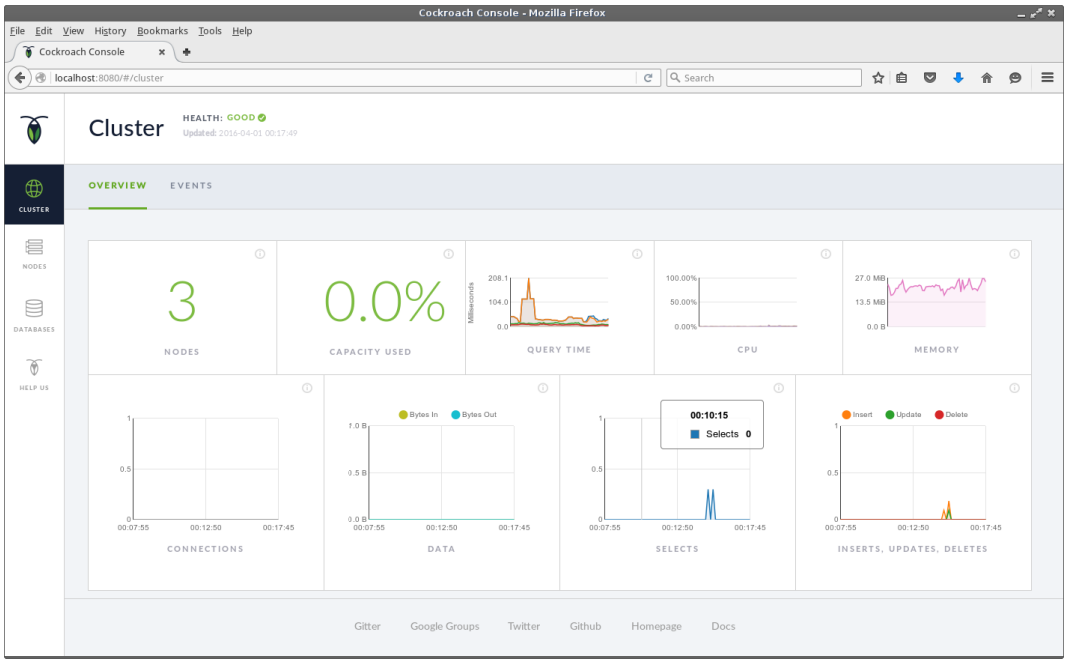

Figure 45: CockroachDB main interface

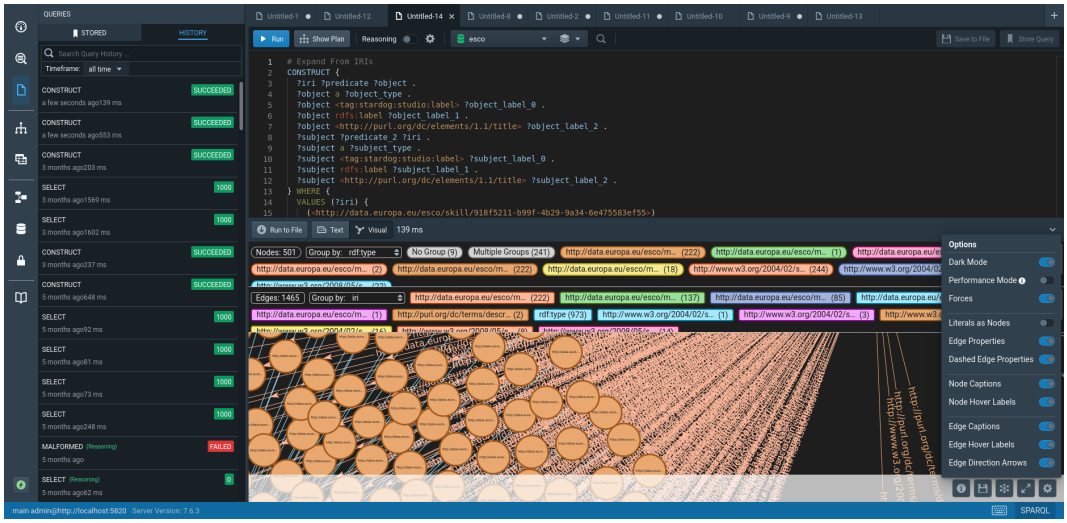

Figure 46: StarDog triplestore DB with ESCO ontology

5.5 FRONTEND

As stated on past sections, besides a mobile version prototype, we mainly focused development on a desktop web frontend interface. It is reactive and responsive. Reactive because it is a Single Page Application (SPA), loading an initial interface with states that will adapt to reactions from the user and automatically adapt to requested data and correctly adapt to display it - a web reactive interface is non-blocking, asynchronous and event-driven. It is also responsive, because it adapts to different screens through different media queries target at different peripherals, able to adapt to any resolution scenario, from the common 1920×1080, to greater ones like 4k or smaller peripherals like tablets and smartphones.

We mainly used the javascript React framework to build the frontend interface, sub-dividing it in modular components, together with React Redux, React Router and Material-UI to improve its capabilities. In the search page we also partially integrated VUE, because initially we used it to build the interface to expose the EntiGraph (developed tool) data. For that we used Micro Frontend concepts, in order to efficiently integrate a VUE interface inside a React base.

The client first request would load the React Single Page Application (thought the Static Layer exposed trough a NINGX router server side, explained in the past sections regarding implementation 5), and it would then communicate with the server infrastructure through RESTfullAPI calls to the core server. The first layer was authentication, thought the API Gateway, and after a successful token was confirmed and in place further requests would be able to communicate with the exposed API.

As exposed on past sections (mainly on the chapter E-portfolio as a Social Network at 4 and Implementation at 5), we aimed at simplifying the concept of an e-Portfolio to its most basic common denominators - the majority of applications related with this subject that we analyzed over-complicate this concept, and tires the users with complex forms and workflows. We planned on bringing the familiarity of already known applications and patterns to the users (e.g. Social Networks, mainly LindedIn since it is our main competitor) and mix it with the e-Portfolio concept, introducing new ways of interacting with Social Networks and exposing creative works, jobs and skills to communities. Initially we decided that this interactivity should be intuitive and simplified, we discarded everything non essential from the e-Portfolios most common designs and flowsharts presented on the State of the Art (2), and, as stated on chapter 4, we compiled a solo form, with the most basic elements needed to define an e-Portfolio and automatically generate also a CV from it. As most extensively explained before, we opted to divide the form into three sub-types: "Did this", "Lived this", and "Said this", both with a common form baseline, augmented by the ESCO ontology, but giving us the needed insight to better categorize and filter the user data as useful data (also augmented by by the EntiGraph tool, developed to extract semantic data from posts and building Knowledge Graphs around it). Figure 48 showcases a screenshot of this final form running, Figure 47 showcases a screenshot of the frontage wall of posts running with the users publications, Figure 49 showcases a concept prototype of the CV template generator that would allow users to exhibit and download highlights of their data in a traditional milestones CV one page format, and Figure 50 showcases a screenshot of the communities page in which users can build communities around their skills and creative interests. We also have the usual pages for login, registration, settings and profile editing, EntiGraph interface (Figure 58 in section 5.7.1), friends

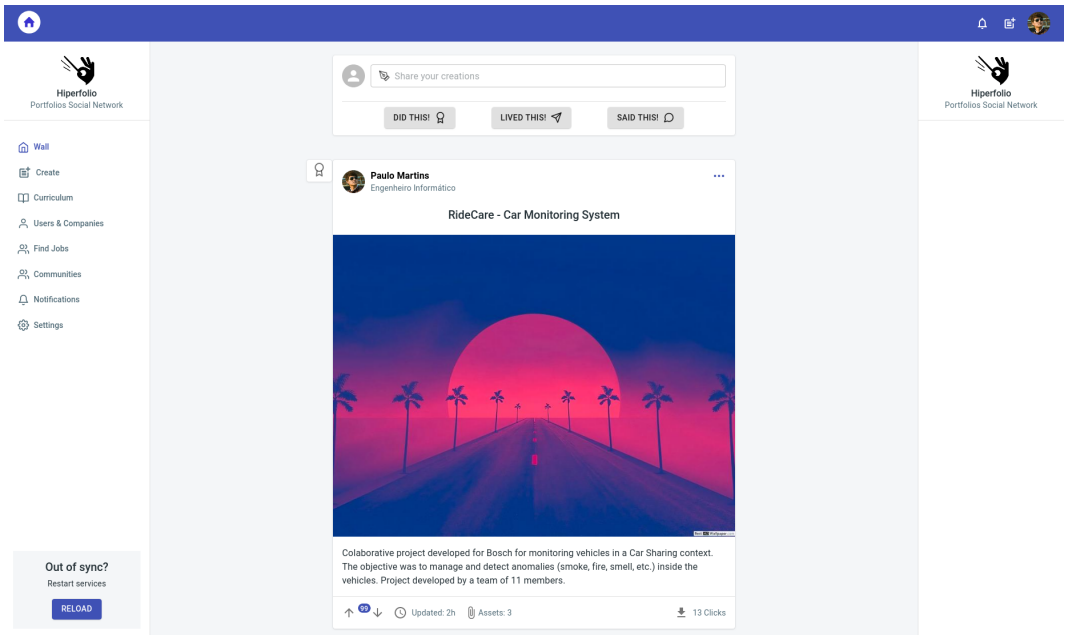

Figure 47: Frontend: main page, wall of posts

search and add page (etc.) but we don't want to make an gallery of screenshots, we would rather encourage you to check the Outputs section (6.1) with a summary of the produced data and the appendix section with hyperlinks to the the GIT repositories and source-codes.

This interface is still in production, target for releasing as a final real product soon after extensive testing and a defined marketing strategy. Currently, in the next chapter, we will analyse survey data testing the interface usability.

5.6 MOBILE

The development process of the UI/UX started with the mobile app (Android) in a webview context, because it would be easier to later expand the frontend to higher desktop resolutions instead of the other way around - we started with mobile resolutions, and incrementally adapted and designed to various intermediary levels of resolutions, testing various responsive scenarios until the most common target of 1920×1080 and beyond.

Our main target was to produce a web product, but since nowadays the web has a strong presence not only on the PC but also on various peripherals like tablets and mobile, we also opted to build a dedicated mobile app. In order to contribute to the initial goals and methodologies of this project, we opted for an exploratory approach, centered on emergent web technologies, namely the low-level Rust programming language and the development of new techniques to implement webviews.

Currently the mobile environment evolved a lot from the traditional native stacks (Java, Kotlin, C, etc.), and offers many alternative approaches and new methodologies, for instance: React Native, VUE Native, Flutter (Google's UI toolkit), Xamarin (C# mobile framework), Kivy (Python mobile framework), Progressive Web App,

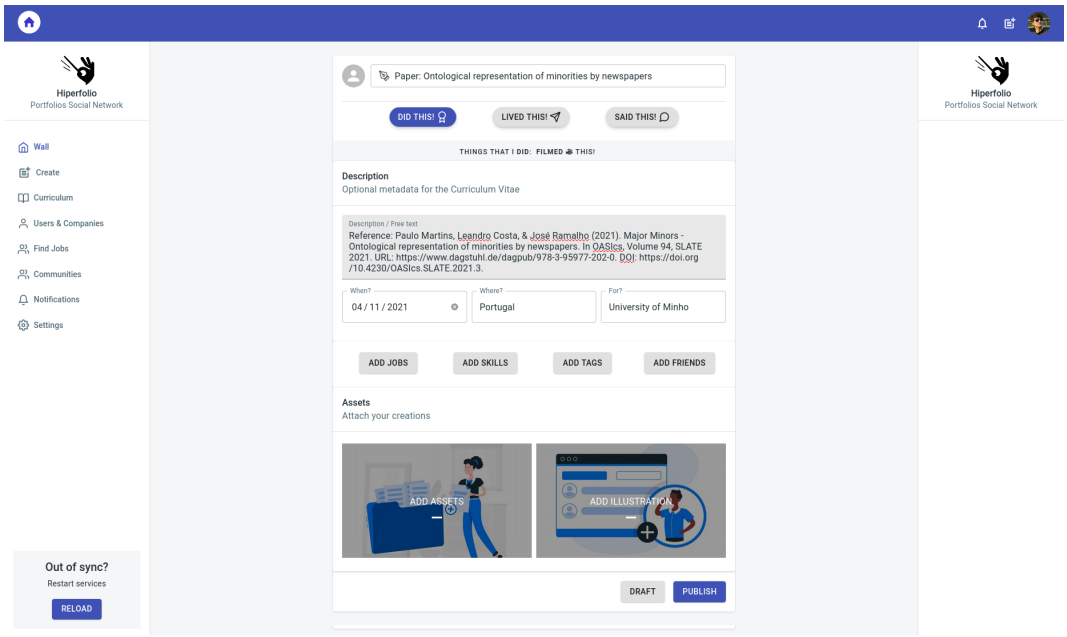

Figure 48: Frontend: page with the form for adding new content

Figure 49: Frontend: idealization of one of the CV templates page

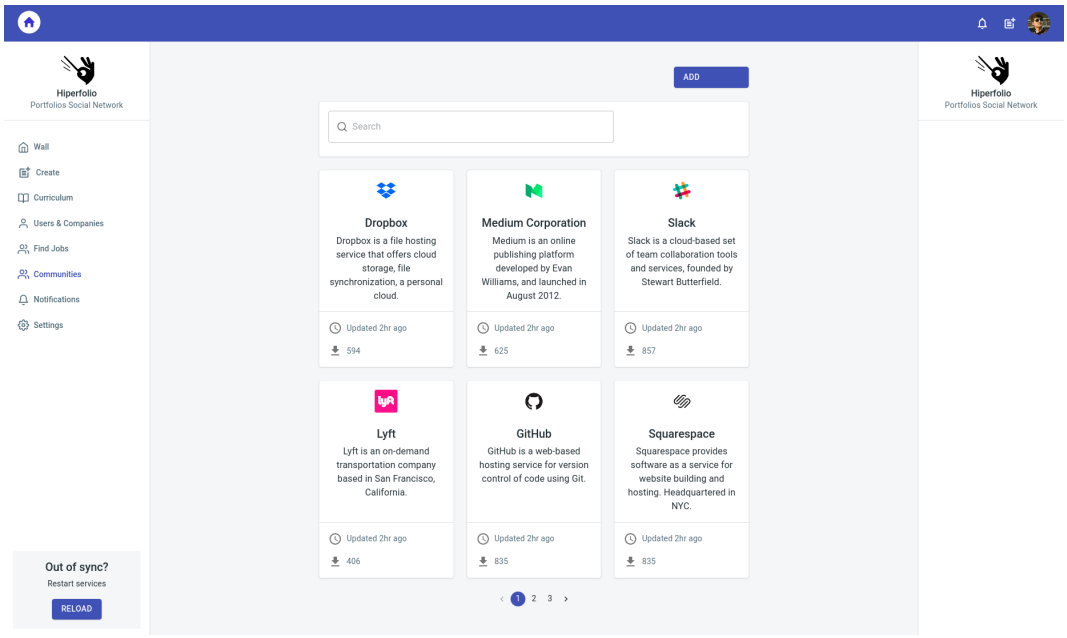

Figure 50: Frontend: communities page

Webviews, etc. Most of these solutions can be divided in three main groups: Native Apps, Hybrid Apps, Web Apps - each one as its advantages and disadvantages. Native Apps are built on specific languages depending on the target system native kit, for instance native iOS apps are built in Objective-C or Swift and Android native apps are usually built in Java or Kotlin - this can result in improved performance and reliability or a native feel, but lack flexibility or cross-platform support. Web apps are websites that look and feel like native apps, being one of best cross-platform options with immense flexibility and being easier to start (e.g. PWAs don't need to be installed), but in some cases can suffer a performance penalty or difficulty in accessing native functionalities. At the other hand, Hybrid apps are a balanced middle ground between Web and Native options offering a little from each side but never the full spectrum, usually allowing a flexible integration with Web or other languages abstractions inside a layer that embeds it in a native runtime.

