UNIVERSITY OF TRENTO

Department of Information Engineering and Computer Science

Master's Degree in
Computer Science

Final Dissertation

Kampas: Streamlining Cybersecurity Control Management

Supervisor

Prof. Silvio Ranise

Student

Giacomo Zanolli

Academic year 2022/2023

Abstract

Cyberattacks are on the rise and companies need to periodically enhance their defenses if they want to safeguard themselves and their customers. Securing a company, however, is a complex task and it can be challenging to be informed about the current best practices in all its aspects.

This is one of the reasons why the so-called cybersecurity frameworks have been introduced and new ones are being developed. These standards and guidelines provide their readers with well-defined taxonomies of the cybersecurity landscape, helping them in choosing what measures are most important to implement to achieve an adequate cybersecurity posture.

Most companies, however, do not have the time or the resources needed to integrate all the guidelines of a framework -we call these *controls*- simultaneously, and some of those guidelines may even be inapplicable to them. There is the need for a selection of the controls tailored to each company. Moreover, the controls expressed in the frameworks can at times be too abstract or vague in their definition to be implemented directly. If a company chooses to use multiple frameworks, then, it is probable that some controls will overlap in full or in part, creating the need for a mapping between the two frameworks to know what has not been covered yet. Finally, the implementation of even a single control may require a company a significant investment of time and effort. Selecting which controls are applicable to a company, prioritizing them and tracking the progress of their implementation can therefore be considered a necessity.

In this work, we provide an analysis of the problem starting from its root cause: the need to secure a company. Then, we examine the ways in which cybersecurity frameworks aim to help a company in this task, showing that there exist several aspects which could become more streamlined. We perform a requirement analysis of the functionalities that the users of a potential solution would need from it and survey existing solutions. This analysis yields the result that there may be a vacant place for an open source, framework-agnostic solution that helps with the problem of managing cybersecurity controls. After having characterized the problem formally, we proceed in designing a software system which can fulfill the identified requirements -called kampas- and to detail the steps of its implementation. We provide a preliminary evaluation of its functionalities through a realistic test in which we use it to manage the guidelines of two popular cybersecurity frameworks. Finally, we summarize the results of the work and provide ideas for possible future developments.

Our results encourage us in thinking that the system proposed in this work may be effective in helping its prospective users in managing the cybersecurity controls they decide to implement in their organization, helping them to prioritize such controls to optimize their implementation efforts, integrate new controls, and see how the company is faring under the point of view of multiple frameworks.

Contents

List of Figures

List of Tables

List of Equations

List of Listings

List of Algorithms

1 Introduction

In the recent decades, companies have become more and more digitalized, in a process that does not show any sign of slowing down [27]. This modernization brings them several benefits, especially in terms of added flexibility and improved efficiency. Yet, this same transformation is bound to also expose them to novel challenges and risks. In this thesis, we are concerned with the ones related to cybersecurity.

The more a company becomes digitalized, the more it becomes at risk of cyber attacks, ranging from ransomware to denial of service to data exfiltration, just to name a few [40]. The profile of the average attackers has evolved over the years, going from the curious tech enthusiast that it was in the 1980s and '90s to what today are fully-qualified companies, capable of carrying out sophisticated attacks over long periods of time [14], [32], [33].

To cope with this type of threat, companies need to integrate into their systems, processes and policies appropriate measures of cybersecurity. These range from the application of general principles -like the one of "least-privilege"-, to policies -like mandating that all corporate accounts have a secure and unique password-, to tools and services that can help automatically scan the assets of the organization and single out the ones that show signs of potential misconfigurations or vulnerabilities.

Every company that takes their cybersecurity seriously will have at least one person dedicated to supervising it. This individual is generally referred to as the CISO -Chief Information Security Officer-. Their role is to assess and improve the cybersecurity posture of the company over time, ensuring that the measures in place within the company provide adequate protection from external -and internal- threats. Some companies -for example those operating with sensitive or financial information- can also have the legal requirement to comply with regulations and standards that detail specific cybersecurity objectives and practices, like the GDPR [21]. Finally, maintaining a healthy cybersecurity posture can also represent a requisite from the business prospective for a company, as some potential customers -especially those in the financial, energy or military sector- may be bound by law to only do business with companies that meet a set of standards and certifications -e.g., the ISO27001 [37]-.

Securing their digital realm is therefore a necessity for a multitude of realities. Wether it is done for a moral imperative of keeping their customer's data safe, or a desire to avoid the damages or sanctions that may stem from an attack. Still, securing a company -and keeping it secure- is a task that quickly grows in complexity. This is given by a variety of factors, including: (i) the diversity of measures that need to be put in place to patch all the potential existing vulnerabilities, (ii) the need of specialized people that understand the multiple facets of cybersecurity -application security, identity and access management, compliance, vulnerability management, &c.-, (iii) the highly dynamic nature of the field of cybersecurity, where new tools, techniques and vulnerabilities get introduced everyday, and (iv) the fact that very things being defended -the assets of the company- are in perennial change as new products and features get introduced, adopted, updated and discontinued.

Securing a company, therefore, is no easy task and the possibility of missing some important aspects that adversaries may exploit is real. This is one of the reasons why reputable entities such as the NIST and the CIS publish frameworks and guidelines: to provide companies with a checklist of measures they should implement to achieve a reasonably comprehensive cyber-defense. These documents -e.g., the NIST *Cybersecurity Framework (CSF) v1.1* [49] and the CIS *Critical Security Controls v8* [12]- usually contain numerous indications of what a company can do to improve its security posture. We will refer to this concept as *"controls"*.

Integrating the guidelines from these documents in a real-world company, however, is a feat that can span across a significant period of time, requiring considerable and sustained investments in terms of capital, time and skilled work. Just to give an idea, the NIST CSF defines 108 security controls and the CIS has 153. Each of the measures that the company may want to adopt to satisfy a control requires careful and informed considerations to produce an assessment of what the impact of its

application would be on the company and determine wether the benefits of applying such a measure outweigh the costs of its implementation or not.

Another degree of complexity is introduced by the fact that the controls of the frameworks can often be given using a generic formulation, which needs to be interpreted before applying it to the reality at hand. Standards and regulations can also be updated and amended, and new ones can be released or adopted by the company. Finally, a control may overlap in full or in part with another from a different framework, making it unclear if some further actions need to be taken to satisfy it. Keeping track of these controls and their status, therefore, risks to quickly become unmanageable as the number of them grows.

Given this, the broad-level problem we strive to solve is the following: helping the company's employees to make sense of the -likely vast- amount of controls they may want to satisfy, optimize their implementation efforts and manage the addition of new controls from other frameworks.

Before proceeding further, we examine in greater detail the task of optimizing the implementation effort. It consists in identifying which measures the company should prioritize to satisfy as many of its controls as possible with the minimum implementation effort.

After giving a precise definition of the terminology, we provide a formal characterization of this problem. We show how it is possible to reconduct it to being an instance of a known computational problem: the set cover problem. By doing this, we are able to prove that all the known algorithms that solve the set cover problem can be used for our problem as well. We then proceed by refining the assumptions in how we model the problem, progressively introducing variants which, in their definition, account for more variables which we deem important for the decision process of which measure should be prioritized. We show how each variant can be reduced to the previous ones by adding some constraints on the additional variables that are introduced with respect to them. We also compute the dimension of the search space for our problem and justify our choices for choosing a greedy approximation algorithm to solve it.

1.1 Kampas

Having identified the aforementioned problem, we perform an analysis of the requirements that a potential solution may need to satisfy. At its core, the aim of the solution is to help its users: (i) gather in one place all the controls their company wants to implement, (ii) assign to each of them one or more actionable *measures* which represent steps that the company can concretely undertake to implement the control in its reality, (iii) inspect what the progress in the implementation of a particular control -or class of controls- is in order to validate that the implementation is yielding the expected results, and (iv) integrate additional security controls among the existing ones, be them from a new framework or because of specific requirements of the company.

With this in mind, we search for existing solutions that may be able to provide the desired features. In our analysis, we include both academic works and commercial products, comparing their strengths, weaknesses and fitness for the identified task. Overall, we observe how the tools which are freely accessible provide only a part of the required features. At the same time, whilst multiple commercial solutions do seem to offer the required capabilities to a more comprehensive degree, we were unable to access any of them for testing due to the requirement of purchasing a commercial license. The result of this analysis, therefore, highlights how there appears to be a vacant place for an open source solution which integrates the required features. An open source product would also bring the intrinsic benefits of being publicly auditable and arguably more apt to being customized to the requirements of its end-users with respect to commercial alternatives.

In the light of this, we detail the process of designing and implementing a software system -called kampas- that could help the user achieving the desired functions. We build its core using the Rust programming language, which can provide fast performance whilst guaranteeing numerous safety properties in the memory management of our program, preventing a whole class of memory-related security vulnerabilities [73]. For the database, we used a multi-model solution -also written in Rust- which offers the developers flexibility in how they choose to represent the program's data without compromising on performance. End-users can interact with the system through a web UI, developed using the SvelteKit framework.

1.2 Outcomes

To perform a preliminary validation of our work, we test it in a realistic example of a fictitious medium-sized financial company looking to adopt the controls from the *Framework for Improving Critical Infrastructure Cybersecurity (NIST CSF), version 1.1* [49] and the *CIS Critical Security Controls, version 8* [12]. During the test, we first gauge the capability of the software of handling a single cybersecurity framework by using it to store the 108 controls of the NIST CSF and rank their associated measures according to their priority. Then, we show how it is possible to manage the integration of 153 additional controls from the CIS' framework and how existing measures can contribute positively to the completion of the newly-added controls. Lastly, we show how it is possible to view the progress according to multiple frameworks.

This test yielded encouraging results, leading us to deem the development phase of this first version of the program concluded.

Given the state reached by the project, and considering that it was built from scratch, we regard it as having met our initial objectives and as being capable of starting to deliver its value to the users. We do have a list of additional functionalities that would, in our opinion, further enhance its capabilities and hope that the work described in this thesis can serve as a starting point for the development of a reliable assistant that can smooth the path towards getting our world more resilient in the cybersecurity realm.

2 The problem

This chapter describes the initial portion of the work of this thesis. First, we acknowledge the necessity of the industry to achieve a good cybersecurity posture in Section 2.1. Then, we overview some of the challenges that a company faces when doing so Section 2.2. In Section 2.3, we consider one of the ways which businesses are using to try to streamline the process: cybersecurity frameworks. We detail the principal concepts of two popular frameworks and highlight their points of strength. Subsequently, we begin analyzing their shortcomings in Section 2.4, generating an explicit framing for the problem we tackle in the rest of this work. Finally, we formalize such a problem in Section 2.5 and map it to a known computational problem.

2.1 A need for cybersecurity

Digitalization is a term commonly used to refer to the process of companies integrating digital technologies in their systems and processes [10].

As a company becomes more digitalized, it is able to streamline some tasks that previously were manual and repetitive, and to become more agile in adapting to the changing demands of its customers.

Yet, as a company adopts more and more digital systems, it creates a bigger and bigger attack surface that malicious actors may use to infiltrate their systems [40].

A cyber attack may be carried out for a variety of reasons. According to the reports of Middleton, from the 80s until now, we have assisted to an evolution in the profile and the motives of the average attacker [47]. In the 1908s, many attacks were performed by individuals, and often out of curiosity or the will to unlock premium features for their personal accounts. In time, however, there has been an emergence of more and more sophisticated attacks, increasingly seeking financial gain [69]. A series of criminal organizations have spawn up to profit from cybercrime [69]. Some try to steal sensitive data from their victims to then sell it in dark web markets, others distribute ransomware that cyphers all the data in a system until a ransom is paid [50]. There also exists a whole set of companies that sell software or information which is highly sensitive, like Zerodium[1] providing undisclosed zero-day security vulnerabilities and the NSO Group[2], which manages software like the Pegasus spyware[3].

The threat of a cyber attack looms on everyone that possesses digital devices, especially those connected to the internet [43]. Companies, therefore, need to take proactive steps in order to review their cybersecurity posture and improve it over time.

2.1.1 Motivations

In this section, we detail some of the motivations for why a company may want to invest its resources in improving its security posture.

2.1.1.1 Avoiding economic damage

Some sources estimate that cybercrime will cost, at the global level, 8 trillion American dollars in 2023, and 10.5 trillion in 2025 [19].

The economic damage of a cyber attack may come in multiple forms. The attackers may damage some equipment or destroy data, rendering them unusable or causing downtime. They could cipher the data, and demand ransoms to decrypt it. Alternatively, they may exfiltrate the data and demand a payment under the threat of rendering it public otherwise.

Another way in which a company may sustain an economic impact is by the loss of revenue that a reputation damaged by a cyber attack can bring. Existing clients may no longer feel safe doing

[1] `https://zerodium.com`

[2] `https://nsogroup.com`

[3] `https://www.theregister.com/2023/01/09/supreme_court_pegasus_spyware`

business with that company and prospective clients may be scared off. According to a report of the Clusit for 2023, 25% of the companies that experienced a data breach saw their valuation diminishing by 7% to 17% after the fact [14].

2.1.1.2 Maintain trade secrets

The intellectual property of a company can represent for them a key competitive advantage with respect to their competitors. Safeguarding it can therefore be seen as business priority, as there have already been cases of breaches involving trade secrets [5], [20].

2.1.1.3 Regulatory compliance

Lawmakers have the power to impose requirements to the companies operating within their jurisdiction, sanctioning the ones that do not abide to them. One notorious example of this is the GDPR -General Data Protection Regulation- [21], which bounds companies that provide services to European Citizens to perform a proper risk assessment for the personal identifiable information they collect and adopt adequate safeguards to protect it. It also requires companies to give the users the possibility to request the correction or deletion of their personal information.

If a company does not comply with the applicable regulations, the consequences can be hefty. One such example is the case of Facebook, fined for 5 billion dollars in the wake of the Cambridge Analytica scandal [22] or British Airways, fined 185 million British pounds for a data breach that involved five hundred thousand people [11].

2.2 How do you secure a company?

In the previous section, we saw why companies should devote part of their resources towards achieving a good cybersecurity posture. Still, securing a company is not a trivial task. In this section, we explore some of the reasons that quickly escalate its complexity.

2.2.1 Cybersecurity is vast and evolving

Some sources have defined cybersecurity as the art of protecting ICT systems [26], encompassing the measures that are aimed to safeguard such systems from threats and the effort of improving these measures over time. Often times, the goals of securing a system are aimed at preserving one or more of the properties of its information expressed in the CIA triad -confidentiality, integrity and availability-.