For this project, initially we developed and performed tests on React Native running on a physical Samsung Galaxy A22 mobile phone (see Figure 51). Our web product is a Single Page Application built using the reactive interface React, so the mobile version of it using React Native would be an easy and direct conversion, and it was a fast prototyping option, so initially it sounded a good option. We started by developing an early and bare bones running prototype (available in the project GitHub, see appendix A). But this option was too easy and common, we would learn nothing new in the process and we would not provide new significant use cases and data for the community neither this study, so it would go against our initial goal and methodology based on Design Science Research, in which we aimed to and study a problem by iterating in a cycle of design and evaluation until we reached a product using exploratory and new methodologies able to provide us new insights and data onto the original problem - the only objective was not only a new end product, but also a novelty or exploratory approach. We concluded that the exploratory, less common or experimental approaches in this design cycle were the ones

able to provide the most novelty data and interest for the scientific community. For this reason, we opted to exclude React Native from the development cycle, because it was too common and there would be novelty on our approach, and instead we dedicated our research time to idealize a mobile app in a Web context made using Rust, a very recent and promising low-level programming language, that has been taking the world by storm. Rust has very few documented projects on mobile, and a lack of frameworks and abstractions for this system, due to it being very recent, but it easily compiles to any modern device with top industry performances (on pair with C and C++), was one of the first languages to compile to WASM with full support, and has an increasingly strong and unique ecosystem of libraries and strong adoption in the Cloud infrastructure, so it was an interesting experiment. Our logic regarding React Native, can also be transposed to Flutter, Dart, VUE Native, Xamarin, etc. - all are amazing and interesting technologies, but their use is so common and well-documented nowadays that it would be to easy to integrate them without a challenge - we aimed at exploring and providing novelty approaches and data for this study, walking the same path as everyone else would not be a very interesting choice. For this reason, a Rust mobile app was the new target.

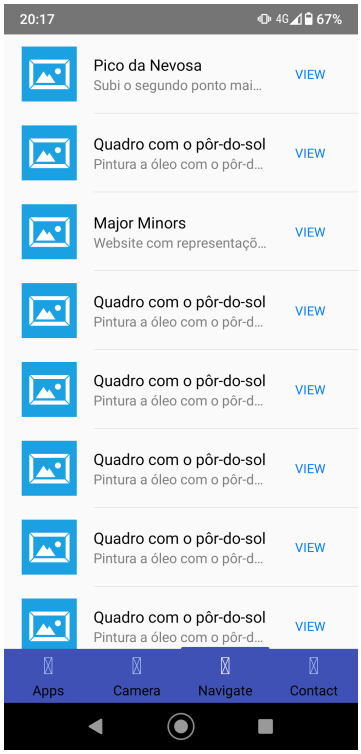

Figure 51: React Native first prototype running on an Android device

We wanted to achieve an app with the possibility of native performances (enter Rust, a low-level programming language with performances similar to C but the security of high-level languages), but also with the flexibility of an Web app - our philosophy is that web technologies are the GUIs Holy Grail, we love them as an universal flexible and cross-platform ecosystem, so we wanted to have them as a central part of our app. Our first design

choice was therefore to have a frontend Web layer (HTML5, CSS3, Javascript) running under a Rust native layer. Figure 52 shows one of the earliest versions of this new methodology running on a physical mobile device.

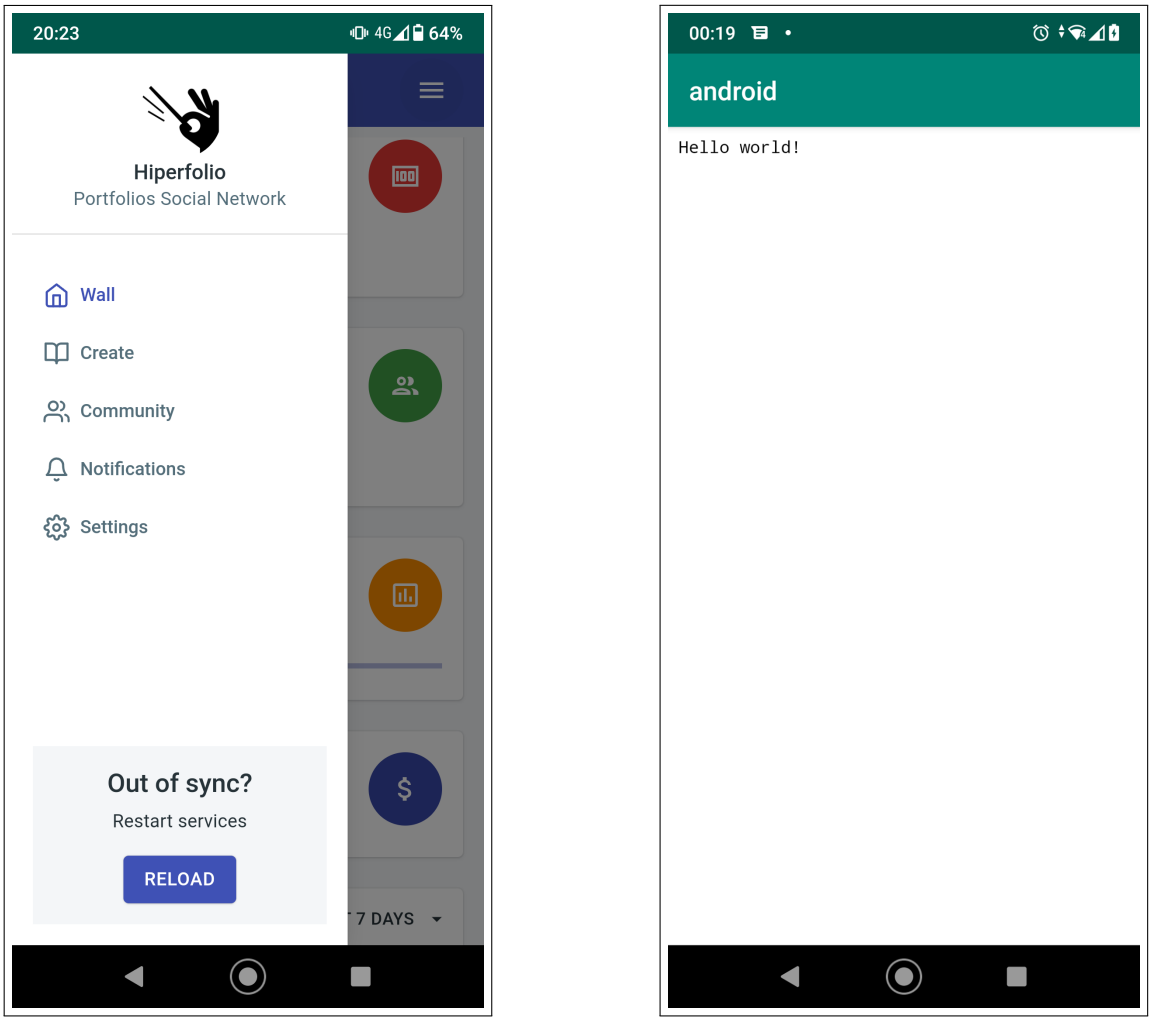

Figure 52: Left: Rust+Kotlin+React early prototype. Right: Hello World app Rust+Kotlin

At the moment Rust allows to compile directly to Android and has stable bindings to the Java Native Interface (JNI)[8] and Android NDK[9] - some of our initial prototypes explored this, compiling and running directly on Android from a Rust source-code, but they were abandoned in substitution of other techniques, but are still available at the project's GitHub (appendix A). We ended up using a mix, compiling Rust to Android and interacting with the system using the JNI and NDK bindings, but instead of running it directly, we opted to start this Rust compiled app from inside a native environment, namely a bare bones Kotlin app, that only has three steps: check if a pre-defined port is free on the device (if not check other until finding a free one), start a threaded Rust application inside the the main folder and pass it the selected port as argument, and then starts a Webview in an other thread opening a Single Page Application inside the assets directory (we opted for a reactive React frontend, but the

8 Rust JNI: https://github.com/jni-rs/jni-rs
9 Rust NDK: https://github.com/rust-windowing/android-ndk-rs

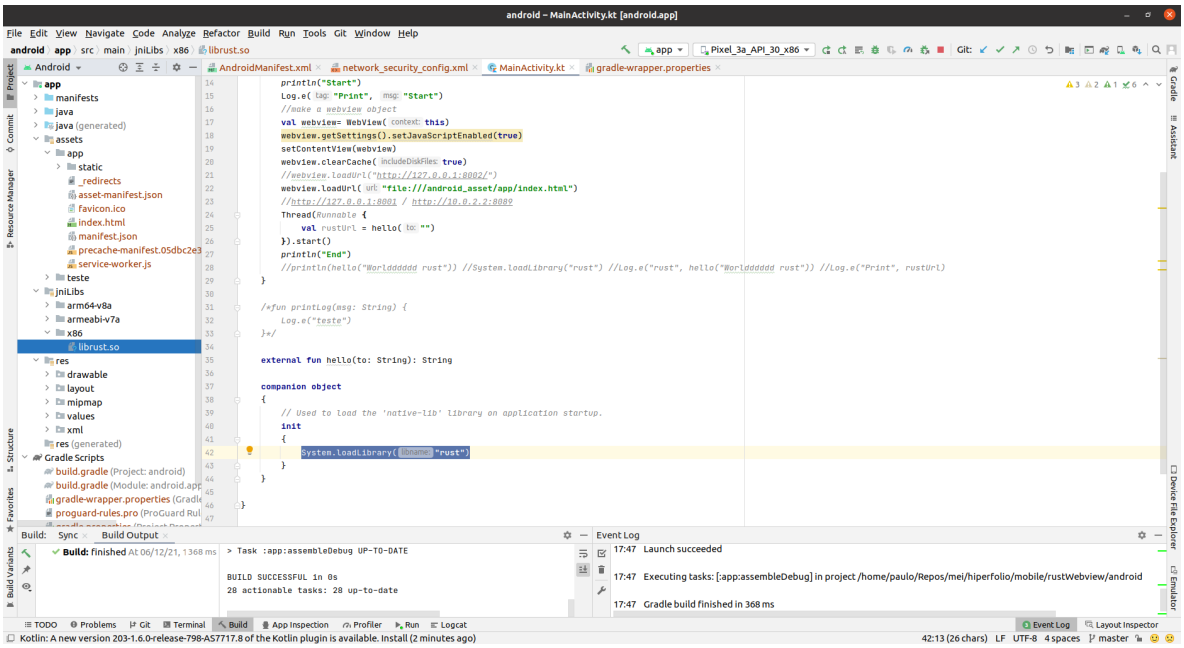

Figure 53: Android Studio with mobile modules structure

initial testes were with vanilla Javascript). The Rust application (compiled to the Android system), when executed, starts a local light web server (Actix Web Rust framework) listening on the predefined port. The webview SPA and the local server communicate through a RESTFul HTTP protocol - the SPA acts as a GUI and the Rust local server has endpoints and full access to the android device through JNI and NDK, plus all the Rust ecosystem potential. This solves the problem that many Mobile Web Apps have, of not being able to access native device functionalities (photos, video, file system, sensors, etc.), and, at the same time, solves the Mobile Native Apps limitation and inflexibility of not allowing technologies outside the system provider, offering a full cross-platform solution (Rust complies to any modern system) with the bonus of having a low-level more performative and secure language (Rust) has a base layer offering even most performance than Java or Kotlin.

We opted to maintain a bare bones light initialization layer based on Kotlin (or optionally Java), because it was easier to implement and gave us more transparency, allowing us to easily test everything inside the Android Studio SDK using mobile emulators and to defined each usual configuration file presented in an Android app with ease - Figure 53 showcases the file strucutre and load modules in the Android Studio (notice the Rust "librust.so" compiled module and React in the assets folder, besides Kotlin loading and gluing all in threads). This was not mandatory, our first prototype (see GitHub) was 100% Rust and ran directly on Android with the option of defining configuration files, but we found this option more transparent and flexible, easier to integrate, without any overhead. It also allowed us to easily define threads inside the Kotlin startup context, separating the different instances (Rust also allowed that through the JNI bindings, but we opted to avoid abstractions when possible).

In summary, we opted for an unusual architecture - a hybrid, server-based application running on a localhost instance on a separated thread, providing and feeding local endpoints of HTTP RESTFul communication to a

reactive frontend displayed on a webview natively initialized by Kotlin (we used Android SDK emulation for testing on the desktop, and later ran it in a real mobile device in developer mode). This local server is low-level, small and highly efficient compiled into Rust (modern language that competes with "C/C++") running "near metal" speeds (Actix Web is one of the most performative web frameworks according to benchmarks). This Rust server can communicate directly with the native device Android environment and, at the same time, with webview GUI running on the device and optionally with the outside world - it can run in offline or online mode, with a persistent state and local cached data (the opposite of Mobile Web Apps, that depend on network connection to fully work). This method allows us to have the advantages of Web Apps, even if in an offline context (we don't fetch the SPA from the Cloud, we have it locally stored, working offline through the Rust local server serving static content and caching locally new fetched data, for example in SQLite, for imediate access even without network), with full native access (the Rust server can communicate with the Kotlin/Java native layer thought the bindings) and superior performance (Rust is one of the most performative modern languages, much superior to java or C# middle tier alternatives, but maintaining or providing more than their flexibility and security).

This methodology, although unusual, provides many advantages: it allows full native access, something that a Progressive Web App (PWA) does not; allows to provide the application in a fully functional offline environment, as opposed to a traditional webview option, because the *frontend* communicates with totally local endpoints, which serve as intermediaries with the external world, fetching data as necessary or providing cached results if the device has not a connection to the internet, or allowing to store data locally; a native option, although it has advantages, it also has many limitations, because it limits the development environment to technologies sponsored by *Android*, blocking options with greater freedom of development or better solutions for certain problems, or, at the same time, limiting the flexibility and performance because Java is far from being the most performative option; the web ecosystem is one of the most cross-platform and complete GUI option, with the bonus of now providing WebAssembly performances.

Our design choice don't wants to claim to be a greater solution. It is an uncommon approach, with some limitations in regard to other options (over-engineering just to be able to use other tech stack, but if that tech libraries and ecosystem justify the need in a specific scenario it could be useful) and needs to be tested in a real-world scenario, but it has a novelty factor that we value. We didn't opted for this approach because it was superior, but because it was a new challenge, with a different approach to a common problem, able to provide new insights and data. We underline that this thesis would been a lot easier if we followed the route of the common ground, developing a monolithic Web solution and a mobile traditional well-documented approach like React Native, but instead we opted for the hard route, investing time researching novelty approaches and brainstorming new ones - Rust documentation in the mobile context and fully working examples are very sparse, we hope that this project can contribute to that by providing a full functional prototype and a different design approach to a common problem.

The end results was positive: we were able to build an app with a new programming language with a performance much greater than Java (Rust compiles to near metal performances equal to *C*), a much greater security and memory strategy than popular choices. We had an interface using pure web technologies (React), without the need to convert them to options like React Native for instance, but at the same time without the limitations

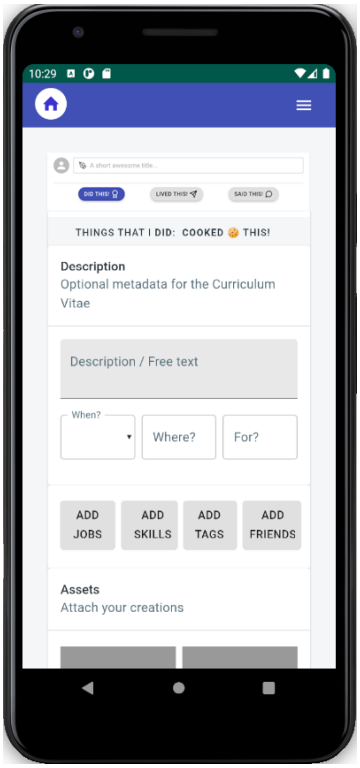

Figure 54: The application running on mobile (form creation page)

of a webview, witch full access to the native device functionality through the Rust local server and its native API access, or capacity to communicate with Kotlin/Java. Figure 54 showcases a screenshot of the final result app running on an Android emulator (we also run it in a real mobile device during tests). At the same time we had offline cache and functionalities (through the Rust layer local server) but also high quality web access through a reactive interface. The final result had very good performance and stability, while having the ease of using directly a web interface (without the need to convert or adapt), without the common limitations of a PWS app, while at the same time using a superior programming language with much better performance and paradigms (Rust), for that reason we enjoyed the experiment. In the future we would like to repeat a similar approach but using a fully compiled WASM interface, using one of the Rust WASM based reactive interfaces for the web.

5.7 SEMANTIC WEB & KNOWLEDGE GRAPHS

In the following sections, we'll summarize the implemented tools and components related with Semantic Web standards (for context, see the related State of the Art section at 2.3.1), mainly: a tool called EntiGraph, for Knowledge Graphs generation, also developed and used in the context of two additional projects (Major Minors and NetLang), and the European Skills, Competences, qualifications and Occupations Ontology (ESCO) ontology, implemented in the current project, developed and publish by the European Union.

5.7.1 *EntiGraph - Ontology generator tool*

One of the key factors of an e-Portfolio is the possibility to share and get feedback, to better get evaluations and assessments of our progression, including to get an overview of our own evolution and to make auto-evaluations, amongst other things. Besides this common factors, a Social Network paradigm introduced in the current e-Portfolio implementation, also transformed it into a **job market** (companies in search for employees and employees in search for a job), a **community aggregator** (people with common interests in search for sharing productions and stories with each other) and into a **creative gallery** (a showcase of productions, from arts, to academia, to education, to engineering etc.). In order to get this overview and interchange of information, better search and indexing mechanisms were needed.

The tradition keyword match search were enough in some scenarios (e.g. search for a community related with arts, by using this keyword), but for more advanced scenarios we found important to develop better strategies based on semantic web strategies (for example to better help companies finding users with the right set of skills by allowing a richer search query; for users to find posts related to some particular thematics; for users to better get a perspective about their self portfolio and a statistical overview etc.). For that, we developed a tool target at identifying and indexing various named entities on texts, building ontogolies around a common corpora, and exposing them in searchable Knowledge Graphs through SPARQL queries and visual graphs.

This tool was initially developed in the context of the *Major Minors* project, also developed by us in 2020/2021 during the current thesis, in which some parts were complementary - this tool is an example, initially designed for *Major Minors*, it was made modular so that it could be easily complied and expanded into an independent agnostic tool for implementation (and improvement) in the context of the current project, since both shared a common technological stack. Initially it was a static tool, only target for one specific corpus and structure, without CLI or GUI interfaces and many limitations (it ran only as a library in the context of a very specific textual structure). At the moment we expanded and improved this tool during the current project, building an API around it and a local REST server, with entry and exit endpoints, CLI and GUI interfaces for penalization of the parameters, able to get any kind of source text and configuration of the input (through JSON) and output structures (through an ontology skeleton, and building automatically a Knowledge Graph (saving a new output file or updating an existing one with new entries) centered around named entities recognition. Figures 56 and 57 showcase the GUI version of the application (launch screen and end screen), and the source-code file structure (made using Python, with a Rust layer for exposing it). Besides this central functionality, it also provides exploration interfaces and local server endpoints to expose the data and communication (this layer was developed mainly using Rust and triplestore libraries). Currently we are also integrating this tool in the NetLang I&D FCT project, in which we are part of the team, collaborating actively with it - recently we presented[Mar21a] an improved version of this tool and a contextualization about NLP and Semantic Web at the event Autumn NetLang Workshop, with the title "From data to knowledge - digital literacy in the service of corpora". In the meanwhile, we published two articles documenting its development and use cases[MCR21, Mar21b].

For developing this tool we applied techniques based on entity identification supported by Knowledge Bases, and built a tool able to transform static Big Data into interactive visual graphs, aimed at improving search mech-

anisms by allowing an entity based search instead of a regular keyword match. This tool is open-source and user-friendly, targeted for data exploration and ETL (Extract, Transform, Load) in corpora projects. It was inspired by the Major Minors[Mar21b, MCR21], a project also developed by us at the same time as the current one, sharing the base framework and tools, which the current one improves upon. Major Minors won the 1st annual prize of the Arquivo.pt competition[10], with the earlier versions of this tool. Since then we published and present two articles[Mar21b, MCR21] at SLATE'21 and Linked Archives 2021, about this same tool. Currently, during this project, this tool was greatly improved upon, now released as version 1.0 ready for the general public usage. Originally, it used only datasets generated by us (19 datasets of different entities collect with crawlers and Regular Expressions[11]), with static generation and without an interface (it worked only as a static library and customization was limited and script based); right now, the new version has CLI and GUI interfaces, local server endpoints and APIs, and webview based interfaces with many options for customization and visual tools for exploration and interaction with the generated graphs, but most importantly, it is starting to incorporate better source data, beyond the initial 19 static hand made datasets, transforming them into expandable ontologies, and at the moment we are incorporating also other ontological sources, namely the YAGO[PTWS20] ontology (developed by the Max Planck Institute for Computer Science), greatly improving the number of entities to search for, from thousands to millions, and greatly improving the semantic data with new ramifications from the YAGO version 4 Knowledge Base. The resulting output of this open-source tool is an ontology compilation with identified entities and relationships in text entries, that can be used as a Knowledge Graph to improve data exploration or search mechanisms in any kind of project with a text based corpus. Figure 55 showcases an overview of this application workflow.

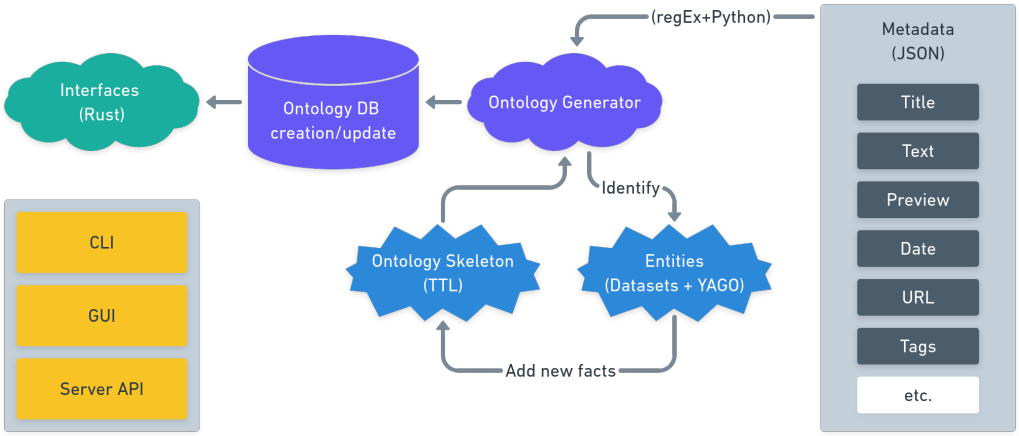

Figure 55: Overview of the EntiGraph workflow

10 Press clippings of the 1st prize: http://minors.ilch.uminho.pt/press
11 Datasets: https://github.com/Paulo-Jorge-PM/datasets-majorminors

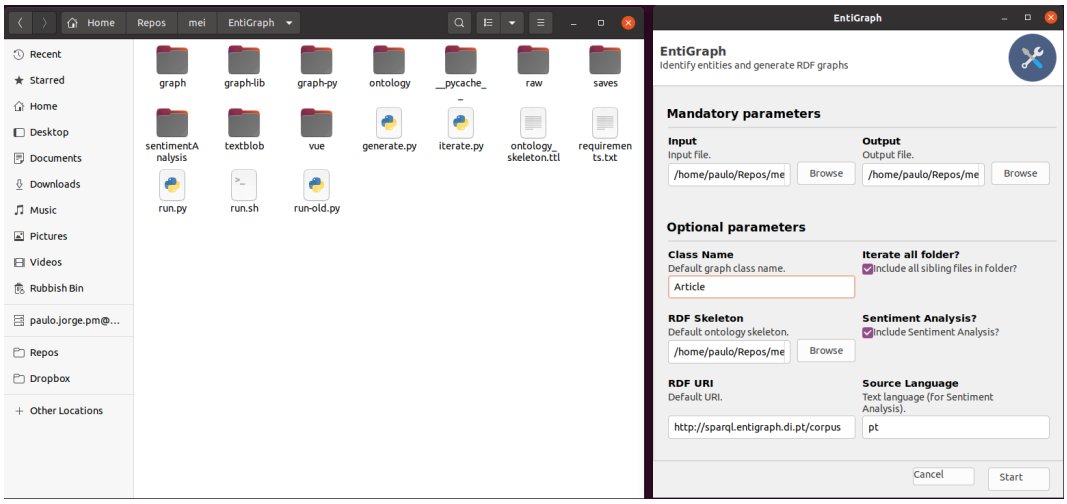

Figure 56: EntiGraph launch page (GUI mode) and source directory

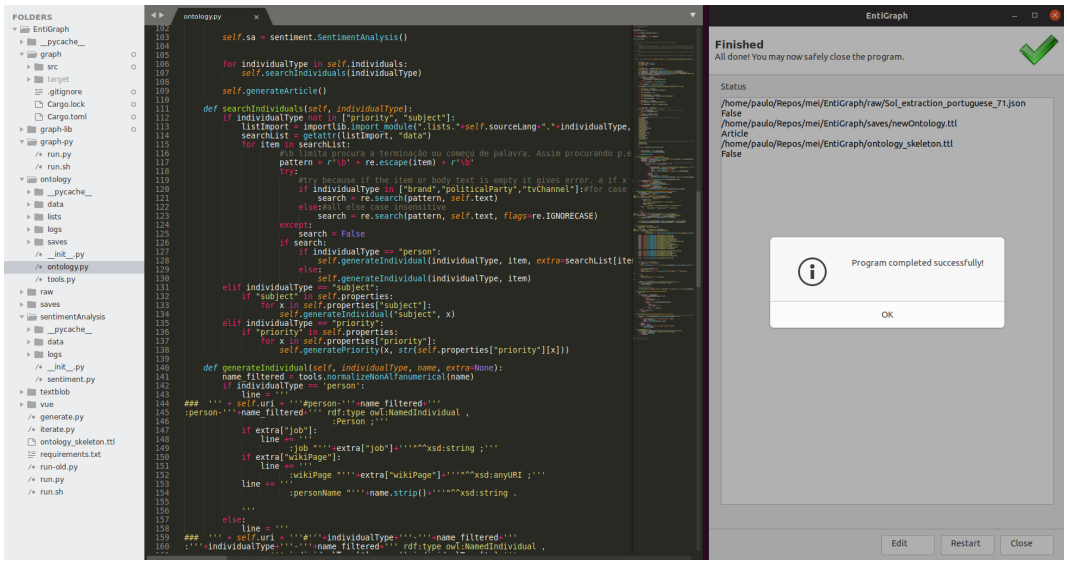

Figure 57: EntiGraph end page (GUI mode) and source-code snippet

A Knowledge Graph is a cluster of graph-structured data models (e.g. ontologies) with common ramifications and ways to efficiently interact with them. They "provide structured data and factual knowledge that drive many products and make them more intelligent and magical"[NGJ+19].