The cybersecurity landscape is composed by a variety of categories. To our knowledge, there is no standard way to enumerate these categories, therefore we report here some of the most common ones, inspired mainly by [29], [49]:

1. Identity Management: the process of creating digital identities for users -be them people or machines-, reliably binding them to such identity and managing the lifecycle of the identity -e.g. credential recovery-;

2. Endpoint Security: the set of practices for ensuring that the machines of the company -e.g., the employees' workstations and the servers- comply with the company policies and do not show signs of suspicious activity, which may indicate a compromise;

3. Asset Management: the set of processes aimed to inventory and track all the ICT assets of the company, be them employees' workstations, servers, hard disks, displays, printers, networking equipment, IoT devices &c. These assets should be tracked during their entire lifecycle, going from their purchase to their disposal;

4. Vulnerability Management and Assessment: the set of techniques to identify and remediate the vulnerabilities present in the system. This usually includes deploying agents that periodically scan the network, the devices and the code repositories to identify misconfigurations.

5. Penetration Testing: a set of activities aimed at testing how the defenses of the company would behave in a realistic attack scenario. A penetration test may be performed by individuals

external to the company but also by internal ones, to simulate an insider threat. Penetration testing activities can be carried out against the company's software, testing the presence of security vulnerabilities, but also against its employees, with techniques like phishing and social engineering.

6. Awareness Training: the set of practices a company can undertake to make its employees -or users- more aware of the threats they may be subject to. This can include formative seminars about detecting and reporting social engineering attempts as well as secure coding courses.

7. Network Security: a broad category that covers the set of all the measures aimed at protecting the company at the network level. It comprises, for example, firewalls, IDS -Intrusion Detection System- and IPS -Intrusion Prevention System- technologies.

8. Supply Chain Risk Management: the set of activities aimed at identifying all the components of the supply chain of the company, e.g., software vendors that provide dependencies, assessing the risk they would pose to the company if compromised, adopt measures to mitigate this risk and monitor each of the identified suppliers for security incidents.

9. Cloud Security: the set of measures aimed at securing the portion of the company's resources that operate in a cloud environment.

10. Threat Intelligence: a set of activities aimed at identifying if any sensitive information about the company has been leaked online. This includes, mainly, access credentials and sensitive files. The objective is to intercept these information before an attacker would to apply proper reactive measures. Another use of threat intelligence is trying to understand if some threat actor is organizing a targeted attack to the company.

11. Data Protection: the set of measures aimed at protecting the data of the business -and its users- from unauthorized access and data loss.

12. Disaster Recovery: the set of measures aimed at restoring business operations after significant event that has damaged the company -e.g., a fire or a natural disaster- [36].

13. Incident Response: the set of measures that are in place to formalize how a security incident is handled. This includes the process of containing and eradicating the threat as well as alerting -if applicable- the board of directors and competent authorities.

14. Compliance: the set of operations which verify the company's compliance with applicable regulations and / or any standard that the company decided to adopt.

Each of the aforementioned areas is a world of its own, with a trove of guidelines, standards, regulations and academic research about it.

Furthermore, cybersecurity is a highly dynamic field, where new tools and techniques are introduced at a very fast pace. This has as its direct consequence that a company and its staff cannot simply implement a good cybersecurity policy and then be done with it, but needs to constantly keep up to date with the latest developments and review their posture to keep it relevant and effective [35].

2.2.2 Lack of personnel and skills

When it comes to cybersecurity, the industry appears to have a shortage of both personnel and skills [26]. Existing studies identify as possible reasons the highly dynamic nature of the field [35], the fast evolution under which it has undergone in the last decades and the many aspects that make up cybersecurity which can make it challenging for new practitioners to form a comprehensive mental model of the cybersecurity landscape [31], [70].

2.2.3 Budget constraints

Cybersecurity is expensive, and the allocation of resources for improving it can at times be hard to justify since it does not provide any immediate benefit to the company, but rather only the hope that some risks have been mitigated [3].

This can be especially true for young or small companies, which tend to operate with limited budgets. Yet, their small size does not necessarily deter the attackers.

One study found how attacks aimed to steal trade secrets are most common among "younger firms, firms with fewer employees, and firms operating in less concentrated industries" [20]. Part of the reason is thought to be relatable to the fact that they tend to have a less mature cybersecurity program in place. According to a recent report, the victims of ransomware attacks in Europe in the second quarter of 2023 were for the 87% enterprises with no more than 50 employees [65].

2.2.4 Asymmetry between attack and defense

Another challenge of securing a company is represented by the asymmetry between the attackers and the defenders. Whilst the cybersecurity team of a company needs to defend all its attack surface, an attacker only needs a single vulnerability to initiate a compromise [32].

Each defense needs to be set up in a careful way, since any misconfiguration may hinder its effectiveness. As the authors of an article on cybersecurity skill shortage said, "Unlike some other aspects of IT, security simply will not work if done badly" [31].

2.3 Enter frameworks

Over the years, multiple institutions have released their own cybersecurity frameworks. The objective of each one of them is to provide its readers with an overview of the requirements that a company should satisfy in order to improve its cybersecurity posture. They also provide a taxonomy of the cybersecurity landscape, establish a common terminology that the companies can use to communicate with their stakeholders and provide some benchmark indicators that can be used as a reference when comparing the posture of multiple companies with one-another.

Some of these frameworks aim to apply to a broad audience, like the NIST's *Framework for Improving Critical Infrastructure Cybersecurity (NIST CSF), version 1.1* [49] and the *CIS Critical Security Controls, version 8* [12]. Others, instead, focus on specific industries, like the *PCI Data Security Standard (PCI DSS), version 4.0*, which caters for companies that process data related to payment cards [56] and the *Cybersecurity Capability Maturity Model (C2M2), version 2.1* for American companies operating in the energy sector [51].

Some of the frameworks are developed starting from existing ones, in a continuous process of enrichment and refinement of the best practices to suggest. One example of this is the *Framework Nazionale per la Cyber Security e la Data Protection, versione 2.0*, which builds upon the NIST CSF, adding to it new controls specific to the data protection -especially in light of the GDPR- and covering the process of deciding which parts of the framework are relevant for a given company through the use of profiles more extensively [2].

Syafrizal, Selamat, and Zakaria provide a precious review of numerous cybersecurity frameworks and standards, cataloguing them, comparing the aspects of cybersecurity they each cover and the context in which they were written [66]. In this section, we present in detail two of the frameworks we encountered most often until now: the NIST CSF and the CIS Critical Security Controls.

2.3.1 NIST Cybersecurity Framework (CSF)

The NIST's *Framework for Improving Critical Infrastructure Cybersecurity (NIST CSF), version 1.1* [49] was first introduced in 2014 and then updated to version 1.1 in 2018. At the moment of writing this work, the draft for version 2.0 of the framework is available for public comment.

2.3.1.1 Functions

The framework organizes the cybersecurity measures a company can take according to five *functions* -also called *outcomes*- [49, §2.1]. These can be seen as representing the five macroscopic actions a company needs to implement to provide protection to their assets. These actions should not be seen as a linear sequence which terminates after the fifth of them, but rather as a circular one, which through multiple iterations allows the company to better prepare for and cope with cyber attacks.

Identify This is, conceptually, the first step and its aim is to render the company conscious of all the assets present in it. This includes -for example- the machines present in the corporate's network, the network equipment, the accounts of the users, and the data.

Protect After having inventoried what a company has, the second step is to harden its security by putting in place preventive measures that shield the assets from attacks.

Detect The next phase is to monitor the assets of the company and the activity of its users to detect when something unusual is happening.

Respond If an attack is detected, proper measures need to be put in place -preferably, before the attack starts- to eradicate the threat.

Recover Finally, the company needs to remediate any vulnerabilities that may have been discovered in the wake of the attack, assess and repair the damages that have arisen from it, and -most likely- plan for further improvements to their cybersecurity posture.

The order in which the functions are defined may be seen as suggesting that each function builds on top of the previous ones. In the case of protect, for example, it is arguable that it "needs" the previous action, identify, since it is impossible to protect your attack surface reliably when you don't know which attack surface you have in the first place [58].

2.3.1.2 Categories

Each of the five functions is divided in multiple *categories* [49, §2.1]. These represent the declination of an action in the multiple facets that it conceptually comprises.

As an example, the first function -identify- has six categories: Asset Management, Business Environment, Governance, Risk Assessment, Risk Management Strategy, and Supply Chain Risk Management. For each of them, the framework includes a brief description.

2.3.1.3 Subcategories

Each category is divided in multiple *subcategories*. These represent specific cybersecurity outcomes that the company intends to achieve [49, §2.1]. For each of the subcategories, the NIST provides, within the framework:

- an identifier, to facilitate the operation of referencing a particular subcategory,
- a brief description of what the objective should be, and
- a list of informative references containing precise pointers to reputable standards and guidelines which are deemed to be related to the subcategory.

2.3.1.4 Implementation tiers

The framework defines the concept of *implementation tiers*. An implementation tier represents how extensively a subcategory should be incorporated in the business processes [49, §2.2].

The framework defines four implementation tiers called, respectively, partial, risk informed, repeatable, and adaptive. The tiers can be said to represents the stage at which an organization is in in their cybersecurity journey.

The partial tier is characterized by an approach to cybersecurity which has not been defined with company policies, but rather is applied on a case-by-case base, possibly in a reactive fashion. On the other end of the scale, the adaptive tier represents the companies in which the risk management processes are formalized, fully integrated and keep evolving according to the threat landscape and the most recent best-practices.

Each organization has to decide which of these tiers is the most appropriate for them to strive for based on, amongst others, their resources, applicable regulations, business needs and their risk appetite.

2.3.1.5 Profiles

Another concept defined by the framework is the one of *profiles* [49, §2.3]. A profile can be viewed, conceptually, as the result of a filtering operation that a company should perform on the requirements set forth by the framework to identify the ones that apply to them and the extent to which they do so.

Let us consider, for example, one of the profiles suggested by the NIST itself for the manufacturing industry [63]. There, it is possible to see how an identification of the possible objectives of the company can help in prioritizing the subcategories that need to be addressed most urgently, removing the ones that offer little to no contribution to the desired outcomes.

To this end, we highlight how one of the objectives of the *Framework Nazionale per la Cyber Security e la Data Protection, versione 2.0* is to provide a more structured approach to the creation of a profile through the use of contextualization prototypes [2, §2.2].

2.3.2 CIS Critical Security Controls v8

The CIS Critical Security Controls is an ongoing effort promoted by the Center for Internet Security (CIS) aiming to promote the sharing of information and knowledge in the cybersecurity industry and the creation of effective guidelines for defense [12]. It wants to provide, quoting from the introduction, a "prescriptive, prioritized, highly focused set of actions" that companies can adopt ro enhance their resilience to cybersecurity threats. Their latest edition, *CIS Critical Security Controls, version 8*, was published in 2021.

2.3.2.1 Controls

The *CIS Critical Security Controls, version 8* divides the actions a company can undertake in 18 categories called *controls*. These may be seen as akin to the concept of categories in the NIST CSF, providing a general categorization of the facets that make up the cybersecurity program of a company.

Each of the controls of the CIS framework is presented alongside a list of references to related material from other frameworks and standards, including ones from the NIST.

2.3.2.2 Safeguards

Each control has a number of safeguards. These are intended as actions that a company can implement to concretely strengthen its cybersecurity posture.

Every safeguard has an identifier, a title and a description. It is also associated with the type of asset that it protects -e.g., users, data, devices- and the *security function* of the NIST CSF to which it conceptually belongs.

2.3.2.3 Implementation groups

The framework provides an ulterior organization of the safeguards according to *implementation groups*. It defines three implementation groups: IG1, IG2 and IG3.

They can be seen as a suggestion on the safeguards to implement, with IG1 representing basilar practices of good cybersecurity and IG2 and IG3 providing incremental improvements and assurance over the posture of the company. All the safeguards included in IG1 are also included in IG2, and the ones included in IG2 are also included in IG3, which comprises all the safeguards detailed in the framework.

2.4 Challenges

The frameworks represent a valuable tool for the companies that choose to adopt them. Their enumeration of the areas in which to concentrate the efforts and suggestions for the aspects to focus on

considerably helps the cybersecurity professional of the company in making sure they did not overlook any aspect.

Yet, there are still other challenges that a business may face when adopting them. In this section, we overview two of them: the need to contextualize the frameworks and the implementation of the measures.

2.4.1 Contextualization and prioritization

The authors of popular cybersecurity frameworks recognize how securing a company is a process that needs to be tailored to each reality [49, §3], [12]. Different businesses have different needs and this can hinder the efficacy of some of the suggestions advanced in a framework, as their application may lead to improvements that are insufficient to justify the cost of their adoption.

A wood working company, for example, may arguably obtain a very negligible benefit from instructing their developers in secure coding practices if the only service that they manage is the website of the company, containing only already-public advertisement material. Perhaps, it would be more advisable to first establish policies for access management, backup and data recovery for the internal data of the company to mitigate the risk of data exfiltration or loss.

The NIST CSF proposes concept of profiles to deal with this necessity and provides some examples of them in additional publications[4].

The CIS Critical Security Controls introduces a similar concept with the notion of implementation groups, essentially providing the users with a macroscopic prioritization of how they may implement the safeguards they propose.

Despite these efforts, however, companies are still required to identify the safeguards that they do not deem applicable to their reality -especially given the conspicuous number of safeguards included in frameworks like the NIST CSF- and define the priority with which they want to implement each of them. This process appears as a necessary yet time-consuming step.

2.4.2 Implementation

After having decided which safeguards a company wants to adopt, these have to be implemented in practice.

At this point, we noticed how -especially in the NIST CSF- the formulation given by the framework of a safeguard may turn out to be unclear or too vague to be actionable. For example, concerning the subcategory "RS.AN-2" of the NIST CSF, the framework states that "The impact of the incident is understood". Arguably, this phrase can lend itself multiple interpretations. It is not specified, for example, which kind of impact we are referring to. Is it an impact in terms of financial loss or does it include the reputation loss? Does it consider only the damages sustained during the handling of the incident or also in its aftermath?