At the other hand, an ontology is a "specification of a conceptualization", "a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents"[Gru09, GOS09].

In 2012 Google launched the Google Knowledge Graph project, aimed at improving their search engine feedback, which popularized the concept of Knowledge Graphs[Sin12]. Since then, many projects have been trying

to translate Big Data into comprehensive data through similar methodologies, for example the ones related with the Linked Open Data initiative, WordNet, YAGO[PTWS20], DBPedia, etc. These approaches are expanding the concept of what we understand to be the World Wide Web, building the basis of the Web 3.0, also known as the Semantic Web.

Some of these projects, namely YAGO[PTWS20] and Major Minors[Mar21b, MCR21], focus on generating Knowledge Bases centered around named entities recognition (identification of real-world objects). They apply different techniques: the former extracts entities from structured sources like WordNet and DBpedia, while the latter builds its own structured data from thematic sources. Both target specialized and structured sources, while traditional projects work directly with raw text using NLP named entity recognition techniques (e.g. Regular Expressions), however these are too broad and inefficient in the context of an open corpus when precision matters. When one aims for maximum precision, a solution is to have an intermediary step with pre-generated and revised Knowledge Bases for supporting the named entities recognition, because it allows for improved precision in exchange for performance.

The current developed tool aims for maximum precision and applies these methodologies by extracting structured data, transforming it into Knowledge Graphs and storing them in triplestore databases. These data is exposed through visual graphs and SPARQL APIs, allowing for complex manipulation and extraction.

In the context of the current product, it feds the search mechanisms and data exploration interfaces. Each post made by an user, is stored in a common datalake, for next to be run and processed by the current tool (we called this entity based ontoloy generator tool "EntiGraph"), identifying and extracting target entities references, and building a graph of relationships around them, allowing the user posts data to be more easily explored, treated and exposed on the interfaces of the product.

One of the main goals of the Semantic Web initiative (Web 3.0) is to achieve a general Natural-language Interface, but it suffers from inconsistency, vastness, and vagueness[MS03]. The Linked Open Data project attempts to make this ideal come true, but it is still a work in progress. An endeavor to contribute to this ideal was conducted with the current tool, following the W3C recommendations and most recent standards intuitive interfaces were built in order to achieve an accessible product, based on Semantic Web standards. Mainly, ontologies were developed with inference mechanisms, using OWL (RDF, Turtle, etc.) to store the data in a graph database supporting SPARQL to query it (we used StarDog and GraphDB to store and interface with this data). Reactive interfaces and APIs were built to interface with this graphs, including GUIs to interact with the tool and multiple endpoints to expose its data.

Figure 58 showcases one of the resulting reactive interfaces (a search interface, to find posts that mention specific entities) developed for supporting data exploration from the EntiGraph generated ontology. The developed interfaces to interact with the resulted ontologies can be divided in three layers: a basic UI gallery through the main website; a reactive interface that generates SPARQL queries automatically in three layers of filtration (general classes, specific individuals and basic relationships); and a complete SPARQL API and/or visual navigation graph (through the triplestore databases default interfaces).

This project was built upon W3C official recommendations: RDF, OWL, and SPARQL. RDF represents information using semantic triples: subject, predicate, and object. We opted for the Turtle syntax for this representation.

Figure 58: VUE interface for data exploration of the generated Knowledge Graph

For storing and managing our triplestores we opted for Oxigraph: a graph database library created with Rust, implementing the SPARQL and RDF standards. Oxigraph includes a standalone HTTP server with an API in port 7878, and three different store implementations (one in memory and two based on file system: we opted for the SledStore mode, because it was native). Oxigraph has also Python bindings available, but we opted to develop its integration in native Rust, with builds for Linux. When executed, a Rust server launches a webview, with a GUI that allows any user to easily interact with this tool, by choosing an outpup and an input file (it supports plaintext, JSON and CSV by default, but allows to easily incorporate new formats). The tool then starts processing the text, interfacing with a Python module were we scripted processes and Regular Expressions to extraction of entities and generation of RDF triples for the ontology output. This ontology when finalized, can be saved or run in a local server that uses Oxigraph to expose an API for SPARQL queries, and launches a webview with a VUE reactive interface that allows to explore he data, or execute SPARQL queries directly.

5.7.2 *ESCO Ontology*

As mentioned before, we used the ESCO ontology, open-source database created by the European Union, to document and standardize occupations, skills and qualifications. Figure 46 in as past section showcase an interface of StarDog with an ESCO graph view, and the State of the Art (2) and the architecture section in the current chapter (5) more extensively described and presented this integration.

We will conclude by stating that the ESCO ontology was incredibly useful in the context of this project: it would be a challenge to standardize and exhibit so much complex data in an intuitive way for the end-user: this tool allowed us to do that standardization very easily, while allowing for an easy and fast querying of data through triplestore databases. Using SPARQL queries with very fast response times, we could develop a text suggestion system to help the user select with ease which skills/communities his posts were part off, and to help

users, companies, communities and education entities more easily find each other through search interfaces and structured data. Even the export system for the portfolio benefited from this ontology, because it helped us to better present filtering options and structures for the documents, because its ontological relationships and cross data gave us insights and semantic aid. This ontology dataset usefulness was invaluable. Figures 59 and 60 showcase screenshots of fields from the main form, dynamically showing test suggestions for skills and jobs to add to the artifact entry, gathered from the ESCO ontology running on StarDog through SPARQL queries.

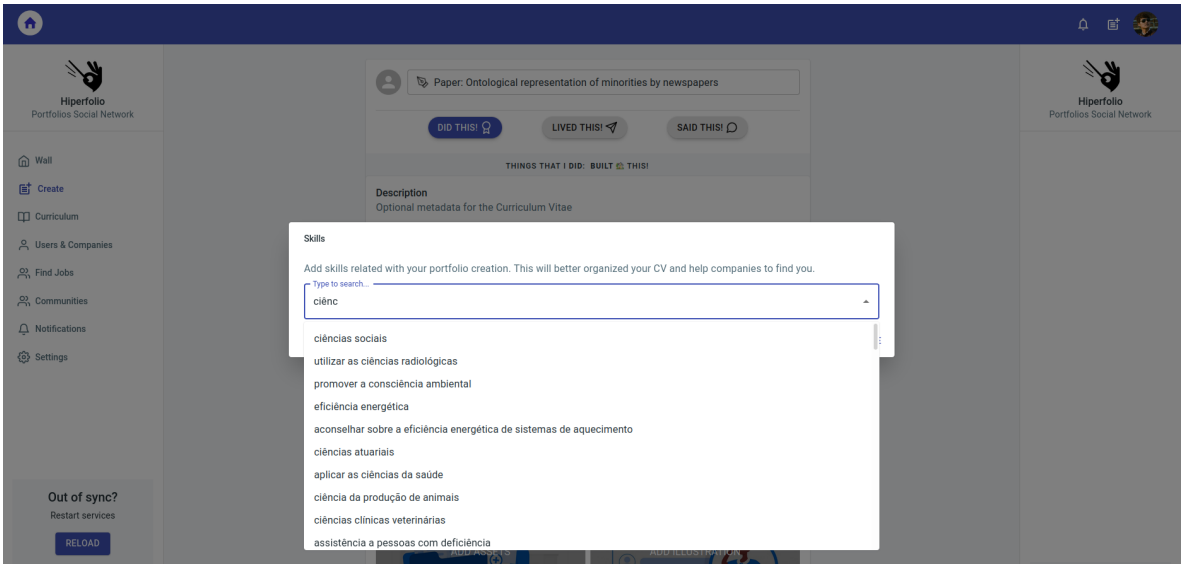

Figure 59: Interface: skills suggestions from ESCO

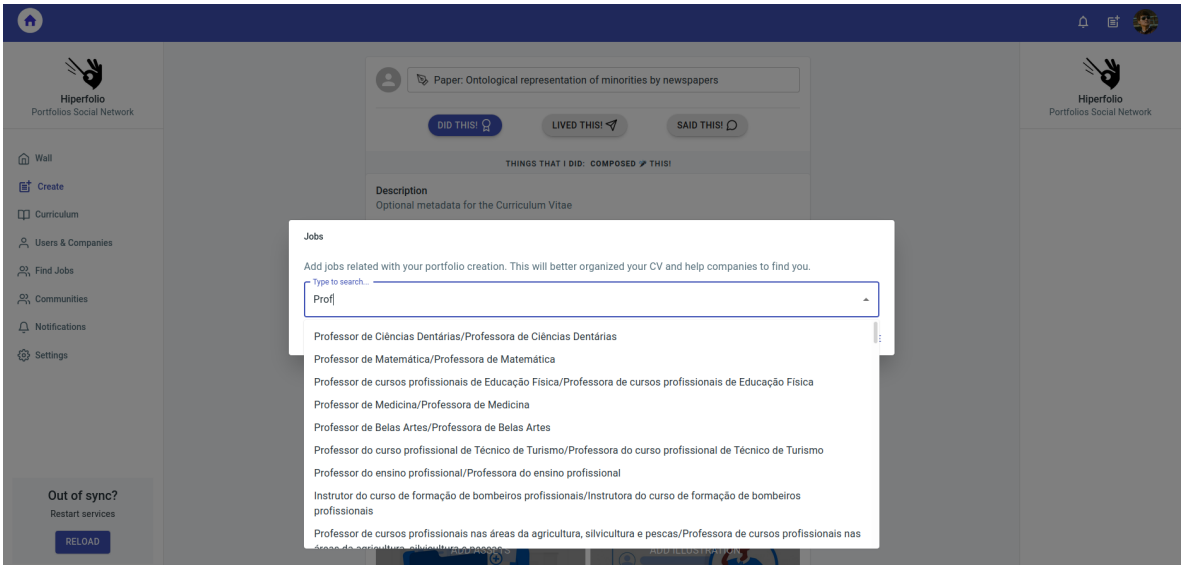

Figure 60: Interface: job suggestions from ESCO

5.8 SUMMARY

This chapter summarises the core implementation of the project, it is the central stage of its development. In it, we summarized the major aspects concerning how and why we implemented each aspect of the final product. We started by summarizing our goals, then presented the initially gathered data from the first survey regarding market prospecting, analyzing it and defining our new goals according to these results (we defined 8 major goals). We then proceeded to present, sequentially and in depth, each step and layer regarding the development of the product prototype, starting by presenting the initial idealized mockups, then each major layer of the architecture (Static Content Layer, API Gateway Layer, Service Mesh Layer, Kubernetes/Orchestration Layer and Message Broker Layer), next we focused the developed databases and schemas, followed by the frontend web application, mobile app, and the Semantic Web approaches (e.g. the adopted European Skills, Competences, qualifications and Occupations Ontology ontology from the European Union) and related developed tools (e.g. the EntiGraph open-source released tool for the creation of Knowledge Graphs).

We focused all the main aspects of the development of the product, in a real context of heavy traffic in a Social Network like system, targeting for high-availability and scalability. For that reason, each layer of the architecture were a key element, aiming at providing that level of high-availability. We also gave preference to the implementation of exploratory techniques that were described (e.g. the mobile app that uses an hybrid unusual approach based on Rust and Kotlin; the specialized architecture design; the semantic web approaches; etc.), because we considered that they would provide the most meaningfully data in a research context due to its novelty.

We also underline the EntiGraph tool and related developed interfaces, that, besides the main website, mobile app prototype and architectural infrastructural design, was also one of the main outputs of this project, created in collaboration with other R&D projects. Each section includes a deeper analysis and overview diagrams that further documents them.

Part III

OUTPUTS - CONCLUDING REMARKS

6

DATA ANALYSIS

In this section we will summarize the main outputs of the project and a brief analysis.

6.1 OUTPUTS

We worked with a Design Science Research (DSR) methodology, in which we explored a Proof of Concept (POC) in incremental cycles of development and evaluation, in which at each cycle step we would evaluate our research and increment the project with new data and targets in new exploratory steps of development. The final result were exploratory products and new documented methodologies achieved from this cycle of research. It was a very productive and multidisciplinary year, in which we integrated some of the tools and design cycles of the current project with 2 additional R&D projects necessities, complementing each other.

Regarding direct outputs of the project, we would highlight the following tools and methodologies:

- **Web app:** we built a React (mixed with some VUE) prototype frontend interface for the end user to interact with the final product.
- **Mobile app:** we built an hybrid mobile app (target for android), with a new exploratory approach to test Rust in the mobile segment (a local server in dialogue with Kotlin and Webviews, that would cache the data for offline use, but at the same time online mode).
- **Cloud infrastructure:** we designed an architecture based on microsservices, target for scalability and high availability, based on K8s (using Rancher as management), with complementary exploratory layers like API Gateway, Message Broker, Static Layer and Service Mesh.
- **Exploratory databases:** we explored new concepts related with databases scalability and high availability, in particular decentralized cluster based databases, like CockroachDB, that would facilitate the distribution in a K8s environment.
- **Semantic Web:** we also developed and integrated exploratory Graph Databases based on triplestores, target for semantic data exploration and scalability based on RDF/OWL and Semantic Web concepts. Besides the ones created by us (with the EntiGraph) we incorporated existing ones like the ESCO ontology from the European Union.

- **EntiGraph tool:** developed an open-source tool also used in the context of 2 additional R&D projects, targeted at text mining, named-entities identification and Semantic Web, able to build Knowledge Graphs and update/expose ontologies from textual sources. This tool has a Command Line Interface and API access level, but also a GUI interface based on Gooey for the general public.
- **Interfaces to explore EntiGraph generated data:** developed VUE interfaces and Rust based tools to explore and interact with the generated Knowledge Graphs from EntiGraph.
- **e-Portfolio methodologies:** we developed new methodologies to better transcribe e-Portfolio concepts to interfaces and databases. The general theory about e-Portfolio and workflows are overcomplicated, we tried to reduce them to the most intuitive format possible, creating new workflows and concepts around them ("did this", "lived this", "said this").
- **Social Network with emergent web technologies:** we did exploratory work on rebuilding the Social Network concept with emergent architectures and technological stacks, namely Microsservice various new concepts, Rust and WebAssembly.

Besides this direct outputs of the project, during this year (2021) we also participated and contributed to Academia outputs, some of them resulted directly from this project, others were developed at the same time and partially integrated (e.g.: some tools from Major Minors were further expanded/integrated on the current project and are being tested and will be integrated in the next year project to be released NetLang). The following list summarizes these academic outputs:

Prizes:

- 1st Prize on the annual Arquivo.pt competition (June 2021), with the project *Major Minors* (which some tools were also developed during the current thesis, namely EntiGraph, released now with version 1.0). Homepage at: http://minors.ilch.uminho.pt/press

Published Papers:

- Paulo Martins, Leandro Costa, & José Ramalho (2021). Major Minors - Ontological representation of minorities by newspapers. In OASIcs, Volume 94, SLATE 2021. URL: https://www.dagstuhl.de/dagpub/978-3-95977-202-0. DOI: https://doi.org/10.4230/OASIcs.SLATE.2021.3.
- Paulo Martins, Leandro Costa, & José Ramalho (2021). Implementation of a Knowledge Graph of press clippings representing social minorities. In Linked Archives 2021 - International Workshop on Archives and Linked Data. INESC TEC, Portugal. Url: http://ceur-ws.org/Vol-3019/. [in press]

Conferences:

- Paulo Martins (2021). Autumn Netlang Workshop. "From Data to Knowledge – Digital Literacy at the Service of Corpora" (Presentation of the EntiGraph tool). University of Minho, Portugal. 11/11/2021. URL: http://cehum.ilch.uminho.pt/events/500
- Paulo Martins, Leandro Costa, & José Ramalho (2021). Conversas@Rossio. "Save the data: Como os arquivos web combatem a desinformação". UNL, Portugal. 28/09/2021. URL: https://rossio.fcsh.unl.pt/2021/09/08/ciclo-de-conversasrossio-com/
- Paulo Martins, Leandro Costa, & José Ramalho (2021). "Major Minors com o Arquivo.pt no Dia Mundial da Preservação Digital". Arquivo.pt, Portugal. 04/11/2021. URL: https://sobre.arquivo.pt/pt/major-minors-com-o-arquivo-pt-no-dia-mundial-da-preservacao-digital-2021/
- Paulo Martins, Leandro Costa, & José Ramalho (2021). Linked Archives 2021 - International Workshop on Archives and Linked Data. INESC TEC, Portugal. 13/09/2021. Url: https://linkedarchives.inesctec.pt/.
- Paulo Martins, Leandro Costa, & José Ramalho (2021). SLATE'21 - 10th Symposium on Languages, Applications and Technologies. School of Technology, Polytechnic Institute of Cávado and Ave, Portugal. 01/07/2021. Url: http://slate-conf.org/2021/home.

Collaborations

- NetLang project: the developed tool EntiGraph, developed for the current project, was also ported to the NetLang FCT R&D project, which we currently are a team member (including crawlers create on the context of Major Minors which were also remade and released with new functionalities and interfaces).
- Major Minors: as the Netlang project, the EntiGraph tool was shared between both projects and greatly improved and released as open-source version 1.0 during the current one, with greatly expanded functionalities and GUI interfaces and APIs. Some of the VUE interfaces for data exploration were also ported and improved upon.

6.2 SURVEY DATA

Besides these outputs (tools, methodologies and academia), we also gathered data before and after the development cycles through surveys, resulting in 2 datasets of surveys. The first survey data, gathered before the project started, as a market prospection, was analysed and explored on previous chapters at C, so we will not repeat it here, just mention that the final results tried to solve the most common user expectations gathered from it (the second survey data indicate that we were able to work on that direction). The second survey, based on Technological Acceptance Model (TAM), was gathered after the main tool prototypes were developed, to determine the quality, usability and general public acceptance of the tools, in accordance to scientific models. In the Appendix section (iv) one can find the full survey model. The gathered data (17 submissions gathered from

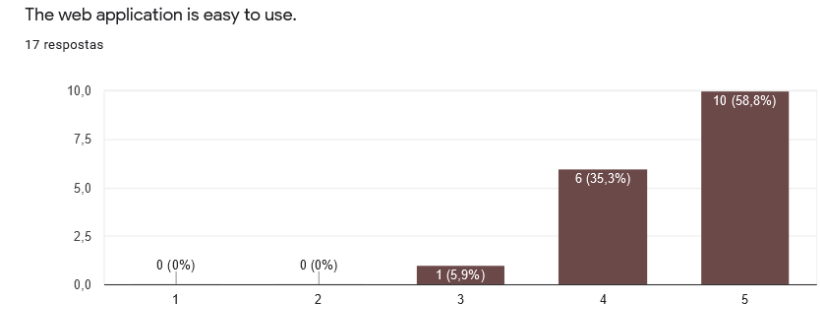

Figure 61: TMA: Perceived ease of use 1

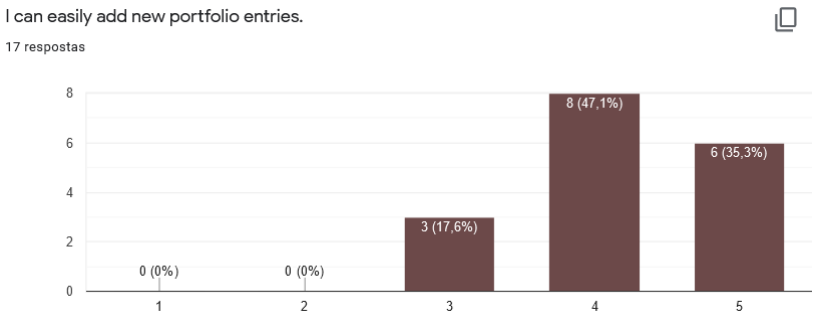

Figure 62: TMA: Perceived ease of use 2

friends, students and family), and the results were positive for the first tests. Figures 61, 62, 63 are an example of the obtained results for the "Perceived ease of use" aspect.

The majority of the results were positive, with an average result, in a scale from 1 to 5, of mainly 5 and 4, with some 3. The majority of the users found the application familiar, intuitive, easy to use, and not too much time consuming to interact with. We'll highlight the results with less balance, that we should work more on, namely Figure 64 ("I can easily edit my past portfolio entries"), in which we obtained 11.8% results with score 2, 29.4% with score 3, 41.2% 4 and 17.6% 5, indicating that almost half the users didn't found easily how they could edit their posts - we plan on work on that interface aspect. When asked if they would use the application again or even suggest it to a friend, the majority answered positively, but some users gave negative scores - we would like to improve this statistic in the final release, this results indicate that we should work on engagement mechanisms and invest more on gamification strategies to create a strong user base.

In general we received positive feedback from users. The tools are in a prototype state, some aspects are not finished, we need to release final versions before public deployment and marketing, but the general acceptance in an initially phase seams positive: we aimed to release an intuitive product, that gave to the general public the power of e-Portfolios in a familiar and not-time consuming way, joining their power with those of Social Networks. The initial feedback indicates that we are on the right track for the general public acceptance.

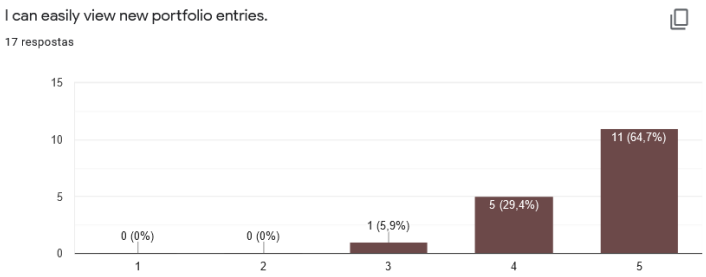

Figure 63: TMA: Perceived ease of use 3

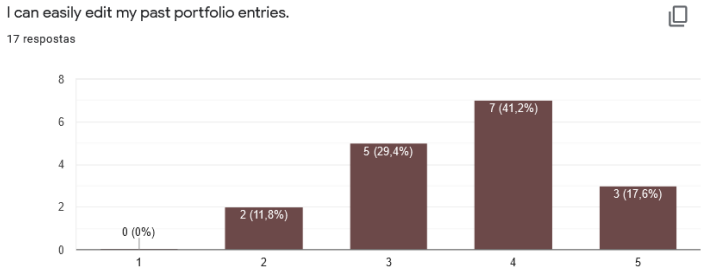

Figure 64: TMA: "I can easily edit my past portfolio entries"

In the Appendix section (iv) one can check a summary of all these outputs, with hyperlinks to the GitHub pages of the developed tools/infrastructures, summary and hyperlinks to the academic outputs, and models/templates used for the two surveys.

In the following section we will briefly summarize an extra output: a new vision regarding microsservices and WASM, an idealization of an Open Linked Web of Processing Agents in contrast with the idea of an Open Linked Web of Data. We planed on implementing it, but we had not time to develop and explore this theory, but we envision it as part of the future of the web and had an initial sketch of this theory in our planned exploratory iteration steps in the development phase, for that reason, we include this sketch here as an output.

6.3 WASM AS A MICROSSERVICE

WebAssembly (WASM) can be conceived as a powerful tool for expanding the concept of *Microservices* and *Micro Frontends*, because it itself is an encapsulation tool, like Docker it can isolate and distribute services in a stable encapsulated containerized way (because it can be compiled to a universal binary format from any major language, and run/understood by any browser or local virtual machine, working as an universal intermediary format). This chapter will address theoretical considerations in this regard and how they could enrich the future of web development and the current present project.

We had no time to implement or explore this theoretical proof-of-concept approach, but we will briefly document it here. We developed this theory during the product Design Science Research methodologies cycles of design and evaluation, where the cycle of exploration of new approaches to the design of the product and its evaluation would lead to new results and ideas. But due to the complexity of the final idea, we had no time to explore more cycles, this theory will be explored in future research projects and papers. We will briefly summarize it in this chapter as an hypothesis, for future researchers that would wish to expand upon it. We started a prototype tool to support it, called Baton (available in the Git repository of the project under abandoned ideas), but never finished it - it is a Rust local server with an interface built using React, with the objective of managing and load balancing WASM services distributed in a cluster (the interface works but we never finished implementing the management system, giving priority to the current product).

The microsservices paradigm is revolutionizing some aspects of software development, introducing modularity and decentralization of responsibilities in the context of products, dividing it in its atomic responsibilities in the form of services. Nevertheless, this method is mostly being applied in the isolated context of individual applications. Orchestration tools like Kubernetes allow for immense scalability powers, distributing the services along clusters of servers, but these immense resources are being used only in the context of each product (a pod in a Kubernetes cluster is a Docker container with a specific set of thecnologies and Operation System parts), and many companies reinvent the wheel with each new microsservice (new builds of a containerized Docker with a full partial Operation System, etc.). One of the major unsung laws of software development is "don't reinvent the wheel" and don't over-engineer.

Microsservices could be, metaphorically, envisioned as single Functions (or procedures or methods), which are a set of instructions bundled together, in most programming languages, to achieve a specific outcome. Programming, in generic terms, could be divided in two main layers: Data and Data Processing - every single piece of software, in its atomic Functions, take some kind of data and processes it, giving a new form of data as result or calling a new process. Functions can be metaphorical viewed as a funnel with a filter, where one poured water (data), it got filtered (processed) and at the receiving end one gets a new kind of liquid (output data, or a set of new function calls). Serveless Functions, very popular at the moment as an alternative microsservices pattern, are an extension of this metaphor, but he suffers from many challenges (mainly cold starts, in which performance gets an hit). A WASM based orchestration system, where each Serveless Function is a WASM encapsulated service could theoretically improve this concept.

At the moment, in the context of computation, the majority of applications are a set of thousands of these funnels bundled as an individual software, each one isolated from the outside world, each one independently developed, never sharing processing resource power or design recycling with other products. New algorithms and necessities are unique funnels (by funnels we mean functions, we are keaping the metaphor), but many common operations are many times repeated and never reused between products (for instance, a very basic and simple example is a function that adds two variables: it is a very common and universal procedure that was typed thousands of thousands of times by development teams; the example is very basic, but imagine it in thousands of repeated simple algorithms throughout each programming language and team).

We envision a distributed programming context for the future, in which these "funnels" are public and/or private endpoints in a linked open web of procedures/methods (in contrast to the Linked Open Data concept, we aim for a Linked Open Processing). We envision a global API, in which any software, local or in the Cloud, using any kind of programming language, could call a simple endpoint using a simple communication protocol like REST, and run a remote distributed Function (procedure/method), from simple ones to sets of Functions that call each other and form more complex algorithms and applications, by inputting data as a request and obtaining a result as response (or a link and data for a new set of distributed Functions). For instance, imagine a global API, like a central repository of this paradigm that we will call "Linked Open Processing" (in contrast to the Linked Open Data project), where a user need to obtain numbers in the N position of the Pi mathematical constant - he could call, from his code base in any programming language, a simple REST endpoint (e.g. "htttps://lop.net/function/piPosition?var1=10"), calling a function called piPosition, inputting the variable "10", and obtaining as result of the endpoint call the number "3", which is the 10th number of the Pi constant. This simple example is not very interesting, but imagine this possibility magnified to thousands of different needs and function calls, in a open-source repository with millions of created "funnels" endpoints, and million of community developed and perfected algorithms shared in open-source format.