In the case of the NIST CSF, the fact that the guidelines appear, at times, to be to general seems to be a design decision. Quoting from one of the official pages related to the framework[5]: "the Framework is outcome driven and does not mandate how an organization must achieve those outcomes". The NIST CSF, therefore, has as its main objective the one of providing its adopters with a defined target, and it will be their duty to understand how ot achieve it in the context of their business. As they say in the same page, this "enables scalability", since it decouples the goal from the implementation details.

2.4.3 Mapping to other frameworks

Cybersecurity frameworks evolve over time: new versions are released, legal requirements change and entirely novel frameworks get introduced. When a company decides to adopt a new framework, however, having a way of mapping the compliance status that the company had in the previous framework with resect to the new one would be a non-negligible convenience.

This mapping may be done to avoid spending effort in satisfying guidelines that have already been implemented, but also in the context of auditing if there was the need to verify the compliance of the company with respect to a framework different from the one they generally refer to.

[4] https://www.nist.gov/cyberframework/examples-framework-profiles

[5] https://www.nist.gov/cyberframework/online-learning/uses-and-benefits-framework

We highlight how the CIS provides[6] mapping between the *CIS Critical Security Controls, version 8* and numerous other frameworks, including the NIST CSF.

The NIST CSF also includes references in each one of its subcategories to other related resources.

2.5 Formalization

This section provides an analysis and a characterization of the core problem we illustrated in Section 2.4 by mapping it to a known computational problem. First, we provide a mapping to the set cover problem in Section 2.5.1. Then, we show how it is possible -through more attentive considerations on the nature of the input data- to express it as a variant of the set cover problem which better represents the identified task in Section 2.5.2. For both, we define the terminology used to describe the problem, the computational problem, the mapping between the computational problem and our problem, and the class of complexity for solving said problems. Then, we provide a more comprehensive modeling of the problem by introducing ulterior variables in Section 2.5.3 and Section 2.5.4 and show how each of them models the previous version of the problem whilst taking into account other variables.

Defining the terminology is necessary to ensure that -throughout this thesis- each fundamental concept is associated to a precise semantic. The same term, otherwise, may carry different meanings depending on the framework in the context of which it is being considered. An example is the word "control", which in this work is used to indicate what in the the NIST CSF is referred to as subcategory referring to something that a company can do to improve a specific aspect of its cybersecurity posture -e.g., 'RS.CO-2': Incidents are reported consistent with established criteria-, whilst in the CIS Critical Security Controls a "control" is more akin to what in the NIST would be called a "category" of controls -i.e., a group of cohesive controls that share a common area of interest- [49, §2.1, Table 3].

By defining an unambiguous semantic for the most important terms in these sections, we ensure that it is possible to discuss about them unequivocally throughout the rest of this work.

2.5.1 Unweighted

In this section, we introduce the terminology we use in describing the problem. Then, we represent it as a computational problem and compute the size of its search space. Finally, we show how it is possible to view the problem as an instance of the set cover problem -in Appendix B, it is possible to see an alternative representation of the problem as an instance of a different, but equivalent, computational problem: the hitting set problem-.

2.5.1.1 Terminology

Control We refer to a control as a specific objective that a company should strive for in order to improve it cyber security. Considering the NIST CSF, we can identify in its subcategories the controls that the framework suggests -e.g., 'RS.CO-2: Incidents are reported consistent with established criteria', or 'DE.CM-5: Unauthorized mobile code is detected'-. If we consider the *CIS Critical Security Controls, version 8*, the concept of controls maps to the one of safeguards.

Measure We define measure using it with its connotation of indicating an action. In our case, it is something a company can concretely do to implement a control -or part of it- in their reality. If the control may represent a general or abstract idea, the measure is a specific and tangible step that can be realized in practice. Building on the previous example of 'RS.CO-2', a measure relative to that control may be to produce or adopt an incident response plan, formalizing the policies and the processes that need to be in place for when an incident strikes.

Satisfying Not all the measures are related to all the controls. Continuing with our example from the previous definitions, a measure about creating an incident response plan is surely useful for complying with the control 'RS.CO-2', but would yield no benefit for a control like NIST CSF's 'DE.CM-1: The network is monitored to detect potential cybersecurity events'.

[6] `https://www.cisecurity.org/controls/v8`

Therefore, we say that a measure satisfies a control if and only if its implementation contributes in some way to achieving the objective specified by said control.

2.5.1.2 Formalization of the core kampas problem

Let:

1. C be a finite set of controls,
2. M be a finite set of measures,
3. $\models \subseteq M \times C$ be the satisfaction relation. $(m, c) \in \models$, alternatively written as $m \models c$, indicates that the measure m satisfies the control c.

Let $[m]_\models$ be the set of controls in C satisfied by m.

$$[m]_\models := \{c \in C \mid m \models c\}$$

Let us assume that, for each control c in C, there exists at least one measure m in M such as $m \models c$.

$$(\exists\ m \in M \mid m \models c)\ \forall\ c \in C$$

Problem statement Given C, M and $\models$, the *core kampas problem* consists in finding $\mathcal{M} \subseteq M$ such as that $\bigcup_{m\ \in\ \mathcal{M}} [m]_\models = C$ and $\nexists\ \mathcal{M}' \subset \mathcal{M}$ such as that $\bigcup_{m\ \in\ \mathcal{M}'} [m]_\models = C$.

As an example, let us consider $C = \{1, 2, 3, 4\}$ and $M = \{a, b, c, d\}$. Each control may be associated to any of the measures. This can be represented graphically by considering the set of controls and the set of measures as the two parts of a bipartite graph in which every edge connects a control to a measure, indicating that the measure satisfies the control. This representation is shown in Figure 2.1, where the edges are also colored to visually group them by measure.

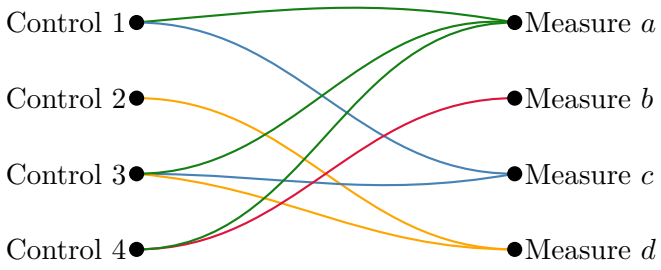


Figure 2.1: Input represented as a bipartite graph

2.5.1.3 Set cover problem

The set cover problem has multiple equivalent formulations. We report here the one provided by Feige[7] in [23, §1]. This definition refers to the optimization version of the problem.

> Let U be a set of n points and $S = \{S_1, S_2, \ldots, S_m\}$ a collection of subsets of U. Set cover is the problem of selecting as few as possible subsets from S such that every point in U is contained in at least one of the selected subsets.

[7] We changed the symbols to make them consistent with the definition of the weighted set cover problem

The set cover problem has two variants, concerned respectively with decision and optimization. As anticipated, the decision variant is one of the Karp's 21 problems [39]. Its objective is to find a set cover with size not greater than a given integer k. Thanks to the work of Karp, we know that the set cover problem in this variant belongs to the class of NP-complete problems [39]. The optimization version, instead, aims to find the set cover of minimum size and is NP-hard [41, §16.1].

2.5.1.4 Mapping with the core kampas problem

The mapping between our problem and the set cover problem is direct. In our case, the universe U can be seen as coinciding with the set of controls C ($U = C$) and the set S can be obtained from the set of measures M by converting each of its elements, m, to the set of controls that m satisfies ($S = \{[m]_{\models} \mid m \in M\}$). Applying the definition of the set cover problem, it is possible to see how the optimal set cover, be it F^*, coincides with the solution $\mathcal{M}$ of the core kampas problem, as $\bigcup_{m \in F^*} [m]_{\models} = C$ is satisfied since $C = U$ and F^* is a set cover over S and the property of it being minimal ($\nexists\, \mathcal{F}^{*\prime} \subset \mathcal{F}^* \mid \bigcup_{m \in \mathcal{F}^{*\prime}} [m]_{\models} = C$) is guaranteed by the fact that F^* is the optimal set cover.

Using the input and color scheme from the example of Section 2.5.1.2,we can see the set cover problem represented graphically in Figure 2.2. In the case of this example, the possible set covers are $\{a, d\}$, $\{a, b, d\}$, $\{a, c, d\}$, $\{b, c, d\}$, and $\{a, b, c, d\}$. $\{a, d\}$ is the cover with the fewest subsets: 2.

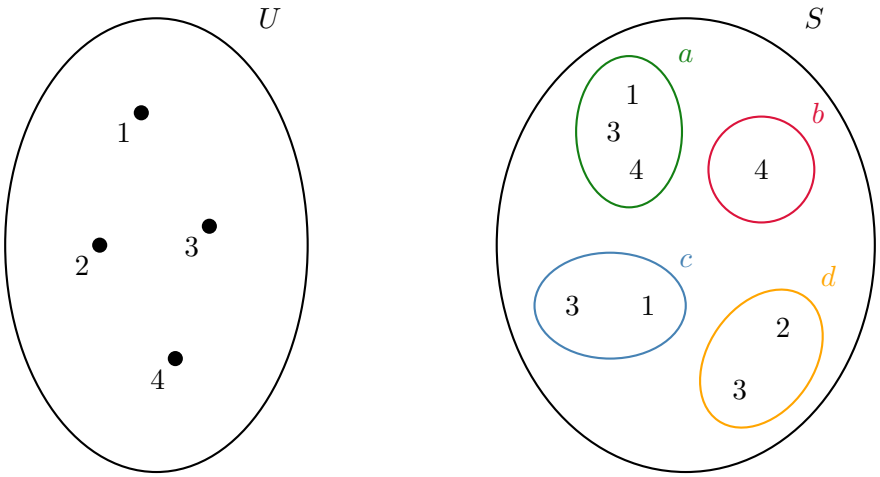


Figure 2.2: Example of the set cover problem

2.5.2 Weighted

In Section 2.5.1, we show how it is possible to describe the problem of managing cybersecurity controls as an instance of the set cover problem.

We may, however, want to reconsider how we characterize the measures -i.e., the sets a, b, c and d in Figure 2.2- by accounting for the fact that not all of them require the same amount of effort to implement. When it comes to practice, one measure may turn out to be easier to implement than another and, therefore, preferable to the latter if they contribute equally to solving the problem. Hence, we proceed in inspecting the weighted variant of the set cover problem, which does accommodate for this additional refinement of the problem modeling in which the difficulty of implementing a measure is also accounted for through the introduction of an additional concept: the cost of a measure.

2.5.2.1 Additional terminology

Progress of a measure We use the term progress to represent how much a measure has been completed. In this thesis, we consider a progress as an integer in the range [0, 100], where zero indicates that the implementation of the measure has not started yet and 100 that its implementation is complete.

Effort of a measure A measure is also associated to an effort. This represents an estimate of the amount of work required to implement the measure in the company. A company may choose a unit of its liking to associate with the effort, like the estimated financial cost or the manhours of work necessary to fully implement the measure.

Cost of a Measure The cost of a measure represents the cost required to complete the implementation of said measure. It is given by the effort of the measure multiplied by the remaining progress. This concept can be quantified as in Equation (2.1), where cost_m represents the cost of a measure m, progress_m its associated progress, and effort_m its effort.

$$\text{cost}_m := (100 - \text{progress}_m) \times \text{effort}_m \tag{2.1}$$

2.5.2.2 Decorated kampas problem

In this section, we formalize the decorated version of the core kampas problem of Section 2.5.1.2. In addition to the definitions of Section 2.5.1.2, let:

1. Π be a set of progress values,
2. H be a set of effort values

Let $M(\Pi, H) \in M \times \Pi \times H$ be the set of decorated measures whose elements are written as $m(\pi, \eta)$ instead of (m, π, η).

Let $\models^d \subseteq M(\Pi, H) \times C$ be the decorated satisfaction relation whose elements are written as $m(\pi, \eta) \models^d c$ instead of $(m(\pi, \eta), c) \in \models^d$ for simplicity.

Let $[m(\pi, \eta)]_{\models^d}$ indicate the set of controls in C satisfied by a decorated measure $m(\pi, \eta)$ according to the decorated satisfaction relation $\models^d$.

$$[m(\pi, \eta)]_{\models^d} := \{c \in C \mid m(\pi, \eta) \models^d c\} \tag{2.2}$$

Let us define, coherently with Equation (2.1), the cost function $\gamma : M(\Pi, H) \to Q^+$ of a decorated measure $m(\pi, \eta)$:

$$\gamma_{m(\pi,\eta)} := (100 - \pi) \times \eta \tag{2.3}$$

Problem statement Given C, $M(\Pi, H)$ and $\models^d$, the *decorated kampas problem* consists in solving the optimization problem expressed in Equation (2.4): finding the set of decorated measure $\mathcal{M}_d$ such as that the sum of the costs of the measures in $\mathcal{M}_d$ is minimal and that the union of the controls satisfied by all the measures present in $\mathcal{M}_d$ is equal to the set of all controls C.

$$\min_{\mathcal{M}_d} \sum_{m(\pi,\eta) \in \mathcal{M}_d} \gamma_{m(\pi,\eta)} \tag{2.4}$$

Subject to:

$$\bigcup_{m(\pi,\eta) \in \mathcal{M}_d} [m(\pi, \eta)]_{\models^d} = C \tag{2.5}$$

2.5.2.3 Weighted set cover problem

The weighted set cover problem is similar to the set cover problem, but we are also given a set of costs -also known as weights- associated to each subset of S and want to find the cover that minimizes the total cost, which is not necessarily the cover with the fewest subsets. Across the literature, especially

in older works, we saw this problem being referred to as "set cover problem" -[13], [68]-, while other authors differentiate it from the set cover problem by referring to it as "weighted set cover problem" -[15], [23]-. We report here the definition provided by Vazirani in [68, §2]:

> Given a universe U of n elements, a collection of subsets of U, $S = \{S_1, ..., S_m\}$, and a cost function $c : S \to Q^+$, find a minimum cost subcollection of S that covers all elements of U.

Vazirani details a greedy algorithm to solve this problem in [68, §2.1]. They prove that the algorithm approximates the optimal solution by a factor of $H_n := 1 + \frac{1}{2} + \cdots + \frac{1}{n}$, the n^{th} harmonic number. This algorithm runs with a time complexity of $O((\sum_{i=1}^{m} |S_i|) \times \log m)$, with $m := |S|$ [15].

Cygan, Kowalik, and Wykurz propose another algorithm that provides a better approximation of the optimal solution at the cost of an increased computational time, which is exponential with respect to the size of the input n [15].