This kind of global repository could decentralize processing power from local computers and better recycle and share code-bases. For instance, one could have a smartwatch, with very low processing power, running effortlessly very intensive algorithms in a quantum computer. Super computers would be perfect candidates for this kind of paradigm, allowing any kind of software to incorporate algorithms that its base hardware would not be able to run. This kind of approach already exist in some degree (e.g. AWS, Chromebooks, etc.), but it is not open-source, neither recycles and centralizes common algorithms and function needs, they are focused on particular objectives. Imagine we needed to create, once again, a certain complex sorting algorithm in an exotic programming language or limited hardware: you could have, effortlessly, access to the best community tested implementation of this algorithm, with potent processing power, by just calling an endpoint and saving its resulting data.

Of course, this approach have many logistic problems. The scalability and processing needs to satisfy an hungry digital world would be immense. But distributed decentralized approaches like the microsservices architecture and Kubernetes already are working on solving them: the Project Google Stadia, for instance, already fights this challengs with some success, it streams in real time heavy intensive videogames, running them on their servers, emiting the result and input interaction with any gadget without much processing power - it proves

that logistically it would be possible. And, of course, this Linked Open Processing repository could have business models, similar to Cloud management, where one has monthly maximum free quotas of processing power, and needed to pay more to go beyond that (limiting abusive traffic). One could also allow the community to create public (free) or private (paid) Functions/Algorithms ("funnels"), in which the private ones could be charged for its utilization (charge the creator, or give a percentage of its use to the creator), or only used by some companies as their private repository.

But now enters the concept of WebAssembly as Microsservices: WASM is the perfect candidate for this kind of paradigm. It is the most performative language for the Web, but not only that, it can beat many low level programming languages, because it compiles to a specific a assembly format (as the name implies, it is Assembly for the Web).

It is a decentralized format, to which many different programming languages already can compile to (Rust, C, C++, C#, etc.). It was made with the philosophy of being an universal, intermediary format. It can be viewed metaphorically as what Java Bytecode is to Java, or what CLI is to C# (an intermediary format), but it is that to any kind of programming language with the bonus of running in the web - in the future it could be the universal intermediary format. The Linked Open Processing repository could be based on WASM microsservices, "funnels" that only have one single responsibility, accepting data, processing it, and returning new data or calling a new process. Each user contributing to this global repository, would submit a compiled and running WASM, and a respective source-code in the original language, so that the user community could check its validity. For avoiding security risks, this repository would have a check and vote system, in which a new "funnel" microsservice would only pass if it has not any kind of malicious code (this kind of repository would face many kind of security challenges for its end-users, from data thief to malicious or troll code, it is never a good idea to execute unknown code, for that reason a check and vote mechanism would be important to exist before allowing an endpoint to go public).

This kind of approach could also give non developed countries access to more processing power, where performance hardware is very expensive - with a low level low cost hardware, people in undeveloped countries could have access to top level processing power from any kind of hardware, only needing internet access and software adapted to this paradigm, in which expensive and/or common functions/algorithms would be called and executed remotely.

This approach is an incentive for companies to share and reuse code bases. It re-imagines the Open Linked Data way of chaining and obtaining data (Web of Data), into a Web of Processing Agents. The objective is to free the end-user of the costs and size of performative hardware - for instance, smartphones using this paradigm could be lighter and smaller, not needing as many expensive components, or even smartwatches or glasses could have access to very expensive processing power. This approach would also be more ecologically, because hardware waste is immense, users would not feel the urge to change hardware as often neither would it drain as much raw materials from the planet to be performative - a network of cloud based Processing Agents would mediate that need (users would not feel the need anymore to every year change their smartphone to next most expensive one, creating enormous ecological wast and bad raw resources management - a simple gadget with internet would have access to all the processing power in the world).

In the later cycles of development of the current project, we want to try this paradigm, converting all the microsservices to WASM services, but we needed to develop a platform for managing in a distributed environment these Processing Agents, but would be a new project by itself, we had no time to implement this idea currently. But we document it here for scientific reasons, to share it so others can explore it.

6.4 SUMMARY

In this concluding chapter, we made a summary and description of the main produced outputs, and a brief analysis of the public acceptance through a Technological Acceptance Model survey, target, amongst other things, at analyzing the perceived ease of use, which was generally positive. The interface aspect with a relatively lower score was related with the ability to edit past portfolio entries, indicating that we could improve it to be more intuitive.

We finalized by including a brief description of an output not implemented or explored, but present in the initial design cycles, relate to the use of WebAssembly as an universal network of microsservices, expanding the concept of the Linked Open Web of Data (Web 3.0) into a Linked Open Web of Processing (we developed an app called Baton, with an initial interface based on React, but never finished this concept, focusing our efforts on more basic aspects of the project, but decided to maintain this brief chapter sketch as a side output, as an exploratory concept).

We also included in this chapter our main scientific outputs (papers, conferences, prizes, etc.) produced during 2021, some from various R&D projects accomplished in parallel with the current one, and which both share some common tools.

7

CONCLUSION

In retrospective, we started with a simple premise: to bring the concept of e-Portfolios to the general public. The past implementations of this concept and resulting applications never had popular acceptance, it never got e-Portfolios to the mainstream audicences, mainly because the theory regarding the concept of e-Portfolios resulted in applications that make it seem more complex that what it really is (e.g. the use of overshadowed terminology like "artifacts", "assets", "evaluation", "reflection", "collections", "feedback", commonly used amongst the most popular authors like Helen Barrett (2010)[Bar10], as analysed in the earlier chapters of the current work), distance the general public from e-Portfolios (only in Education and some sectors line Medicine or general projects like some developed by the European Union it has had some relative acceptance, but a very limited one). This project main objective was to transmute the concept of e-Portfolio into its most basic and simple formats, and to transcribe it to the masses, in a familiar way, into an intuitive product (Proof of Concept) for creative communities or individuals simply wishing to share their creations for professional/educational progression and feedback.

The adopted strategy, in a first instance, was to join the concepts of Social Network with the e-Portfolio, because both could complement very well each other vectors (Social Networks gave e-Portfolios the feedback, reflection and community aspect that it needs, while making this interactivity familiar at the same time).

The second step was to transcribe the concept of e-Portfolio into a simplified data format, reduced to its most essential elements, because we found that one important aspect that was failing general public acceptance was inefficient attention to easy of use, and lack of not time consuming forms of input (statistically, users decide if they will use an application ever again in its first seconds of experience, we couldn't overwhelm users with complex forms and unfamiliar concepts from the get-go). For that reason, we created a new framework and workflow to represent e-Portfolios dynamics, mainly by reducing input to the bare minimum and dividing them into three different categories: "Did this", "Lived this" and "Said this" - besides sounding like a motto and having a positive memorization marketing effect, this strategy also allowed us to filter the types of post inserted into the application ("Did this" feeds the main portfolio entries, "Lived this" filtered traditional milestones achievements common to a Curriculum Vitae, and "Said this" filtered small talk and generic entries). Initial surveys suggest that this strategy was a success (6.1).

A third important strategy that we adopted was related with Semantic Web and new, emergent technologies: we integrated a European Union project called European Skills, Competences, qualifications and Occupations Ontology (ESCO), which is an ontology, in dozens of languages, with thousands of skills, jobs and education

diplomas in an ontology format (RDF Web 3.0 standard), with rich relationships and metadata. It allowed us to better categorize the data, and to more easily help users to tag and show their content. We also developed new tools around this concept, like the EntiGraph open-source tool, targeted at creating Knowledge Graphs from corpus of texts, creating automatically new ontologies from texts with named entity recognition and graph data exploration - this helped us to better expose search interfaces to the users, communities and entities seeking them. Rust and WASM emerging technologies aided us a lot in this process.

Another important strategy was to give the user its data back in a useful format: he/she could generate CVs, portfolios and download it in more traditional formats, or to expose himself to companies and communities interested in sharing his passions for their creations and common fields/skills. We should also mention the importance of the data categorization in the form inputs, most of them non mandatory, but enough to not tired the user or be time consuming, but at the same time to give us the needed bases to use the data in a meaningful way (e.g. we allowed the user to attach skills, jobs, friends, artifacts/assets to its creations, in an intuitive way, but we made these fields optional, later, if generating a CV the need arise, the user could fill the needed data with time to spare). The diagram 27 was important at documenting this workflow strategy.

The community aspect was also very important, to give the feedback and reflection vectors that make e-Portfolios so important, for that reason the integration of the Social Network model was essential, through it we tried to give users these kind of tools.

Last, but not least, the most important aspect of this work was the technical one: we aimed, through Design Science Research and Proof of Concept methodologies, to use the premise of this project to research and develop exploratory approaches to emergent web technologies, documenting them in the Implementation chapter (5). We aimed to design a Social Network future-oriented, in the context of very intense traffic, high-availability and scalability, researching and designing modern architectures target at solving these non-functional requirements (microsservices, Kubernets, Rancher, etc.) and by using various emergent technologies in order to better achieve it (WebAssembly, Rust, Service Mesh, Message Brokers, Semantic Web techs etc.). From this, resulted different exploratory approaches (an exploratory mobile app with an uncommon design proving that Rust could be useful in this context; triplestore databases based on Rust; WASM implementations and interfaces for data exploration; a new tool called EntiGraph for Semantic Web outputs; a modern architecture design aimed for scalability and high-availability, etc.). During the various design cycles of this project, we incremented it with new exploratory vectors. Our feedback of each explored tool implemented is very positive: we research many, but opted for the best ones in the context of the current project, and each one proved immensely valuable to our objectives: Rust promises to revolutionize web development with a new approach to programming (joining the security of a high-level programming language with a the performance of a low-level language) and a rich ever evolving cloud ecosystem; WASM promises to revolutionize the future of the web, bringing assembly like performances to the browser, while destroying the gap between desktop and web, and at the same time promising to become an universal glue between different programming languages; Semantic Web standards (ontologies, RDF, OWL, etc.) promises to take the internet to Web 3.0, by transforming it from a Linked Open Web of Documents to a Linked Open Web of Data; and new microsservices architectures and methodologies (K8s, Service

Meshes, Service Brokers, etc.) promise to augment distributed systems with more efficient management of their resources etc.

In the academic aspect, during 2021, we collaborated with 2 R&D projects (one of them resulting in a prize), we also produced some results, namely two publications (with other two in production, for publication during 2022) and 5 conference presentations. The Appendix (iv) and the Data Analysis chapter (6.1) summarized these achievements.

The resulting outputs had a positive reception in the first analysis surveys based on TAM ((6.1). We conclude by stating that we were able to answer the majority of the initial identified questions (3) through the outputs produced and documented exploratory methodologies in previous chapters. Namely, throughout the previous chapters, we were able to answer positively to initial questions like:

- How to convert the e-Portfolios concept to an intuitive web/mobile product?
- What distinguishes Curriculum Vitae based products from the e-Portfolio?
- Which projects already exist on this area, did they succeed and why?
- How to reimagine Social Networks with an e-Portfolio baseline?
- How to reimagine architectures for high-availability and scalability?
- Which emergent technologies could redefine Web Engineering for the following decades?
- How to integrate Microsservices inspired architectures into a Social Network context and how could they benefit from it?
- How can Microsservices (in specific orchestration through Kubernets) contribute to the future of Web Development?
- How can Rust contribute to the future of Web Development?
- How can WebAssembly contribute to the future of Web Development?
- How can Semantic Web approaches contribute to the future of Web Development?
- What is the relevance of Micro Frontends to the future of Web Development?

We will only highlight that the solo question that we found a negative answer was the last one: initially, Micro Frontends were targeted on the first design cycles of the project, we explored them, both during the State of the Art and during the implementation of the EntiGraph interfaces, but by the end of the project we concluded that they could only be useful in very specific scenarios, they try to over-engineering something that by its nature is already very modular (reactive interfaces already provide that kind of functionality in a positive level), resulting in much more complex implementations without the real benefits of microsservices. It could be useful in scenarios where one would need to interconnect different and complex frameworks in the same frontend (for example,

mixing React with Vue in the same interface), but this scenario is so rare and not advised (it introduces heavy performance penalties and management problems), for that reason we found it a niche technology, that would need further improvement until ready and really useful (it could be useful for a better management of the teams, but poor for performance, because we already consider reactive interfaces to be too intense, we don't wish to overload users browser side with more resources consumption). We envision WebAssembly as a possible better solution in this context, by unifying all the frontend frameworks, allowing to create a really performative microsservices ecosystem in the frontend context, but that future is still very distant from mainstream acceptance because WebAssembly is still very recent.

For all the other questions, the previous chapters were able to answer them positively. It was a joy to implement each one of the exploratory technologies in analysis, the final product is a testimony to its competence, and they promise for a more performative, more scalable, and more secure web (Rust, WASM, K8s/Microsservices, etc.) and new ways to explore and interact with data (Semantic Web approaches), indicate a bright and joyful future for Web Engineer and an increasingly fine line between Distributed Systems and traditional desktop applications.

In conclusion, we would testify that the current project was very positive for our growth as researchers, and most importantly, to the community and social impact, being it through the resulting tools or resulting exploratory implementations, which European Union is also targeting (refer to previous chapters 2 regarding ESCO and EU work towards e-Portfolios). Our initial questions and aims ended up being positively answered in the conducted surveys (6.1), in which the majority of users scored positively the Output Quality, Perceptions of External Control, Perceived Usefulness, Perceived Ease of Use (etc.) of the initial product prototype. This means that the users understood the e-Portfolio concept and its implementation through the current project. The final step would be to finish the test phases, perfect the prototypes, and release the product publicly for general use - we hope these results to be useful to the Academy and the society.

BIBLIOGRAPHY

[3fa20] 3factor app, Aug 2020.

[Age16] Lise Agerbæk. Europortfolio. *Gymnasiepædagogik*, 2016(104), 2016.

[All18] Chris Allen. Haskell and Rust. *FP Complete*, 2018.