It is also possible to express the weighted set cover problem with an Integer Linear Programming formulation. It consists in minimizing $\sum_{s \in S} w_s \times x_s$, where w_s represents the weight of the subset s, and $x_s \in \{0, 1\}$ represents our choice to include s in the solution when its value is 1, and the choice to exclude it when its value is 0. The minimization is subject to the following condition: $\forall e \in U$, $\sum_{s:e \in S} x_s \geq 1$. Therefore, for every element e in U, the sum of the x_s of all the sets s that include e must be at least one -enforcing the requirement of covering all the elements in U with at least one subset s-.

The decision variant of the weighted set cover problem is NP-complete, while its optimization variant is NP-Hard [41, §16.1].

2.5.2.4 Mapping with the decorated kampas problem

To map the decorated kampas problem to the weighted set cover problem, it is possible to perform the steps detailed hereafter. Consider the set of controls C as the universe U. Let S be the set of subsets obtained from $M(\Pi, H)$ by mapping each element to the set of controls it satisfies ($S = \{[m(\pi, \eta)]_{\models^d} \mid m(\pi, \eta) \in M(\Pi, H)\}$). Let the cost function c be given by $c(s) = \gamma_{m(\pi,\eta)} \mid m(\pi, \eta) \in M(\Pi, H), [m(\pi, \eta)]_{\models^d} = s$. The optimal weighted set cover of this input, be it F^*, can be intuitively seen as also the solution to the decorated kampas set problem, as F^* (i) has the lowest cost -by virtue of it being optimal, i.e., the minimum cost subcollection- and this cost is the same as the one we aim to optimize in Equation (2.4), and (ii) is a set cover of S for U, meaning that all the elements of U are covered by at least one element in s, satisfying the condition of Equation (2.5).

2.5.2.5 Deciding which algorithm to apply

The size of the input n -the number of controls- in our program can easily reach the hundred units: the NIST Cybersecurity Framework has 108 controls. Th number of measures, m, also has the potential to become too high to process with algorithms that are exponential in its regard.

Considering this, we decided to prioritize speed of execution over a more precise approximation of the optimal solution, selecting the greedy weighted set cover algorithm presented by Vazirani for our software. We use as the weighting function the cost associated to a measure, as defined in Equation (2.1). The implementation is detailed in Section 4.1.3.

2.5.3 With coverage

In this section, we introduce a refinement with respect to how we model the controls and the measures in our program. It is based on the observation that, when a measure satisfies one or multiple security controls, it may not necessarily contribute to each one in equal proportion -a concrete example of this is present in Section 2.5.3.1-. To account for this, we introduce the new concepts of coverage between a control and a measure and of completion of a control.

We show that the decorated kampas problem represents a special instance of this problem when each measure contributes in full to each control it satisfies.

2.5.3.1 Additional terminology

Coverage Each measure has as its end goal the one of satisfying one or more controls. Yet, a measure does not necessarily satisfy all its associated controls in an equal manner. As an example, a measure concerned with creating a backup copy of all the data the business deems critical may be considered as sufficient for satisfying a control requiring to have a backup of the important data the business needs to operate. Still, it should be considered only a partial step in the implementation of a control which -for example- demands the creation of a disaster recovery plan -since a disaster recovery plan aims to ultimately restore the business operations, not just the data [67]-.

To account for this disparity in how comprehensively a measure can satisfy a given control, we introduce the concept of coverage. Every time we associate a measure to a control, saying that such measure satisfies said control, we shall also include a number indicating the coverage or, in other words, the degree to which the measure, if implemented, would contribute to a complete implementation of the control. We express the coverage as a number in the interval $]0,\ 100]$. The value zero is excluded since a coverage of zero would mean that the measure and the control are actually not related.

Completion of a Control We define a control's completion as a number representing how complete the control is, currently with respect to the set of measures. The completion of a control is computed by summing the progress of each measure that satisfies said control. Since when a measure is associated to a control this association has a coverage, defined above, the progress of the measure is multiplied by the coverage, be it $\text{progress}_{(m,\ c)}$, normalized between 0 and 1. This results in a completion $\mathcal{C}_c$ for a control c and a set of measures M represented by Equation (2.6). As a progress of 100 indicates the completion of a measure, a completion of 100 -or greater- indicates the completion of a control as well.

$$(\mathit{completion})_{(c,M_c)} := \sum_{m\ \in\ M_c\ \mid\ m \text{ satisfies } c} \text{progress}_m \times \frac{\text{coverage}_{(m,c)} - 0}{100 - 0} \tag{2.6}$$

2.5.3.2 Kampas problem with coverage

In addition to the definitions of Section 2.5.2.2, let Ξ be a set of natural numbers in the range $]0, 100]$, representing the coverage values.

Let us define $\models^d_\chi \subseteq M(\Pi, H) \times C$, for $\chi \in \Xi$, as the family of decorated satisfaction relations indexed over Ξ, whose elements are written as $m(\pi, \eta) \models^d_\chi c$ or simply as $m \models_\chi c$ instead of $(m, \chi, c) \in \models^d$.

Let us denote with $\chi(m(\pi, \eta), c)$ the value $\chi \in \Xi$ such that $m(\pi, \eta) \models^d_\chi c$.

Let us assume that, for every $m(\pi, \eta) \in M(\Pi, H)$ and $c \in C$, there exists no $\{\chi, \chi'\} \subseteq \Xi$ such as that $\chi \neq \chi'$, $m(\pi, \eta) \models^d_\chi c$ and $m(\pi, \eta) \models^d_{\chi'} c$.

Let $[m(\pi, \eta)]_{\models^d_\bullet}$ indicate the set of controls in C satisfied by a decorated measure $m(\pi, \eta)$ for some coverage in Ξ.

$$[m(\pi, \eta)]_{\models^d_\bullet} := \{c \in C \mid (\exists\ \chi \in \Xi \mid m(\pi, \eta) \models^d_\chi c)\} \tag{2.7}$$

Let us define, propaedeutically, the completion $\mathcal{C}_{(c,M_c)}$ of a control c, for a set of measures M_c that satisfy it, coherently with what detailed in Section 2.5.3.1:

$$\begin{gathered} \mathcal{C}_{(c,M_c)} := \sum_{m \in M_c} \pi \times \frac{\chi(m(\pi, \eta), c)}{100} \\ M_c \subseteq \{m(\pi, \eta) \in M \mid c \in [m(\pi, \eta)]_{\models^d_\bullet}\} \end{gathered} \tag{2.8}$$

Let us define the contribution of a single measure $m(\pi, \eta)$ for the completion of a control c.

$$\delta_{(m(\pi,\eta),\ c)} := \pi \times \frac{\chi(m(\pi,\eta), c)}{100} \tag{2.9}$$

If we change the progress of a measure to be π', we define the variation in contribution with respect to the original progress π as follows:

$$\begin{aligned} \Delta_{(\delta_{(m(\pi,\eta),c)},\pi')} &:= \delta_{(m(\pi',\eta),c)} - \delta_{(m(\pi,\eta),c)} \\ &= (\pi' - \pi) \times \frac{\chi(m(\pi,\eta), c)}{100} \end{aligned} \tag{2.10}$$

Let us define the increase in completion $\Delta_{\mathcal{C}_{m(\pi,\eta)}}$ that a measure $m(\pi, \eta)$ would bring if fully implemented ($\pi' = 100$) as:

$$\begin{aligned} \Delta_{\mathcal{C}_{m(\pi,\eta)}} &:= \sum_{c \in [m(\pi,\eta)]_{\models^d_\bullet}} \Delta_{(\delta_{(m(\pi,\eta),c)},\pi')} \\ &= \sum_{c \in [m(\pi,\eta)]_{\models^d_\bullet}} (100 - \pi) \times \frac{\chi(m(\pi,\eta), c)}{100} \\ &= (100 - \pi) \times \sum_{c \in [m(\pi,\eta)]_{\models^d_\bullet}} \frac{\chi(m(\pi,\eta), c)}{100} \end{aligned} \tag{2.11}$$

Let us define the cost function $\gamma : M(\Pi, H) \to Q^+$ of a decorated measure $m(\pi, \eta)$ as in Equation (2.3):

Problem statement Given C, $M(\Pi, H)$, $\models^d_\chi$ and the cost function $\gamma_{m(\pi,\eta)}$ defined in Equation (2.3), the *kampas problem with coverage* consists in solving the optimization problem expressed in Equation (2.12): finding the set of decorated measures $\mathcal{M}_d$ such that (i) the sum of the costs of the measures contained in it is minimal, and (ii) when all the measures present in $\mathcal{M}_d$ are fully implemented, every control in C has a completion of at least 100.

$$\min_{\mathcal{M}_d} \sum_{m(\pi,\eta)\ \in\ \mathcal{M}_d} \gamma_{m(\pi,\eta)} \tag{2.12}$$

Subject to:

$$\forall c \in C, \left(\sum_{m(\pi,\eta)\ \in\ \mathcal{M}_d\ \mid\ c\ \in\ [m]_{\models^d_\bullet}} 100 \times \frac{\chi(m(\pi,\eta), c)}{100} \right) \geq 100 \tag{2.13}$$

Which is equivalent to:

$$\forall c \in C, \left(\sum_{m(\pi,\eta)\ \in\ \mathcal{M}_d\ \mid\ c\ \in\ [m]_{\models^d_\bullet}} \chi(m(\pi,\eta), c) \right) \geq 100 \tag{2.14}$$

Theorem 2.5.1. *In the special case where all the coverage values are equal to 100* ($\Xi = \{100\}$) *the kampas problem with coverage becomes an instance of the decorated kampas problem.*

Proof. Before proceeding, let us rewrite the condition expressed in Equation (2.5) in an equivalent form:

$$\forall c \in C, \exists\ m(\pi,\eta) \in \mathcal{M}_d \mid c \in [m(\pi,\eta)]_{\models^d} \tag{2.15}$$

Let us consider the special case in which all the coverages are 100%, meaning that all measures contribute equally and in full to each control they satisfy. We show that the problem reduces to an instance of the decorated kampas problem.
If the coverage is 100, Equation (2.14) becomes

$$\forall c \in C, \left(\sum_{m(\pi,\eta) \ \in \ \mathcal{M}_d \ | \ c \ \in \ [m]_{\models^d_\bullet}} 100 \right) \geq 100 \tag{2.16}$$

We can simplify the 100

$$\forall c \in C, \left(\sum_{m(\pi,\eta) \ \in \ \mathcal{M}_d \ | \ c \ \in \ [m]_{\models^d_\bullet}} 1 \right) \geq 1 \tag{2.17}$$

It is to be observed how this condition is verified if and only if the cardinality of the set over which we perform the summation, $m(\pi,\eta) \ \in \ \mathcal{M}_d \mid c \ \in \ [m]_{\models^d_\bullet}$ is greater or equal to one.

$$\forall c \in C, |m(\pi,\eta) \ \in \ \mathcal{M}_d \mid c \ \in \ [m]_{\models^d_\bullet}| \geq 1 \tag{2.18}$$

This is equivalent to the condition in Equation (2.15) -and the function to minimize in Equation (2.4) and Equation (2.12) is the same-. □

2.5.4 Incremental kampas problem

In this section, we introduce a final refinement of the modeling of the problem. This is given by considering that a controls' final completion is given not only by the contribution of all the measures we select for the solution ($\mathcal{M}_d$) when completed, but also from the contribution of all the measures that we do not select, in their current state.

We show that the kampas problem with coverage represents a particular case of this refined problem, where the progress of all the measures we do not select is counted as zero.

This adds a final layer of complexity to our problem, to the point where we were not able to map it to any known computational problem. To solve it, we adopted a modified version of the weighted set cover greedy algorithm.

Problem statement Given C, $M(\Pi, H)$, $\models^d_\chi$ and the cost function $\gamma_{m(\pi,\eta)}$ defined in Equation (2.3), the *incremental kampas problem* consists in solving the optimization problem expressed in Equation (2.19): finding the set of decorated measures $\mathcal{M}_d$ such that (i) the sum of the costs of the measures contained in it is minimal, and (ii) when all the measures present in $\mathcal{M}_d$ are fully implemented, every control in C has a completion of at least 100. When computing the completion, the existing contribution of the controls that have not been included in the solution shall also be accounted for.

$$\min_{\mathcal{M}_d} \sum_{m(\pi,\eta) \ \in \ \mathcal{M}_d} \gamma_{m(\pi,\eta)} \tag{2.19}$$

Subject to:

$$\forall c \in C, \left(\sum_{m(\pi,\eta) \ \in \ M(\Pi,H) \ | \ c \ \in \ [m]_{\models^d_\bullet}} p_{m(\pi,\eta)} \times \frac{\chi(m(\pi,\eta), c)}{100} \right) \geq 100 \tag{2.20}$$

$$p_{m(\pi,\eta)} := \begin{cases} 100 & \text{if } m(\pi,\eta) \in \mathcal{M}_d \\ \pi & otherwise \end{cases} \tag{2.21}$$

Theorem 2.5.2. *In the special case where we consider the progress value, π, of all the measures we do not select in $\mathcal{M}_d$ to be equal to zero -either because it really is zero or we do not account for these*

measures in computing the coverage of a control-, the incremental kampas problem becomes an instance of the kampas problem with coverage.

Proof. First, we observe that Equation (2.19) and Equation (2.12) are identical. Then, since all the progress values for the measures that are not included in $\mathcal{M}_d$ are equal to zero -by hypothesis-, we can rewrite Equation (2.21) as follows:

$$p_{m(\pi,\eta)} = \begin{cases} 100 & \text{if } m(\pi,\eta) \in \mathcal{M}_d \\ 0 & otherwise \end{cases}$$

We can now observe that in the summation of Equation (2.20), all the measures that are not included in $\mathcal{M}_d$ have a $p_{m(\pi,\eta)}$ of zero, meaning that we can exclude them from the summation as they would only add zeros to it. We obtain that

$$\forall c \in C, \left(\sum_{m(\pi,\eta) \in \mathcal{M}_d \mid c \in [m]_{\models^d_\bullet}} p_{m(\pi,\eta)} \times \frac{\chi(m(\pi,\eta),c)}{100} \right) \geq 100 \tag{2.22}$$

We can now observe that, by the second step of this proof, $p_{m(\pi,\eta)}$ is equal to 100 for all the $m(\pi,\eta) \in \mathcal{M}_d$. Thus, we substitute $p_{m(\pi,\eta)}$ with 100.

$$\forall c \in C, \left(\sum_{m(\pi,\eta) \in \mathcal{M}_d \mid c \in [m]_{\models^d_\bullet}} 100 \times \frac{\chi(m(\pi,\eta),c)}{100} \right) \geq 100 \tag{2.23}$$

This is equivalent to Equation (2.13). □

2.5.5 Ranking

With the work of the previous sections, we have a way of knowing -given a set of controls to satisfy and one of measures to implement- which of the measures are worth to implement. Now, we introduce the concept of a ranking to attempt to find not only the set of measures that are worth implementing, but also the order in which it is best to implement them to obtain the best efficacy.

2.5.5.1 Additional terminology

Ranking A ranking indicates a list of measures that the user can adopt as a reference to help prioritizing which measures it is best to implement to satisfy all the controls. Ideally, we want this array to be sorted so that the "best" measures are shown first.

More formally, we may say that a ranking r is a collection of measures represented as an array, where each element of such array belongs to the set of measures M.

$$r := [m_1,\ m_2,\ \ldots,\ m_n], \quad m_i \in M \tag{2.24}$$

Optimal Ranking Let us define R_M as the set of all the possible rankings for our set of measures M. The goal is to find, amongst all the possible rankings in R_M, a ranking r so that r is optimal.

We define such a ranking in terms of being optimal for a set of measures M and a set of controls C. The optimal ranking is the ranking that allows us to satisfy the most controls in C with the smallest overall cost. We will use the cost definition of Section 2.5.2.1.

2.5.5.2 Search space

Before proceeding further, it is worth noting that the search space -also known as feasible region- for this problem corresponds to all the distinct ordered subsets of a set N with n elements. Let us calculate its size.

Let S be an improper subset of N -$S \subseteq N$-. The cardinality of S will be in the range [0-n]. This range is inclusive since we include the empty set and N itself. Considering i in the range $[0 - n]$, we can use the binomial coefficient to know how many sets S of cardinality i can be constructed in N: $\binom{n}{i}$. The total number of distinct subsets is therefore $\sum_{i=0}^{n} \binom{n}{i}$. To give a reference of the scale, it is also known that this number -the number of possible subsets of a set N- is 2^n. Since in our case we care about the order of the elements within these subsets, we need to multiply each S for the number of its permutations, which is $i!$.

$$\sum_{i=0}^{n} \binom{n}{i} \times i!$$

By definition of binomial coefficient this is equivalent to

$$\sum_{i=0}^{n} \frac{n!}{i! \times (n-i)!} \times i!$$

At this point we can simplify $i!$

$$\sum_{i=0}^{n} \frac{n!}{(n-i)!}$$

Now, we observe that $i = n - n + i = n - (n - i)$. Let $k := n - i$, so that $i = n - k$. Let us consider i in the range $[0, n]$. We have a series of equivalences: $i = 0$, $i = 1$, ..., $i = n$. By replacing i with $n - k$ we obtain the sequence of $n - k = 0$, $n - k = 1$, ..., $n - k = n$, which is the sequence going from n to zero. Considering i in the range $[0, n]$ therefore is equivalent to considering k in the range $[n, 0]$. Inside the sum, we replace the i in the denominator with $n - k$ since they are equivalent by the definition of k.

$$\sum_{k=n}^{0} \frac{n!}{(n - (n-k))!}$$

Simplifying the denominator and reversing the order in the sum -since it is commutative-, we get

$$\sum_{k=0}^{n} \frac{n!}{k!}$$

According to the equivalences (35) and (36) in the Wolfram page on binomial sums[8], this is equivalent to

$$\lfloor n! \times e \rfloor$$

It is worth noting that this number represents the absolute worst-case scenario, in which we evaluate all the permutations of all the possible subsets of the input.

If, on the other hand, we were able to first find the subset S that contains the best permutation and then find such permutation, the complexity would be expressed as 2^n to scan all the subsets and find S and $k!$ to find the best permutation given $k := |S|$, resulting in a complexity of $2^n + k!$.

2.5.5.3 Algorithm

At this point we observe that the greedy weighted set cover algorithm already provides the best ranking for the solution it finds. This is not the optimal ranking as the greedy algorithm is not guaranteed to find the best weighted set cover, but does represent the optimal ranking for the identified set of measures.

[8] `https://mathworld.wolfram.com/BinomialSums.html`

3 Solution design

In this chapter, we perform a requirements analysis on a potential solution for the issues identified in the previous chapter in Section 3.1. Then, we show how existing state of the art solutions fail to address the problem effectively in Section 3.1.3. Finally, we bootstrap the design process for the service we present in this thesis in Section 3.3.

3.1 Requirements analysis

In the requirements analysis detailed in this section, we aim to identify three key aspects of a potential solution: (i) who will use the system, which pain points do they have, (ii) what functionalities this solution needs to provide to be effective -also known as functional requirements [61, §4.1.1]- and (iii) the presence of constraints in terms of performance, cost or regulatory requirements -also known as non-functional requirements [61, §4.1.2]- that may cause us to dismiss potential candidate solutions.

3.1.1 Users

Primary user The primary user in our context would be the CISO. They are the principal decision maker of their company when it comes to cybersecurity. Their duty is to constantly monitor and improve the cybersecurity posture of the company. They are also the person responsible for achieving and maintaining compliance with any and all the cybersecurity frameworks that the company chose to adopt and applicable regulations on the matter. While performing their duty, they must respect the budget allocated by the company and cope with the evolution of the services offered by the company, the systems they use to provide said services and the cybersecurity beast practices.

A pain point of this user is definitely represented by the sheer number of aspects they need to keep under control at any given time. If on one hand they can delegate the majority of the tasks we just mentioned to several other figures in the cybersecurity division of the company, on the other they need to stay constantly informed on the status of the company. They need visibility into the cybersecurity status of the company to identify and pursue the decisions that will best contribute to its strengthening.

Secondary users A secondary user can be identified in a generic employee of the cybersecurity division of the company. In general, they are tasked with keeping under control a particular area of the threat surface -e.g., physical access control, vulnerability management, &c.-. This person usually periodically reports to their supervisor on how the company is faring in their area of competence. We may say that this person has a similar pain point with respect to the CISO: they need to manage multiple objectives and either oversee or undertake their concrete implementation inside the company.

Two additional secondary users may be identified in (i) a compliance auditor tasked with verifying if the measures in act in the company comply with a specific standard -or set thereof- and (ii) a member of the executive board of the company to which the CISO periodically reports. Unfortunately, we do not have much knowledge on these roles and their pain points, therefore we only advance the hypothesis that to fulfill their duty at their best, they need the information neatly organized to quickly get to the data they care about in that moment.

3.1.2 Requirements

Having identified the archetypical users of the service, we list some of the functionalities that we think they would need, or like, in a tool capable of assisting them in their task:

- Possibility to manage a growing number of the security controls. This means being able to view the ones that the company is trying to implement -or has already implemented- and check their progress.

- Assistance in deciding which controls it is best to fulfill next.
- Possibility to search or filter the controls to quickly find the one of interest during normal operations or an audit.
- Ability to support multiple users collaborating on the same set of data to ease the effort that otherwise would be on a single person.
- Ability to see the evolution in time of the progress of the company, as a whole and in multiple areas. We think that this functionality may be especially useful to the CISO and the member of the executive board to gauge the trajectory of the company holistically and be able to initiate eventual corrections.
- Ability to generate reports, which may be especially useful for providing them to the compliance auditor or the executive board member.

3.1.3 Constraints

Having identified the functionalities we expected in a solution, we proceed listing some of the constraints that it may need to respect:

- Simple to use: The users should be able to get familiar with the solution quickly, without requiring extensive training. As a benchmark, we may deem this requirement to be satisfied if the user is able to utilize the solution autonomously after having spent no more than an hour consulting dedicated tutorials.
- Low overhead: The solution should be quick to use. As a benchmark, we may verify if an experienced user can complete the following operations in less than one minute each:
  - Add a new security control;
  - See the completion of a control;
  - Obtain a list of controls filtered according to criteria specified by the user;
  - Obtain a global view of the progress of the company across all controls.
- Secure: The solution should not become a weak point in the perimeter of the company
- Cost-effective: The cost associated with running and managing the solution will need to be acceptable by the company.
- Sovereignty over the data: The software will contain classified information about the security objectives of the company and how they plan to achieve them. This information should not be disclosed to third parties. To limit this risk, it is worth giving the company the possibility to host its data in a server under their ownership.

3.2 Existing solutions

Having obtained a clear picture on the requirements that a solution should satisfy in Section 3.1, we now proceed inspecting the existing services and frameworks to determine if some of them can already be employed as a satisfactory solution.

3.2.1 Academic works

3.2.1.1 MSRMP

MSRMP -Multi Stakeholder Risk Minimization Problem- was recently formalized by Mollaeefar and Ranise [48]. Its primary purpose is to help a company choose the right set of security controls to implement in order to minimize the risk for multiple stakeholders.

The complexity of the task is given not only by the number of security controls, that may be implemented, but also from the benefits and the costs that each of them brings: one control may be beneficial in reducing the risk of a possible attack, but the cost of its implementation may render its

choice prohibitive. The work also accounts for multiple stakeholders, each of which has a different perception of the true impact of a control both in terms of benefits and costs. Citing from their work -§4- "Different stakeholders have different criteria that define what they consider risky. Data controllers (e.g., companies) typically choose business impact criteria, such as financial impact or reputation, whereas data subjects (e.g., individuals) evaluate risk based on impact on their personal sphere".

Though in [48] the primary concern is about controls related to data protection, this is not a limiting factor in the applicability of the approach to other contexts.

In their solution, each stakeholder can express their preferences independently of one-another and the algorithm identifies the pareto-optimal solution which satisfies all of them to the largest extent possible.

In the case of our problem, multiple stakeholders may be represent the CISO, the members of the executive board, and the developers who have to implement the measures that get chosen.

The authors did produce an implementation of their proposed approach. It consists of a Java application that takes as input a set of JSON files with the data, computes the optimal solution and outputs a visualization of the data. The source code of the project is available on GitHub[1]. To operate the tool, some technical knowledge is definitely needed and the system lacks of interactive features: the user can only edit the JSON files to define the input, run the program and get its output. It is not possible to add the controls iteratively or get different visualizations of the data. These limitations are perfectly comprehensible given that the program is a proof of concept of the novel methodology detailed in the paper, which is the main contribution of the work.

3.2.1.2 CryptoAC

Berlato *et al.* proposed a methodology to help companies evaluating different Cryptographic Access Controls (CACs) architectures for cloud computing in order to find the one which is most suited for their threat model [7].

Besides formalizing the problem and detailing a methodology to obtain a solution to it, the authors also provide a tool, CryptoAC[2], which automates the computing of the best CAC given as input constraints the trust assumptions and requirements of the architecture. The output of the program is an ordered list of the architectures that best satisfy the expressed needs, with the best one shown first.

Despite not being focused on the topic of selecting cybersecurity measures or controls, this work does provide an interesting reference for how a problem similar in complexity to the one we are analyzing in this work may be managed. It is worth noting that their tool is open source[3].

3.2.2 Commercial offerings

3.2.2.1 Spreadsheets

Spreadsheets have been a constant staple for computer users. The oldest examples we could find of their introduction dates back to 1978 with Visicalc [57], while the first version of Microsoft Excel was released in 1985[4].

They enable the users to store significant amounts of data in a structured way and provide the functionality to perform a plethora of operations on it -arithmetics, statistics, sorting, &c.-. Some software -like Microsoft Excel- have gone so far as to turn their spreadsheet's syntax into Turing complete languages [4].

A clear advantage of spreadsheets when compared to the other solutions surveyed in this chapter is that many users are already familiar with them, at least at a basic level. However, this software was not conceived with this specific aim -of managing and ranking cybersecurity controls and measures- in mind and the level of skills necessary to adapt it to manage hundreds of security controls arguably exceeds the thresholds we set for the ease of use and low overheads in Section 3.1.3.

[1] https://github.com/stfbk/MSRMP

[2] https://st.fbk.eu/complementary/TOPS2020_2

[3] https://github.com/stfbk/CryptoAC

[4] https://www.versionmuseum.com/history-of/microsoft-excel

3.2.2.2 CIS CSAT

The CIS -the same entity which publishes the *CIS Critical Security Controls* [12]- also provides a tool to help companies self-assess their level of maturity with respect to the CIS controls: the CIS Controls Self Assessment Tool (CSAT)[5].

There are two versions of the tool: hosted and pro. The pro version is available exclusively with an active subscription to the "CIS SecureSuite", whilst the hosted version is managed by the CIS and is free to use. Both of them are web applications, with the pro version offering additional features.

The free version offers the possibility to register multiple users under the same organization and mark some of them as admins. The administrators of the company can start a new "assessment". When they do so, they are required to specify which version of the CIS Critical Security Controls should be considered -either 7 or 8-.

Performing an assessment in this tool consists in iterating over all the CIS controls of the framework that was chosen as reference and verifying the status of each one. All the controls are listed in a single page in a tabular format -see Figure 3.1-. Clicking on a control brings the user to a page dedicated to such control, in which it is possible to assign tags to the control, though some predefined ones are immediately specified to help map the CIS controls to other well-known frameworks -note the tags in Figure A.1 in Appendix A-. For each of these controls, the user can mark it as either applicable or not applicable. If a control is applicable, then it is necessary to fill the corresponding columns indicating if a policy for the control was written and approved or if the control was implemented. A distinction is made between a control that was 'implemented', 'automated' and 'reported'. We were not able to find the official documentation on this terminology, but our guess is that the first term indicates an implementation that is effective only momentarily, the second one that will continue being effective on its own and the third indicates that besides the control implementation, an automated process is also in place to report on its status.

Nonetheless, the users specify the progress of each of the controls on a scale from one to four, going from not implemented to fully implemented. Since there are over one hundred controls, it is possible to assign them to various users in the organization, giving to each one a deadline to report on the status of the control. This can be seen in Figure A.1 in Appendix A. Each user can view their deadlines on a dedicated calendar view. After a user has verified the status of a control and updated it in the tool accordingly, they mark said control as 'completed', with the semantic that the assessment of that control is completed, not necessarily the control itself.

Throughout the process of assessment, it is possible to oversee the progress and the overall results from a dedicated dashboard page -visible in Figure A.2in Appendix A-.

The CIS CSAT also allows to generate a report of the assessment in PDF format, as an excel spreadsheet, and as a simple PowerPoint presentation.