[and21] Rust in the Android platform, Oct 2021.

[Arn06] Michel Arnaud. Improving european employability with the e-portfolio. In *Handbook on quality and standardisation in e-learning*, pages 263–273. Springer, 2006.

[Asa20] Matt Asay. Why AWS loves Rust, and how we'd like to help, 2020.

[Bar05] Helen C Barrett. Researching electronic portfolios & learner engagement. 2005.

[Bar10] Helen C Barrett. Balancing the Two Faces of E-Portfolios. 2010.

[BB20] Joydeep Bhattacharjee and Joydeep Bhattacharjee. *Basics of Rust*, page 1–30. Apress, 2020.

[BBB+17] Abhiram Balasubramanian, Marek S. Baranowski, Anton Burtsev, Aurojit Panda, Zvonimir Rakamari, and Leonid Ryzhyk. System Programming in Rust: Beyond Safety. In *Proceedings of the Workshop on Hot Topics in Operating Systems - HOTOS*, volume Part F129307, page 156–161. IEEE Computer Society, May 2017.

[BBH18] Brendan Burns, Joe Beda, and Kelsey Hightower. *Kubernetes*. Dpunkt, 2018.

[BBH19] Brendan Burns, Joe Beda, and Kelsey Hightower. *Kubernetes: up and running: dive into the future of infrastructure*. O'Reilly Media, 2019.

[Bel19] Pavan Belagatti. "Docker Swarm or Kubernetes?": Is It the Right Question to Ask? *Dzone*, Oct 2019.

[BFW13] Ulrik Brandes, Linton C Freeman, and Dorothea Wagner. *Social networks*. 2013.

[Bib19] Microservices Inter-Service Communication - Dinesh on Java, May 2019. [Online; accessed 26. Oct. 2021].

[Bra13a] Rick Branson. Messaging at Scale at Instagram, 2013. [Online; accessed 6. Dec. 2021].

[Bra13b] Rick Branson. Slides: Messaging at Scale at Instagram. 2013. [Online; accessed 6. Dec. 2021].

[But06] P Butler. Review of the Literature on Portfolios and Eportfolios. *Retrieved April*, 13:2009, 2006.

[C21] Arghya C. WebAssembly - What it is & Why is it so important, Jul 2021.

[Cae20] Tino Caer. How Microsoft Is Adopting Rust - Tino Caer - Medium. *Medium*, Aug 2020.

[Cal20] Lee Calcote. *The enterprise path to service mesh architectures*. O'Reilly Media, Incorporated, 2020.

[CB09] Carolina Cañibano and Barry Bozeman. Curriculum vitae method in science policy and research evaluation: the state-of-the-art. *Research Evaluation*, 18(2):86–94, 2009.

[CB19] Lee Calcote and Zack Butcher. *Istio: Up and Running: Using a Service Mesh to Connect, Secure, Control, and Observe*. O'Reilly Media, 2019.

[CB+20] Ramaswamy Chandramouli, Zack Butcher, et al. Building secure microservices-based applications using service-mesh architecture. *NIST Special Publication*, 800:204A, 2020.

[Che17] Jinshi Chen. An e-portfolio-based model for the application and sharing of college english esp moocs. *Higher Education Studies*, 7(2):35–42, 2017.

[CNC20] CNCF. CNCF Cloud Native Interactive Landscape, 2020.

[Com] European Commission. ESCO Strategic framework.

[Com17] European Commission. *ESCO handbook*. Number September. 2017.

[Com18] European Commission. *Promoting online training opportunities for the workforce in Europe: Interim report*. Publications Office of the European Union, Oct 2018.

[Com21a] European Commission. E-Portfolio and Web-based Tools, 2021.

[Com21b] European Commission. Europass e-Portfolio, 2021.

[CS11] Joseph Check and Russell K Schutt. *Research methods in education*. Sage Publications, 2011.

[Dav85] Fred D Davis. *A technology acceptance model for empirically testing new end-user information systems: Theory and results*. PhD thesis, Massachusetts Institute of Technology, 1985.

[DBW89] Fred D Davis, Richard P Bagozzi, and Paul R Warshaw. User acceptance of computer technology: A comparison of two theoretical models. *Management science*, 35(8):982–1003, 1989.

[DSJVPM19] Magdaleen De Swardt, Louis S Jenkins, Klaus B Von Pressentin, and Robert Mash. Implementing and evaluating an e-portfolio for postgraduate family medicine training in the western cape, south africa. *BMC medical education*, 19(1):1–13, 2019.

[EMZ19] Amine El Malki and Uwe Zdun. Guiding architectural decision making on service mesh based microservice architectures. In *European Conference on Software Architecture*, pages 3–19. Springer, 2019.

[Eura] European Commission. ESCO: Connecting education and training with the labour market.

[Eurb] European Commission. ESCO: harnessing the power of big data analysis for a functional labour market.

[Eur21] European Commission. ESCO Skill-Occupation Matrix Tables: linking occupation and skill groups. (April), 2021.

[FL14] Martin Fowler and James Lewis. Microservices, 2014. *martinFowler*, 1(1):1–1, 2014.

[For16] Cipriano Forza. Surveys. In *Research Methods for Operations Management*, pages 95–180. Routledge, New York, NY, USA, May 2016.

[Gee17] Michael Geers. Micro Frontends. *Micro Frontends*, Aug 2017.

[Gee20] Michael Geers. *Micro Frontends in Action*. 2020.

[Geo12] Fred George. μService Architecture:A Personal Journey of Discovery. 2012.

[GMNHSR20] Gabriel García-Murillo, Pavel Novoa-Hernández, and Rocío Serrano Rodriǵuez. Technological acceptance of Moodle through latent variable modeling – a systematic mapping study. *Interactive Learning Environments*, pages 1–17, Dec 2020.

[GOS09] Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer, 2009.

[Gru09] Tom Gruber. Ontology. *Encyclopedia of database systems*, 1:1963–1965, 2009.

[GUR17] Emet GURL. Swot analysis: A theoretical review. 2017.

[HC10] Alan Hevner and Samir Chatterjee. Design science research in information systems. In *Design research in information systems*, pages 9–22. Springer, 2010.

[HLT02] Mounira Harzallah, Michel Leclère, and Francky Trichet. Commoncv: modelling the competencies underlying a curriculum vitae. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 65–71, 2002.

[HMPR04] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

[Hof16] Kevin Hoffman. *Beyond the Twelve-Factor App*. O'Reilly Media, Inc., Sebastopol, CA, USA, Apr 2016.

[HPBB05] Veronika Hornung-Prähauser, Wernher Behrendt, and Motti Benari. Developing further the concept of ePortfolio with the use of Semantic Web Technologies. page 12, 2005.

[IS18] Kasun Indrasiri and Prabath Siriwardena. Service mesh. In *Microservices for the Enterprise*, pages 263–292. Springer, 2018.

[Jac19] Cam Jackson. Micro Frontends, 2019. [Online; accessed 29. Oct. 2021].

[Jan18] Kinnary Jangla. Containers. In *Accelerating Development Velocity Using Docker*, pages 1–8. Springer, 2018.

[JLMH20] Caroline Jobin, Pascal Le Masson, and Sophie Hooge. What does the proof-of-concept (poc) really prove? a historical perspective and a cross-domain analytical study. In *XXIXème conférence de l'Association Internationale de Management Stratégique (AIMS)*, 2020.

[Jon18] M. Tim Jones. Why you should learn the Rust programming language. *IBM Developer*, Mar 2018.

[Kah20] Adham Kahlawi. An Ontology Driven ESCO LOD Quality Enhancement. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 11(3), 2020.

[Kea21] Anja Kammer and et. al. Service Mesh Feature Comparison, 2021.

[KK20] Anjali Khatri and Vikram Khatri. *Mastering Service Mesh: Enhance, secure, and observe cloud-native applications with Istio, Linkerd, and Consul*. Packt Publishing Ltd, 2020.

[Kla18] Steve Klabnik. *Rust in Production*. Øredev'17. Oct 2018.

[Kri21] Paul Krill. Big Next.js upgrade has new Rust compiler, ES Modules support. *InfoWorld*, Oct 2021.

[KSLB18] Sangkyun Kim, Kibong Song, Barbara Lockee, and John Burton. What is gamification in learning and education? In *Gamification in learning and education*, pages 25–38. Springer, 2018.

[LCAS19] Wilma Lorena Gavilanes López, Blanca Rocio Cuji, Maria José Abásolo, and Gladys Lorena Aguirre Sailema. Technological Acceptance Model (TAM) using Augmented Reality in University Learning Scenarios. In *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE, Jun 2019.

[Lew12] James Lewis. Micro services - Java, the Unix Way. 2012.

[LI05] George Lorenzo and John Ittelson. An Overview of E-Portfolios. 2005.

[LLG+19] Wubin Li, Yves Lemieux, Jing Gao, Zhuofeng Zhao, and Yanbo Han. Service mesh: Challenges, state of the art, and future research opportunities. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 122–1225. IEEE, 2019.

[LMG04] Douglas Love, Gerry McKean, and Paul Gathercoal. Portfolios to Webfolios and Beyond: Levels of Maturation. 2004.

[Luk17] Marko Luksa. *Kubernetes in action*. Simon and Schuster, 2017.

[Mah20] Omar MK Mahasneh. A proposed model for the university students' e-portfolio. *Journal of Education and e-Learning Research*, 7(1):28–33, 2020.

[Mar21a] Paulo Martins. From data to knowledge - digital literacy in the service of corpora. In *Autumn NetLang Workshop*, Nov 2021.

[Mar21b] Martins, Paulo and Costa, Leandro and Ramalho, José Carlos. Major Minors - Ontological Representation of Minorities by Newspapers. In *10th Symposium on Languages, Applications and Technologies (SLATE 2021)*, volume 94 of *Open Access Series in Informatics (OASIcs)*, pages 3:1–3:13, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Mat20] Philian Mateo. Top 6 Programming Languages 2020 (It is worth the attention), 2020.

[McL16] Matt McLarty. Microservice architecture is agile software architecture, May 2016.

[MCR21] Paulo Martins, Leandro Costa, and José Carlos Ramalho. Knowledge Graph of press clippings referring social minorities. In *Linked Archives*. CEUR-WS, 2021.

[MFS20] P Mariano, D Filippo, and F Santoro. Design science research: fazendo pesquisas científicas rigorosas atreladas ao desenvolvimento de artefatos computacionais projetados para a educação. *Metodologia de Pesquisa Científica em Informática na Educação: Concepção de Pesquisa. SBC*, 2020.

[Mic21] Microsoft. Communication in a microservice architecture, Oct 2021. [Online; accessed 26. Oct. 2021].

[Miń18] Piotr Mińkowski. Communicating Between Microservices. *Dzone*, Mar 2018.

[Mor11] Robin D Morris. Web 3.0: Implications for online learning. *TechTrends*, 55(1):42–46, 2011.

[MS03] Catherine C. Marshall and Frank M. Shipman. Which semantic web? *The fourteenth ACM conference*, page 57, 2003.

[MTTMG+08] M. R. Martínez-Torres, S. L. Toral Marín, F. Barrero García, S. Gallardo Vázquez, M. Arias Oliva, and T. Torres. A technological acceptance of e-learning tools used in practical and laboratory teaching, according to the European higher education area. *Behaviour & Information Technology*, 27(6):495–505, Nov 2008.

[NGJ+19] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs: lessons and challenges: five diverse technology companies show how it's done. *Queue*, 17(2):48–75, 2019.

[NI14] L. T. Nguyen and M. Ikeda. ePortfolio System Design Based on Ontological Model of Self-Regulated Learning. In *2014 IIAI 3rd International Conference on Advanced Applied Informatics*, page 301–306, Aug 2014.

[Nic19] Carol Nichols. *Rust: A Language for the Next 40 Years*. Emerging Technologies for the Enterprise Conference. May 2019.

[NJ71] Jay F Nunamaker Jr. A methodology for the design and optimization of information processing systems. In *Proceedings of the May 18-20, 1971, spring joint computer conference*, pages 283–294, 1971.

[NJCP90] Jay F Nunamaker Jr, Minder Chen, and Titus DM Purdin. Systems development in information systems research. *Journal of management information systems*, 7(3):89–106, 1990.

[npm19] Community makes Rust an easy choice for npm here The npm Registry uses Rust for its CPU-bound bottlenecks. 2019.

[Nut21a] Seiya Nuta. Kerla, Oct 2021.

[Nut21b] Seiya Nuta. Writing a Linux-compatible kernel in Rust, Oct 2021.

[Opp21a] Philipp Oppermann. Redox - Your Next(Gen) OS, Jul 2021.

[Opp21b] Philipp Oppermann. Writing an OS in Rust, Oct 2021.

[Ort20] Basti Ortiz. Rust Reviewed: Is the hype justified?, 2020.

[Pel21] Miika Peltotalo. A case study on cloud migration and improvement of twelve-factor app. 2021.

[Pon15] Julie Ponto. Understanding and Evaluating Survey Research. *J. Adv. Pract. Oncol.*, 6(2):168, Mar 2015.

[Pre21] Charles Pretzer. Service mesh up and running with linkerd. 2021.

[Pro21] Chromium Project. Memory safety - The Chromium Projects, 2021.

[PTR08] K. Peffers, T. Tuunanen, and M. Rothenberger. Methodology for information systems research. 2008.

[PTRC07] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[PTWS20] Thomas Pellissier-Tanon, Gerhard Weikum, and Fabian Suchanek. Yago 4: A reason-able knowledge base. *ESWC 2020*, 2020.

[Rai19] Matt Raible. How to Win at UI Development in the World of Microservices, 2019.

[Ray03] Eric S Raymond. *The art of Unix programming*. Addison-Wesley Professional, 2003.

[Res21] Bob Reselman. An illustrated guide to 12 Factor Apps. *Red Hat*, Feb 2021.

[Ric] Comité Richelieu. Le médiateur des entreprises (2019). *Osez l'innovation, De l'idée à l'industrialisation: réussissez votre preuve de concept, Ministère de l'Economie et des Finances, Paris*.