Once an assessment is marked as complete, it is possible to create a new assessment, either reusing the data of the previous one as a baseline, importing the data from an excel file or starting from scratch.

Finally, the dashboard shows the compliance score of the company during the last ten months along with the industry average score for the same period. The industry average score is calculated thanks to the fact that when an organization signs up, they need to specify in which sector they operate and, by using the hosted version, they agree that the data on their controls completion can be collected and aggregated for these statistics. This sharing is optional for the users of the pro version.

Overall, the CIS CSAT appears to be a solid product, backed by a reputable company and with an assortment of features tailored for the enterprise word. The necessity of a pro license to have the possibility to keep the assessment data hosted on a system controlled by the company may be an acceptable cost for many businesses. As for the drawbacks, we highlight how this tool (i) does not offer the possibility to define the measures to undertake in order to implement the controls, (ii) does not make it possible to define custom controls either, (iii) does not give any assistance in prioritizing which controls to implement -besides filtering them by implementation group-, and (iv) forces the users to measure the progress with a fixed scale (policy defined, control implemented, control automated and control reported), which may be too coarse for some realities.

[5] `https://www.cisecurity.org/controls/cis-controls-self-assessment-tool-cis-csat`

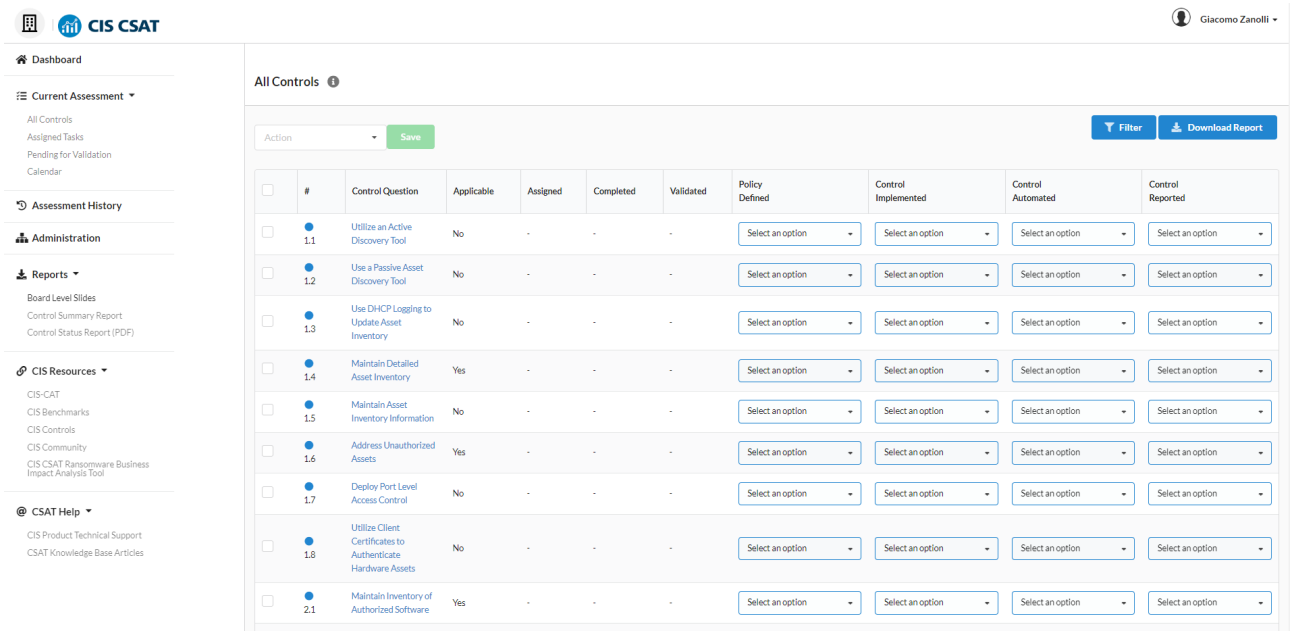


Figure 3.1: CIS CSAT web app. Controls list page

3.2.2.3 Other products

In our research, we found several companies offering products which aim to solve very similar -if not the same- problem that we are considering in this work: AuditBoard (`auditboard.com`), ByteChek (`bytechek.com`), ControlMap (`controlmap.io`), Drata (`drata.com`), Hyperproof (`hyperproof.io`), Scrut (`scrut.io`), and Scytale (`scytale.ai`).

Unfortunately, we were unable to get access to a demo of these products and -to our knowledge- none of them provides a free-to-use edition.

While we could not test the products themselves, we can say that they appear to provide viable solutions to the problem and a polished user experience.

3.2.3 Summary

Table 3.1 shows a direct comparison between the various solutions we reviewed. Overall, each of them has its own strengths and weaknesses, with the CIS CSAT representing the one that is currently closest to being the most effective tool for addressing the problem identified in this thesis. Its limitations, however do -in our opinion- justify the effort of creating a new solution that can be accessed more easily by the users and grants them greater control over their data and their choice of the reference framework.

3.3 Planning

Having completed the analysis of the problem and the survey of the existing solutions, this section presents the initial considerations on how to structure the development of the software we propose in this thesis.

3.3.1 Features

In this section, we detail the features that the program shall have to allow us to satisfy the requirements defined in Section 3.1.

3.3.1.1 Control management

The user shall be able to add the controls to the program in order to store them in a dataset that can be queried. They shall also be able to see all the controls in the dataset and to update them to amend incorrect information or introduce clarifications.

Feature	MSRMP	CryptoAC	Spreadsheets	CIS CSAT
Tool available	●	●	●	●
Open source	●	●	◐	○
Sovereignty over data	●	●	●	○
Support for conflicting interests	●	◐	◐	○
Easy to use	○	●	◐	●
Inspect progress over time	○	○	◐	●
Support for multiple users	○	○	◐	●
Mapping to other security frameworks	○	○	◐	●
Provides a filtering system	○	◐	◐	●
Ability to manage many controls	●	○	◐	●
Helps in prioritizing controls	●	○	◐	○

● = provides the feature; ◐ = partially provides the feature; ○ = does not provide the feature;

Table 3.1: Comparison of existing solutions

3.3.1.2 Measures management

As with the controls, the user shall be able to store and view an arbitrary amount of measures in the database. They shall also be able to modify existing measures.

3.3.1.3 Tags

We introduce a tagging system, allowing the company to divide their dataset of controls and measures by categories. The users shall be able to create an arbitrary number of tags, to associate a tag to multiple controls, and to associate a control with multiple tags to organize the controls according to user-defined categories. As an example, tags may be useful for distinguishing which controls belong to each framework -or part thereof- or to mark all the controls that are concerned with a particular aspect, e.g., authentication.

For now, tags will refer only to controls. This means that there will not be tags for measures. The tags of a measure will be given by the union of the tags of its associated controls. We argue that this design decision encourages the users to focus more on the controls rather than the measures, or, if we consider it in a philosophical perspective, on the end result rather than on how they are going to achieve it, since one aim of the proposed system is precisely to aid them in this part of the process of deciding how to implement the controls.

3.3.1.4 Satisfiability

The user shall be able to associate each measure to an arbitrary number of controls. A control can be associated to multiple measures as well. Every time a measure is associated to a control, it is associated with a user-specified coverage value. The coverage concept is expressed in Section 2.5.3.1.

3.3.1.5 Rankings

The users shall be able to generate rankings -as defined in Section 2.5.5.1-. These ordered lists of measures serve as a baseline to help them prioritize which measures are worth to implement first.

It should be possible to create multiple rankings, and the users should be able to create a ranking only for a subset of the controls. In this regard, the tag system may serve as a filter. This functionality may be useful if, for example, two teams were assigned the task of implementing different parts of the controls of a framework -e.g., one team for the controls of the network and one for those concerning physical access to the facilities-.

The user shall also be able to see the existing rankings and, for each ranking, it shall be possible to see the relative measures and controls.

3.3.1.6 Visibility

The user shall be able to see the current posture of the company with respect to the controls, the measures or a subset of them. This includes, for example, seeing how many controls have been fully implemented and how many are still incomplete.

If possible, this visualization shall make it possible to see the evolution in time of the data it presents so that the users may identify emerging trends in the data.

3.3.1.7 Multi-user

The system shall allow multiple users to interact with it independently and simultaneously. This will make collaborating on the same dataset possible, easing the burden on any single user.

3.3.1.8 Basic security

There shall be an authentication/authorization system in place to restrict access to the system only to authorized users and to ensure that each of them only interacts with the resources they have been authorized for by an administrator.

3.3.2 Proposed workflow

In this section, we imagine at the general level how the interaction of the users with our program should likely unfold.

When they first connect to the system, there will be no data in it. Therefore, the first step will be to add all the controls that the company wishes to satisfy. If the user identifies common themes throughout multiple controls, they may start tagging them accordingly.

Then, they will iteratively add to the system the measures that they intend to implement to fulfil each control, associating each one to the relative control(s). During the execution of this phase, they will be able, from the general view, to see how many controls are present in the dataset and how many measures each of them is associated with. Same for the measures.

When the company feels they have enough measures and controls, they will create a ranking, which will give them an ordered list they can use as reference to prioritize the implementation of the measures.

As the company progresses in implementing the measures, they will update their progress in the program and this will automatically reflect in an increased completion of all the related controls.

At any point, the company may add new controls, measures, tags or create new rankings.

3.3.3 Overarching design decisions

In this section, we briefly present why we chose a client-server architecture and to have the software as single-tenant instead of multi-tenant.

3.3.3.1 Architecture

Given the requisite of having multiple users collaborating on the same data, a client-server architecture immediately appeared as a viable choice [52]. A considerable advantage with respect to a more decentralized architecture is avoiding problems of balancing data consistency, availability and partition tolerance -CAP theorem- [28] by having a single source of truth.

The next architectural decision was about it was best to have a single server managing both the frontend client and the data or two separate servers: one dedicated exclusively to the implementation of the core business logic -managing database access and creating the rankings- and one responsible only for retrieving the data from the first one through an API and providing it to the user in a graphical interface.

We chose to separate the two servers. Throughout the rest of the work, they will be referred to as, respectively, the backend for the server responsible of the interaction with the database and the frontend for the server responsible for managing the application the end-user will interact with.

Given this separation between the two servers, if another developer in the future decides to completely rewrite the frontend, no change is needed in the backend -and vice-versa- as long as the API is left intact.

3.3.3.2 Multi-tenant or single-tenant?

Another decision which was taken in the early stages of the development was wether to make a single-tenant service, where each company would have its own instance of the service or a multi-tenant service where multiple parties would share the same instance.

In light of the facts that (i) we identified data sovereignty as one of the desirable properties our application should provide -see Section 3.1.3-, and (ii) a product with multi-tenancy would require the implementation of a proper layer of logical isolation between the tenants to prevent security vulnerabilities [34], we adopted the single-tenant architecture.

3.4 Backend design

In this section, we present the principal design decisions for the backend server -defined in Section 3.3.3.1-. Namely, we report our motivations for the choice of the language in which we wrote the software and the database management system.

3.4.0.1 Language

A crucial decision at the start of a software development project is represented by the choice of the language(s) to use. Each of them has its own advantages and drawbacks. In this section, we explore the ones we considered as possible candidates for the backend server.

TypeScript is a statically typed superset of JavaScript that embeds the types into its syntax. By integrating type annotations directly into the code, TypeScript prevents a significant number of type-related errors at compile-time.

Unlike JavaScript, which is typically interpreted, TypeScript undergoes a transpilation process to generate JavaScript code that can then be executed by browsers or JavaScript runtime environments like NodeJS[6]. A notable feature is TypeScript's ability to transpile its code in multiple versions of JavaScript, automatically including any polyfills[7] that may be necessary for older versions and excluding them for new ones.

Since TypeScript is a superset of JavaScript, it is compatible with the JavaScript packages available through the Node Package Manager (NPM)[8]. This represents a great strength considering the sheer volume of packages available on the NPM -around 2.5 million, as of now[9]-.

Rust aims to be citing from their website, "A language empowering everyone to build reliable and efficient software"[10]. It is a fast and memory-efficient low level programming language. It was developed with a strong focus on memory safety and preventing common programming errors such as data races and null pointer dereferences, which can result in both application errors and security vulnerabilities like buffer overflows. Rust achieves this safety by enforcing strict ownership and borrowing rules, which allow for safe concurrent access to data without the need for runtime checks or garbage collection.

One of the key features of Rust is its emphasis on zero-cost abstractions. It provides high-level programming constructs that are designed to have minimal impact on runtime performance. This allows developers to write code that is both expressive and efficient, without sacrificing performance-critical operations.

[6] `https://nodejs.org`
[7] `https://developer.mozilla.org/en-US/docs/Glossary/Polyfill`
[8] `https://www.npmjs.com`
[9] `https://github.com/nice-registry/all-the-package-names/blob/master/names.json`
[10] `https://www.rust-lang.org`

Rust is a statically-typed language, with built-in type inference based on the usage of a variable. This can lead to increased reliability and faster development cycles, as developers do not need to specify the type of all their variables. The compiler ensures that programs are free from type-related errors during the compilation process.

Another notable aspect of Rust is its extensive support for concurrent programming. The language includes built-in primitives, such as threads and channels, which enable developers to write safe and efficient concurrent code. Additionally the ownership model ensures that data races and other concurrency-related bugs are caught at compile-time, eliminating the need for expensive runtime checks.

Rust's package manager, Cargo, enables the developers to efficiently manage their project's dependencies, build it, and run tests. It simplifies the process of distributing and sharing Rust libraries and applications, promoting code reuse and collaboration within the Rust community.

Golang is a compiled and performant programming language engineered by Google to be easy to understand and program in[11]. It follows an object-oriented paradigm and offers a concurrency model which allows the developers to easily manage multi-threaded applications using coroutines. The language abd the compiler automatically handle these, allowing to execute multiple tasks even on a single thread.

Golang has a static type system and many of the applications written in Golang can be compiled into a statically-linked binary. This makes it possible to ship them in containers -see Section 3.6.2 on Docker- that contain only the compiled binary, removing the increase in image size and attack surface that comes from including a base image with an operating system, like Alpine[12].

Between these languages, we chose to use Rust as the one for the backend given its strong type and memory safety guarantees and its remarkable performance both in terms of speed of execution and memory efficiency. Golang would have been the second choice, mostly because of our limited familiarity with it.

3.4.0.2 Web server

Having decided which language to use for the backend, we surveyed which web frameworks could be used to ease the task of programming a web server in Rust.

Rocket[13] and Actix[14] appeared to be two popular choices in our initial research. Both framework appear to be well-established and offer all the essential features a web app may need -e.g., middleware, routing, JSON serialization and deserialization-. They also support more advanced features like the use of web sockets.

Both of them make an extensive use of the macro feature in the Rust language in an attempt to reduce the boilerplate the developer is required to write. They also harness the powerful type system of the Rust programming language to enforce at compile time a variety of properties -see, for example, request guards in Section 4.1.5.3-.

For this thesis, we chose to use the rocket framework. This decision was determined mainly by our existing familiarity with it, having participated in some seminars about it with its creator.

3.4.0.3 Database

Having decided to use the Rust programming language for the backend, we started considering which database management system would best suit the requirements of the application.

This section presents the alternatives we evaluated.

[11] `https://go.dev`
[12] `https://hub.docker.com/_/alpine`
[13] `https://rocket.rs`
[14] `https://actix.rs`

MySQL is a staple in DBMSs. It is owned by Oracle and was first introduced in 1995. To date, over 1 million systems worldwide run it as their database[15]. MySQL is a relational database and it uses SQL as its query language. Scaling is achieved through mechanisms like replication, partitioning and sharding [59, §11.2.4].

CockroachDB is a relatively new DBMS written in golang. Its main advantage when compared to the alternatives we are considering in this section is being able to provide an SQL relational database that is built for resiliency and running at scale.

It allows the company run the database across multiple computers, which are joined together by CockroachDB as the nodes of a cluster. To increase the storage capacity or the performance of the database, it is sufficient to add more nodes and the data and the queries are balanced automatically.

CockroachDB also strives to ensure resiliency of the service and the data in case of disasters. It allows the administrators to set a minimum amount of replicas for each record in the database and to distribute them in different geographical regions. If one node becomes unavailable, the other nodes holding replicas of the data create additional replicas until the predefined target is reached.

Geographically-distributed nodes can also contribute to reduce the latency by storing the data and providing access to it from the location that is closest to the users. It is also possible to set policies restricting some -or all- data from being stored inside or outside certain regions, helping with regulatory compliance.

Internally, CockroachDB uses the raft algorithm -[53]- to establish the consensus on the validity of any write operation on the database[16]. This guarantees consistency of the database even in the case of node failures.

Immudb is another relatively new DBMS written in golang. It can operate both as an SQL and as a key-value database. Its distinctive characteristic is that it is an append-only database which allows the clients to verify that the data stored in the database has not been tampered with thanks to all the updates being logged in an auditable and tamper-evident changelog. More details about the immudb architecture and the cryptographic techniques it uses -mainly Merkle hash trees- is available in [54].

In immudb, records cannot be deleted. It is possible to update a record, but only by creating a new version of it -but the two versions of both records will still exist in the changelog-.

As of now, immudb offers SDK integrations with golang, java, JavaScript and python.

From our preliminary analysis, immudb appears to have a good support for scaling through the use of "replication"[17].

SurrealDB defines itself as "The ultimate multi-model database"[18].

It is a DataBase Management System written in Rust. It supports schema-less tables like a NoSQL database, but also gives the user the possibility to declare -and enforce- a schema for its tables through the strict typing feature. This can be particularly convenient during the first phases of development since it allows for faster iterations of the code, opting in to a schema for the table only once the data models have been consolidated.

SurrealDB, however, also supports other database paradigms such as timeseries and graphs, allowing developers to efficiently store and query data that evolves through time -e.g., readings from a sensor- or to model relations between the records as the edges in a graph, with each of these relations optionally having associated attributes.

The data inside SurrealDB can be queried through the use of SurrealQL, a language that bears significant similarities with SQL, but also introduces new operators and statements to support functions like querying the graph relations.

SurrealDB, at the time of writing, provides SDKs for Rust, python, JavaScript, java, WebAssembly and golang.

[15] source: `https://www.statista.com/statistics/809750`

[16] source: `https://youtu.be/20QzUUvfXjI`

[17] `https://docs.immudb.io/master/production/replication.html`

[18] source: `https://surrealdb.com`

In terms of scalability, it can run on a single node -either in-memory or using the disk- or on multiple nodes in a cluster -either on top of a FoundationDB or a TiKV cluster-.

A complete list of the features is available on their website[19].

After having surveyed the candidate DBMSs, we decided to utilize SurrealDB. The choice was motivated by (i) the flexibility that it affords to the developers in the data modeling, (ii) the amount of modern features it provides, including live queries, functions, expressions and access control, and (iii) the numerous SDKs which allow the developers to interact with it in the language they prefer -which should be particularly useful if a need to rewrite the backend in another language or to create other services that communicate with the database should arise-.

Having decided the database management system, we now include a slight digression to consider its license terms.

In software development, it is important to check that, for every component you use, you are in compliance with its license terms. The license may or may not grant you the right to utilize the software how you would need to do. For example, if a library you want to use is licensed AGPLv3, then you can only use it if you make your product open source too.

In the case of our database management system, SurrealDB, we found multiple licenses. As stated on their website[20], their various software components are distributed under multiple licenses, mostly Apache 2.0 which is a permissive license that allows for commercial use and does not mandate the disclosure of the source code.

The core database software, however, has a different license, the Business Source License, which basically says that you can't use it to offer a competing database service until 2027. If you want to make commercial use that is not granted by the license, it is your duty to purchase a commercial license from SurrealDB or refrain from using the software.

According to the license, you may use SurrealDB as you please as long as you do not offer a "Database Service", meaning a commercial service where the customers can choose the schema of the tables in the database.

Kampas, at the time of writing this thesis, is in compliance with this license since it (i) does not allow the end-user to specify the schema of the tables since everything is statically-typed in the Rust code, and (ii) is not a commercial offering.

More details on the license can be found at `https://github.com/surrealdb/surrealdb/blob/main/LICENSE`.

3.5 Frontend design

The aim of the frontend is to provide the end-users with a graphical interface through which they can interact with the backend. To create it, we have several options. A first dichotomy can be considered between native and web applications.

A native application would run directly on the device of the user and will likely need to be compiled for each platform we intend to support. A web version, on the other hand, would run inside a web browser and, therefore, be more portable across different platforms and architectures. If we consider access to sensors or specialized hardware resources, native applications usually have an advantage over web applications in the variety of APIs they can use.

Given that our application does not have particular requirements in terms of access to system APIs and hardware peripherals nor does it require expensive computation on the client, a web application appears to be the preferable approach given the better cross-platform compatibility.

3.5.1 Candidates

Having decided to create a web application, we briefly survey the candidate web frameworks we could use to implement such a project.

[19] `https://surrealdb.com/features`
[20] `https://surrealdb.com/license`

We considered the following options: Next.js[21], Nuxt[22], SvelteKit[23] and Tauri[24].

The first three of these options share significant similarities between one-another. Each of them enables the developers to write performant web applications providing powerful routing systems. All of them support capabilities like SSR -Server-side Rendering- and SSG -Static Site Generation- which can be used to optimize page load times and for SEO -Search Engine Optimization- purposes. All of them allow the programmer to specify which rendering strategy they wish to adopt as the default, but also to define exceptions directly at the route level. All of these frameworks promote the use of a component-based architecture and provide a thriving ecosystem of plugins and libraries that can help speeding up the development process.

The principal difference between these three web frameworks is in the frontend framework they use: Next uses React, Nuxt uses Vue and SvelteKit uses Svelte.

Tauri, on the other hand, is a framework for building multi-platform desktop applications using a mix of Rust and web technologies -HTML, CSS and JavaScript-. A Tauri application runs essentially as two processes: one is a a native application built in Rust serving as a backend and the other displays the UI in a webview commonly referred to as frontend. The frontend can communicate with the backend through the use of a `Command` and the webview is provided by the host's default browser. At the moment, Tauri supports Windows, MacOS and Linux, but it has plans to expand to Android and iOS as well.

Among these four alternatives, we decided to proceed using SvelteKit. The choice was mainly motivated by our existing familiarity with the UI framework, which is the main differentiating factor between SvelteKit, Nuxt and Next.js.

3.6 Additional elements

In this section, we briefly present the auxiliary technologies we used to streamline the development process of the solution.

3.6.1 Git

Version control is considered to be a best practice in software development [61, §22.1]. For this project, we chose to use git [44] as our version control system, given our existing familiarity with the tool and its widespread adoption.

Git tracks changes made to the files present in a given directory -known as a git repository- by letting the developer create a history of the revisions through the use of commits. Every time a developer creates a commit, git takes a snapshot of the content of the repository and stores internally the differences between the current commit and the previous one. This creates a timeline -known as a git branch- in which it is possible to see the evolution of the code and identify the modifications that occurred. It also allows the developers to move back in this history to see previous version s of the code. A git branch can also branch in two separate directions. This enables parallel development, with one team working on one feature while another team works on another. These additional branches can then be merged back into the main one -usually this is done when they have been completed-. Git has a decentralized architecture in which each developer has a complete copy of the repository on their disk, making it tolerant to network failures and enabling offline work.

In the case of our project, we chose to use git for version management and host the repository both on our computers and on GitHub to have the possibility to make the code publicly accessible and to create a backup copy of the repository in case we experienced some hardware failure.

3.6.2 Docker

Docker is a containerization platform adopted both in software development and software deployment [46]. It allows developers to package their applications -and all their dependencies- into portable images and run them. These images are then instantiated as containers [9].

[21] `https://nextjs.org`
[22] `https://nuxt.com`
[23] `https://kit.svelte.dev`
[24] `https://tauri.app`

A container is a running instance of an image. Conceptually, it bears heavy similarities with a Virtual Machine, but the two technologies have significant differences. Containers are generally more lightweight [60, §5.1, §6.1] and performant [38], [60] than virtual machines.

Since an image bundles all the application's dependencies inside itself, the problems of missing dependencies is greatly reduced. Also, since the dependency is present in the container itself at runtime, nothing prevents two containers to run simultaneously using each a different version of the same dependency.

Docker, therefore is a powerful tool for deploying software: once the application is bundled into a docker image, it can be run on a multitude of environments without the risks of missing some dependencies or of some dependencies conflicting with those of other applications running on the same host.

Another popular use-case for docker is development: it is possible to create a docker image containing all the dependencies needed to develop a project -programming languages' SDKs and libraries for example- and ensure that all the developers have them at their disposal [9].

In the case of our project, both the backend and the frontend require additional services in order to operate: the databases. For the backend, we chose to use SurrealDB -see Section 3.4.0.3-. For the frontend, we used Redis to manage the users' session data. Instead of installing these services directly on the host where we were performing the development, we instantiated them as docker containers.

3.6.3 Nix

Nix is a package manager developed by E. Dolstra during their PhD [16]. It emphasizes reproducibility and explicit declaration of dependencies, both at build time and runtime.

Each nix package is represented as an *expression* written in the Nix language[25]. This expression includes all the necessary inputs for building and running the package -e.g., source code, build-time dependencies, and runtime dependencies- along with instructions to build the final artifact, known as a *derivation*.

Derivations are stored in the nix store, which acts as a repository of packages managed by Nix. Nix takes dedicated steps to protect packages from tampering after they have been built. A package can be obtained by either evaluating the corresponding Nix expression -most times this equates to building it from source- or by downloading it from a package *cache* -some trust considerations in this regard are detailed in [16]-.

Since its introduction, Nix has found various applications, including enabling the creation of NixOS [17], a purely functional, declarative Linux distribution.

For development purposes, Nix provides the ability to create environments with all the necessary tools for a project, such as compilers and language SDKs, using tools like `nix-shell`.

It is also possible to use Nix to obtain reproducible and developer-friendly development environments [64, §4.3.2]. In this regard, an additional tool named direnv[26] can automatically load (and unload) these environments as soon as the user enters (leaves) a project's directory.

With these tools, it is possible to specify all the project's dependencies in a file. Direnv will use Nix to automatically install and provision them when entering the project directory.

In our project, we used Nix and direnv to provision the necessary packages for Rust and SvelteKit development, along with the code editor and required extensions.

[25] see `https://nixos.org/manual/nix/stable/language/index.html`

[26] `https://direnv.net`

4 Implementation

Having defined the requirements and drafted an initial implementation plan in Chapter 3, we started implementing the software itself. Its development process is reported in Section 4.1 for what concerns the backend and in Section 4.2 for the frontend part. While the processes are described separately in distinct sections, in reality their development did proceeded in sync: as we implemented features in the backend, we continuously wired those to the frontend and tested their functionality.

4.1 Backend

The core of the kampas project is surely represented by the backend. This component is responsible for storing all the controls and the measures of the company in the database and providing access to them. It also implements the business logic necessary to create new rankings of the measures and provides the access control logic which ensures that each user can access only that they are authorized to access.

The backend is written in the Rust[1] programming language and is structured as a single binary crate named `kampas`. As anticipated in Section 3.4.0.2, we chose to use the Rocket[2] framework to support us in the development of the server. The following sections detail the main components of the system and how they interact with one-another.

4.1.1 Models

The models are the components we use to represent the entities we need to act on -controls, measures, rankings, &c.- into our programming language of choice. We defined the role of each one of them in Section 2.5.

4.1.1.1 Control

The control -with the semantic we defined in Section 2.5.1.1- is the core entity of our program. Everything revolves around it since it is the first piece of information that the user acquires, during the review of the frameworks -see Section 3.3.2-. A control represents for the user what they ultimately want to achieve. The end-goal of their process is to satisfy the controls. They represent the security requirements.

We model a control with the following attributes:

- **identifier**: A probabilistically-unique 16-characters random hexadecimal `String` that distinguishes this control from all the others.
- **title**: A title for the control. Ideally, this should be a short string that allows the user to differentiate two controls from one-another. As an example, the title for a control may be "NIST DE.CM-2", referring to the continuous security monitoring control of the detect section of the NIST CSF. This field represented as a `String`.
- **description**: A long-form description of the control, providing information about the context in which it is supposed to be applied or a space for notes. Coherently to the example title we just mentioned, a description may be borrowed from the NIST framework itself: "The physical environment is monitored to detect potential cybersecurity events". This field is also represent as a `String`.
- **created_at**: Timestamp in UNIX milliseconds UTC of when this control was created. Stored as a `String`.