[RL11] Rajiv and Manohar Lal. Web 3.0 in Education. *International Journal of Information Technology*, 3(2):973–5658, 2011.

[RMG17a] Kalthoum Rezgui, Hédia Mhiri, and Khaled Ghédira. Ontology-based e-Portfolio modeling for supporting lifelong competency assessment and development. *Procedia Computer Science*, 112:397–406, Jan 2017.

[RMG17b] Kalthoum Rezgui, Hédia Mhiri, and Khaled Ghédira. Ontology-based e-Portfolio modeling for supporting lifelong competency assessment and development. *Procedia Computer Science*, 112:397–406, January 2017.

[RMSG18] Kalthoum Rezgui, Hedia Mhiri Sellami, and Khaled Ghédira. Towards a common and semantic representation of e-portfolios. *Data Technologies and Applications*, 52, August 2018.

[Rus17] Rust. Ashley Williams - How I Convinced the World's Largest Package Manager to Use Rust, and So Can You!, May 2017.

[rus21] Stack Overflow Developer Survey 2020, Oct 2021.

[RWC+15] Gareth Roy, Andrew Washbrook, David Crooks, Gang Qin, Samuel Cadellin Skipsey, Gordon Stewart, and David Britton. Evaluation of containers as a virtualisation alternative for hep workloads. In *Journal of Physics: Conference Series*, volume 664, page 022034. IOP Publishing, 2015.

[Sar20] Edwin M Sarmiento. Introduction to containers. In *The SQL Server DBA's Guide to Docker Containers*, pages 1–8. Springer, 2020.

[Say17] Gigi Sayfan. *Mastering kubernetes*. Packt Publishing Ltd, 2017.

[Sch13] Andreas Schmidbauer. *Showcase E-Portfolio usage in the workplace. The beneficial and disruptive potential in the context of information silos*. 2013.

[SF08] Neal Sumner and Olivia Fox. Implementing an institution-wide e-portfolio: The city university experience.(telling deep stories about institutional change). In *e-Learning*, pages 311–316, 2008.

[Sha06] Victoria Shannon. A 'more revolutionary' Web - The New York Times, 2006.

[SHO13] R. Arteaga Sánchez, A. Duarte Hueros, and M. García Ordaz. E-learning and the University of Huelva: a study of WebCT and the technological acceptance model. *Campus-Wide Information Systems*, Mar 2013.

[Sin12] Amit Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 5:16, 2012.

[SMG19] Hafizan Mat Som, Ahmad Muhaimin Mohamad, and Amzari Jihadi Ghazali. An E-portfolio Application for MSTP Students: What Can We Learn from It? *International Academic Journal of Social Sciences*, 6(1):18–30, 2019.

[Ste10] Paul Stephensen. *Designing ePortfolios for Music postgraduate study. A practice-led inquiry*. PhD thesis, Queensland University of Technology, 2010.

[Sti94] Richard J Stiggins. *Student-centered classroom assessment*. Merrill New York, 1994.

[SZH] Meina Song, Chengcheng Zhang, and E. Haihong. An Auto Scaling System for API Gateway Based on Kubernetes. In *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, pages 23–25. IEEE.

[TdSDAKT20] Jaime A Teixeira da Silva, Judit Dobránszki, Aceil Al-Khatib, and Panagiotis Tsigaris. Curriculum vitae: challenges and potential solutions. *KOME: AN INTERNATIONAL JOURNAL OF PURE COMMUNICATION INQUIRY*, 8(2):109–127, 2020.

[Thö15] Johannes Thönes. Microservices. *IEEE software*, 32(1):116–116, 2015.

[Tom21] Tomaka. Redshirt, Oct 2021.

[Tun21] Liam Tung. Google backs effort to bring Rust to the Linux kernel. *ZDNet*, Apr 2021.

[Tze11] Jeng-Yi Tzeng. Perceived values and prospective users' acceptance of prospective technology: The case of a career eportfolio system. *Computers & Education*, 56:157–165, Jan 2011.

[Vas19] Jerry J. Vaske. *Survey Research and Analysis, 2nd Edition.* Sagamore-Venture. 1807 North Federal Drive, Urbana, IL 61801. Tel: 800-327-5557; Tel: 217-359-5940; Fax: 217-359-5975. Web site: https://www.sagamorepub.com/, Urbana, IL, USA, 2019.

[VHP+21] R. Verborgh, K. Hose, H. Paulheim, P. A. Champin, M. Maleshkova, O. Corcho, P. Ristoski, and M. Alam. *The Semantic Web*. Springer International Publishing, Cham, Switzerland, 2021.

[Vla19] Charis Vlados. On a correlative and evolutionary swot analysis. *Journal of Strategy and Management*, 2019.

[W3C19] W3C. World Wide Web Consortium (W3C) brings a new language to the Web as WebAssembly becomes a W3C Recommendation, Dec 2019.

[Wan09] Shouhong Wang. E-Portfolios for Integrated Reflection. 2009.

[Wig21] Adam Wiggins. The Twelve-Factor App, Oct 2021.

[Wil17] Ashley Williams. *How I convinced the world's largest package manager to use Rust, and so can you!* May 2017.

[WRPP19] Conrad Watt, Andreas Rossberg, and Jean Pichon-Pharabod. Weakening webassembly. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–28, 2019.

[WSBH16] Andi Wilson, Ross Schulman, Kevin Bankston, and Trey Herr. Bugs in the system. 2016.

[WW05] Ing-Long Wu and Ker-Wei Wu. A hybrid technology acceptance approach for exploring e-crm adoption in organizations. *Behaviour & Information Technology*, 24(4):303–316, 2005.

[Yas17] Amir Yasin. *High performance apps with JavaScript and Rust, it's easier than you think*. May 2017.

[YPB18] Fei Yu, Michele Pasinelli, and Alexander Brem. Prototyping in theory and in practice: A study of the similarities and differences between engineers and designers. *Creativity and Innovation Management*, 27(2):121–132, 2018.

[Zim17] Olaf Zimmermann. Microservices tenets. *Comput. Sci. Res. Dev.*, 32(3):301–310, Jul 2017.

[ZJJ18] J. T. Zhao, S. Y. Jing, and L. Z. Jiang. Management of API Gateway Based on Micro-service Architecture. *J. Phys. Conf. Ser.*, 1087(3):032032, Sep 2018.

Part IV

APPENDIX

A

SOURCE-CODE (GITHUB)

- **Hyperfolio - Cloud:** https://github.com/Paulo-Jorge-PM/hyperfolio-cloud
- **Hyperfolio - Frontend:** https://github.com/Paulo-Jorge-PM/hyperfolio-cloud
- **Hyperfolio - Mobile:** https://github.com/Paulo-Jorge-PM/hyperfolio-mobile
- **Hyperfolio - Miscellaneous:** https://github.com/Paulo-Jorge-PM/hyperfolio
- **EntiGraph tool:** https://github.com/Paulo-Jorge-PM/entigraph

B

SCIENTIFIC OUTPUTS

In this Appendix we will summarize scientific outputs (Academia) that we achieved during 2021 during the execution of the present project, which tools were integrated/developed in both. We highlight the Major Minors and Netlang R&D projects, which share many of the tools developed during the current project (e.g. EntiGraph release during this thesis), and which were developed during the same period during 2021 and tested/integrated in both projects.

B.1 PRIZES

- 1st Prize in the annual Arquivo.pt competition (June 2021), with the project *Major Minors* (which some tools were also developed during the current thesis, namely EntiGraph, released now with version 1.0). Homepage at: http://minors.ilch.uminho.pt/press

B.2 PUBLISHED PAPERS

- Paulo Martins, Leandro Costa, & José Ramalho (2021). Major Minors - Ontological representation of minorities by newspapers. In OASIcs, Volume 94, SLATE 2021. URL: https://www.dagstuhl.de/dagpub/978-3-95977-202-0. DOI: https://doi.org/10.4230/OASIcs.SLATE.2021.3.
- Paulo Martins, Leandro Costa, & José Ramalho (2021). Implementation of a Knowledge Graph of press clippings representing social minorities. In Linked Archives 2021 - International Workshop on Archives and Linked Data. INESC TEC, Portugal. Url: https://linkedarchives.inesctec.pt/. [in press]

B.3 CONFERENCES

- Paulo Martins (2021). Autumn Netlang Workshop. "From Data to Knowledge – Digital Literacy at the Service of Corpora" (Presentation of the EntiGraph tool). University of Minho, Portugal. 11/11/2021. URL: http://cehum.ilch.uminho.pt/events/500

- Paulo Martins, Leandro Costa, & José Ramalho (2021). Conversas@Rossio. "Save the data: Como os arquivos web combatem a desinformação". UNL, Portugal. 28/09/2021. URL: https://rossio.fcsh.unl.pt/2021/09/08/ciclo-de-conversasrossio-com/

- Paulo Martins, Leandro Costa, & José Ramalho (2021). "Major Minors com o Arquivo.pt no Dia Mundial da Preservação Digital". Arquivo.pt, Portugal. 04/11/2021. URL: https://sobre.arquivo.pt/pt/major-minors-com-o-arquivo-pt-no-dia-mundial-da-preservacao-digital-2021/

- Paulo Martins, Leandro Costa, & José Ramalho (2021). Linked Archives 2021 - International Workshop on Archives and Linked Data. INESC TEC, Portugal. 13/09/2021. Url: https://linkedarchives.inesctec.pt/.

- Paulo Martins, Leandro Costa, & José Ramalho (2021). SLATE'21 - 10th Symposium on Languages, Applications and Technologies. School of Technology, Polytechnic Institute of Cávado and Ave, Portugal. 01/07/2021. Url: http://slate-conf.org/2021/home.

B.4 COLLABORATIONS

- NetLang project: the developed tool EntiGraph, developed and released during the current project, is at the moment also being ported to the NetLang FCT R&D project (2021-2022), in which we are currently also a team member. Project website: https://netlang-corpus.ilch.uminho.pt/

- Major Minors: we were team members and coordinators, finishing this project in June 2021 with the Arquivo.pt 1st prize. The first early versions of the EntiGraph tool, very incomplete and without CLI/GUI, were developed in that context, further expanded and released as 1.0 open-source format during the current dissertation development. Project website: http://minors.ilch.uminho.pt/

C

SURVEY QUESTIONNAIRE 1 - MARKET PROSPECTION

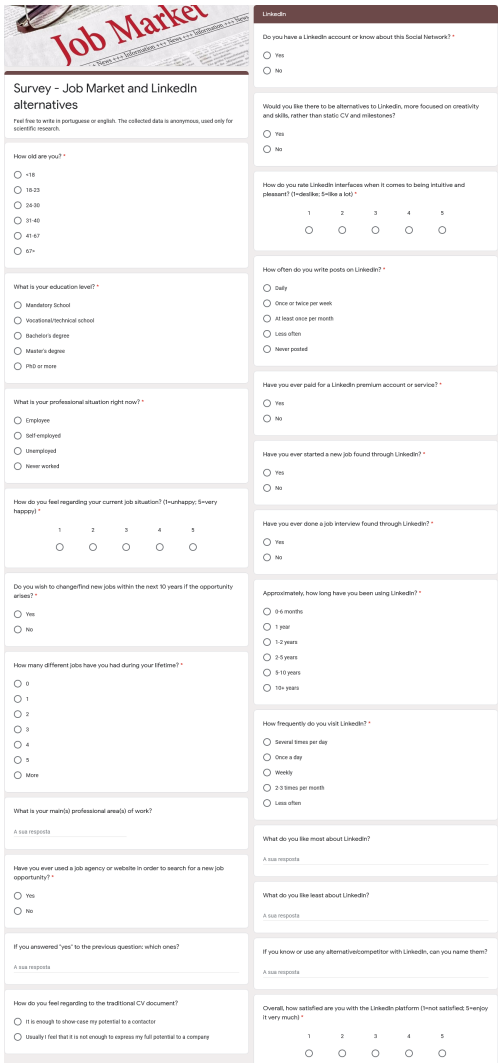

Figure 65: Survey 1: Market Prospection

D

SURVEY QUESTIONNAIRE 2 - PROTOTYPE EVALUATION BASED ON THE TECHNOLOGICAL ACCEPTANCE MODEL

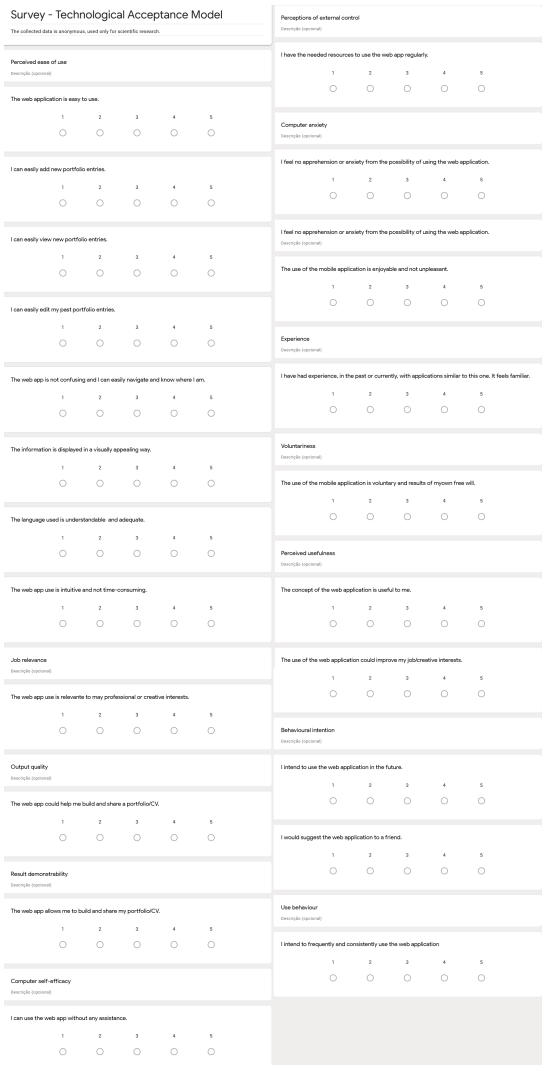

Figure 66: Survey 2: Technological Acceptance Model