[1] `https://www.rust-lang.org`

[2] `https://rocket.rs`

Here we see one of our design decisions: to include an identifier as part of the control. The alternative would have been to have it independently of the control, generated by the database and managed by returning a tuple (control, id). This could have been generalized into its own Rust struct -see Listing 1- which would be applicable not only to the controls, but also to the other entities. Despite having considered it, we decided not to introduce this abstraction in order to maintain all the information in one struct. This also proved convenient once we started using the crate `ts-rs`[3], which allowed us to generate programmatically the TypeScript type definitions for the frontend code -see Section 4.2.2-.

Listing 1 Wrapper struct to represent an object and its identifier in Rust

```
struct DBObject<T>{
    object: T,
    id: String
}
```

4.1.1.2 Measure

After having implemented the `Control` struct, we proceeded with the measures. As said in Section 2.5.1.1, a measure conceptually represents a concrete action -or set thereof- that the company can undertake in order to satisfy in full or in part a security control.

To represent a measure in our program, we chose the following structure:

- **identifier**: A probabilistically-unique 16-characters random hexadecimal `String` that distinguishes this measure from all the others.
- **title**: A title for the measure. This should be a short string allowing the users to distinguish it from the other measures. Stored as a `String`.
- **description**: An textual description of the measure. Here the users may put details like the tools they will want to integrate in this measure. Stored as a `String`.
- **progress**: This is an integer representing the progress of the implementation. We arbitrarily decided to have in a range between 0 and 100, inclusive, with 0 signifying "not started yet" and 100 meaning "complete". In our opinion, this range should provide a sufficient level of granularity whilst still enabling the users to use it more coarsely if they so wish. That said, it is trivially modifiable at the source-code level. Being in the range $[0 - 100]$, this datum is stored as an `u8`, since this primitive type can accommodate values in the range $[0 - 255]$.
- **effort**: This field represents the amount of -estimated- effort that is necessary in order to bring this control from zero to fully implemented across the company -see Section 2.5.1.1-. We decided to let each company decide in which unit of measure they evaluate their effort. Two possible units we had in mind whilst designing this were 'weeks of work' or 'manhours of work', but each company may choose the most suited one. As of now, it is stored as a `u8` since we were not able to think about any reason why an effort may be negative.

4.1.1.3 Tag

Tags -as presented in Section 3.3.1.3- are a way in which the users can organize their controls and, by extension, their measures. A tag comprises the following fields:

- **identifier**: A probabilistically-unique 16-characters random hexadecimal `String` that distinguishes this tag from all the others.

[3] `https://docs.rs/ts-rs/7.0.0/ts_rs`

- **name**: The name for this tag. This is decided by the user and some examples may be "NIST", "NIST CSF", "Access Management" or "NIST, Respond, Communications". A tag may refer to anything: a framework, a group of controls, or even a specific tool -e.g., Splunk SOAR[4]- or a broader category like "Network Security" `String`.
- **color_hex**: This field contains a color for the tag, represented as a six digits RGB hexadecimal color code. Being able to have a color in the tags may result helpful in distinguishing them from one-another. The company may choose to have a different color for each tag, or to visually group some tags by assigning them similar or the same color. This filed is stored as a `String`.
- **created_at**: Timestamp in UNIX milliseconds UTC of when this control was created. Stored as a `String`.

4.1.1.4 Ranking

The ranking object represents which measures the users shall implement in order to satisfy the controls in an efficient manner. It has the following properties:

- **identifier**: A probabilistically-unique 16-characters random hexadecimal `String` that distinguishes this ranking from all the others.
- **name**: A human-readable name for this ranking. Stored as a `String`.
- **minimum_coverage**: The minimum coverage value that a control must have in order to be considered satisfied -see Section 2.5.3.1 for how this is defined-. Stored as a `u8`.
- **measures**: A list with the identifiers of the measures that are present in this ranking sorted in order of increasing cost -see Section 4.1.3 for the creation strategy-. Represented as a `Vec<String>`.
- **controls**: A list with the identifiers of the controls that this ranking is able to satisfy once all its measures are completely implemented. Represented as a `Vec<String>`. Identifiers of the controls
- **created_by**: The username of who created this ranking, stored as a `String`.
- **created_at**: Timestamp in UNIX milliseconds UTC of when this ranking was created. Stored as a `String`.
- **ordering** A human-readable string representing which algorithm was used to create this ranking. Represented as an `enum`, to ensure that only predefined values can be entered. The enum is serialized to and deserialized from a `String`. As of now, there is a single allowed value of "greedy_w_set_cover".

4.1.1.5 User

This object represents a user of the service. It has the following properties:

- **username**: A unique `String` that serves as an identifier for this user. It can only consist of lowercase alphanumeric characters [a-z,1-9] or underscores. Its length has to be greater than 3 and smaller than 30. Usernames that start or end with an underscore are not allowed.
- **password_hash**: The hash of the password of this user, stored as a `String`. This string is in PHC String Format [55].
- **roles**: A list of the roles that this user has. By default, it is empty when a user gets first created. Valid roles are defined in a separate `Role` enum -e.g., "Admin", "GetControls", "CreateTags", &c.-. Stored as a `Vec<Role>`.

[4] `https://www.splunk.com/en_us/products/splunk-security-orchestration-and-automation.html`

4.1.1.6 Token

A token represents an authenticated user on the service. In order to access any protected endpoint, the client must supply a valid token to be regarded as the corresponding user -see also Section 4.1.5.3-. A token object has the following properties:

- **token**: A probabilistically-unique 64-characters random hexadecimal `String` that represents the token.
- **username**: The username of the user that this token refers to. Stored as a `String`.
- **expiry**: The moment at which this token expires and is not considered valid anymore. Stored as a `DateTime<Utc>`.

4.1.2 Database

Having implemented the models, we coded the interactions with the database. As said in Section 3.4, we chose to use SurrealDB for this project, and we refer the reader to Section 3.4.0.3 for an overview of its features. This section details how we setup the database to be able to connect to it -Section 4.1.2.2-, to insert and update the data -Section 4.1.2.3-, and how we used a third-party tool to obtain a graphical user interface, which proved to be valuable in debugging during the development process -Section 4.1.2.4-.

4.1.2.1 Database instance

SurrealDB offers various possibilities to configure an instance. It is possible to have running in-memory -therefore with no persistence of the data-, on top of a TiKV[5] cluster or in a single node persisting the data to the filesystem.

For our use-case, we decided to run it in the third configuration: as a single node persisting data to the filesystem. This choice was motivated by the need to have data persistence -which ruled out the in-memory option- and the intention to avoid the overhead of setting up a TiKV cluster.

In this project, SurrealDB runs inside a docker container, using the official image from Docker Hub[6] `surrealdb/surrealdb:1.0.0-beta.11`.

4.1.2.2 Connection setup

The first step in any program that communicates with a database is establishing a connection to it. In our case, we used the official crate for SurrealDB: `surrealdb` -version 1.0.0-beta.9-.

At a general level, the crate allows us to instantiate a connection by providing the URL of the database instance, the namespace, the name of the database and the credentials to access it.

The values of these properties are traditionally regarded as having different levels of sensitivity. The database instance, the namespace and the name of the database are usually considered configuration values, and it is generally preferred to store them in a config file rather than in the source code for ease of configuration and separation of concerns. The credentials, on the other hand, -the username and the password- represent more sensitive information, and are therefore kept outside of configuration files and usually supplied as environment variables.

We used the `config` crate in its version 0.13.3, allowing us to specify any of these values either in a configuration file or in an environment variable, with the latter taking precedence in case of multiple definitions. This allows maximum flexibility for the end-user, enabling them to use only environment variables -following the philosophy of the twelve-factor app [72]-, only config files, or a combination of both. Also, having the possibility to use environment variables to override the values specified in the config file allows the users to undertake a cyber-deception approach, by inserting canary tokens -also known as honeytokens [45], [62]- in the configuration files and overriding them with environment variables for the actual instance of the app. This can help detecting unauthorized access attempts.

We define the following config variables, whose names should be self-explanatory by now:

[5] `https://tikv.org`

[6] `https://hub.docker.com`

- "surreal_url",
- "surreal_namespace",
- "surreal_database",
- "surreal_user", and
- "surreal_password".

As the program starts, the values of these variables are read from the configuration. Then, we establish the connection to the database. We implemented a retry logic that attempts to connect to the database 12 times, waiting 10 seconds between each attempt to accommodate for the possibility of the database being started at the same moment as the backend.

If it is not possible to establish a connection within the specified number of attempts, the program panics and terminates. If, instead, we were able to create a connection, it is passed to rocket as one of the properties that constitute the state of our server[7]. Rocket will then provide this connection to all the request handlers that need it during the execution of the program. For a request handler to declare its need to access the database, it is sufficient for it to accept a parameter of type `&State<Surreal<Client>>`.

4.1.2.3 Interacting with the database

Having established a connection to the database, the program can now start reading and writing data to it. We defined in the code a series of helper functions, which help separating the logic for interacting with the database from the rest of the program, and discuss them here grouped by the entity to which they refer to. Each of the items we are about to detail corresponds to a function defined in the final code. For brevity, we omitted the SurrealDB client in the parameter lists, but it is always present as all these functions need to interact with the database. Unless otherwise specified, the `Result<T>` type refers to a `surrealdb::Result<T>`, therefore representing either a successful value of type `T` or an error from the SurrealDB client.

Control

1. `add_control(c: Control) -> Result<()>`

   This method is responsible for adding a new control, `c`, to the database. It returns a result containing either the unit type -to signal a successful creation-, or an error that prevented the new record from being created.

2. `edit_control(c: Control) -> Result<()>`

   This method allows to update the contents of a control with new ones. The identifier of the control will remain unchanged.

3. `get_controls() -> Result<Vec<Control>>`

   This method returns all the controls present in the database.

4. `get_controls_for_measure(measure_id: String) -> Result<Vec<Control>>`

   This method returns all the controls that are being satisfied by the measure specified as parameter. If the measure identifier that was specified is inexistent, it simply returns an empty vector. To obtain the controls associated to a measure, we run the following query: ``SELECT * FROM control WHERE id INSIDE $measure_id->satisfies.out''.

5. `get_control(control_id: String) -> Result<Option<Control>>`

   This method returns a control given its identifier. If no control with such identifier is present in the database, it returns a `None` value.

[7] `https://rocket.rs/v0.5-rc/guide/state`

6. `get_control_completion(control_id: String) -> Result<f64>`

   This method returns the completion of a control, as defined in Section 2.5.3.1, given its identifier.

   The completion is computed by adding the contributions of all the associated measures. If the specified control does not exist, it returns zero. To get the measures associated to a control and the relative coverage, we use the relation feature of SurrealDB and run the following query: ``SELECT coverage, in.progress as progress FROM $control_id<-satisfies''.

7. `get_control_completion_batch(control_ids: Vec<String>) -> Result<Vec<f64>>`

   This method is analogous to `get_control_completion`, but accepts multiple control identifiers and, correspondingly, returns multiple progresses. It allows the backend to combine into one what would otherwise have been multiple queries to the database.

8. `get_measures_for_control_count_batch() -> Result<HashMap<String, u64`

   This method returns a map in which every control identifier present in the database is associated with a number representing the number of measures that satisfy said control. To obtain this information, we run the following query: ``SELECT id, count(<-satisfies) FROM control''.

Measure

1. `add_measure(measure: Measure) -> Result<String>>`

   This method is responsible for adding a new measure -provided as the first parameter- to the database. It returns either the identifier of the newly-created measure or the error that prevented its addition to the table.

2. `edit_measure(m: Measure) -> Result<()>`

   This method allows to update the contents of a measure with new ones. The identifier of the measure will remain unchanged.

3. `get_measures() -> Result<Vec<Measure>>`

   Returns a vector with all the measures that are present in the database.

4. `get_measures_for_control(control_id: &str) -> Result<Vec<Measure>>`

   This method returns the measures that satisfy the control with the provided identifier, as a vector. If the control identifier is inexistent, an empty vector is returned.

5. `get_measure(id: String) -> Result<Option<Measure>>`

   This method returns the measure with the given identifier, if any such measure exists in the database.

6. associate_measure(measure_id: &str, control_id: &str, coverage: u8) -> Result<()>:

   This method registers in the database the association between a measure and a control -whose identifiers are provided as parameters-. This is done by running the following query on the database: ``RELATE $measure_id->satisfies->$control_id SET coverage=$coverage''.

7. `get_measure_control_association()-> Result<BTreeMap<String, Vec<(String, u8)>>>`

   This method returns a map where the identifier of each measure present in the database is associated to a vector containing tuples. Each of these tuples contains the identifier of a control the measure satisfies and the coverage of that measure in relation to that control. To obtain this data, we run the following query: ``SELECT in as measure, out as control, coverage FROM satisfies''.

8. `get_coverage_for_measure(measure_id: &str) -> Result<HashMap<String, u8>>`

   Given the identifier of a measure, this method returns a map that associates the control_ids satisfied by this measure to their coverage.

9. `get_tags_for_measure(measure_id: &str) -> Result<Vec<Tag>>`

   This method returns the tags associated to a measure given its identifier. As specified in Section 3.3.1.3, a measure does not have any tag associated directly to it. Instead, it inherits the tags from all the controls it satisfies. To obtain the tags, we run the following query on the database: ``SELECT * FROM $measures->satisfies.out<-tags.in.*''.

10. `update_measure(measure: Measure) -> Result<String>`

    This method updates an existing measure present in the database with the provided data, returning its identifier if the operation is successful.

11. `get_number_controls_batch(ids: Vec<String>) -> Result<Vec<u64>>`

    This method accepts as its first parameter a vector of measure identifiers. For each of them, it returns how many controls are currently being satisfied by such measure. The query which provides this data is the following: `SELECT id as ids FROM $measure_ids->satisfies''`.

Tag

1. `add_tag(tag: Tag) -> Result<String>`

   This method adds the provided tag to the database, returning its identifier in case of success and the error that prevented the creation of the new tag in case of failure.

2. `get_tags() -> Result<Vec<Tag>>`

   This method returns a vector containing all the tags present in the database.

3. `tag_control(control_id: &str, tag_id: &str) -> Result<()>`

   This method associates an existing tag to an existing control. The identifiers of the tag and the control are provided as parameters. This is the query used to perform the association: `RELATE $tag_id->tags->$control_id''`.

4. `untag_control( control_id: &str, tag_id: &str) -> Result<()>`

   This method removes the association between a tag and a control, provided their identifiers. In order to achieve this, we run the following query: ``DELETE FROM tags WHERE in=$tag_id AND out=$control_id''.

5. `get_tags_for_control(control_id: &str) -> Result<Vec<Tag>>`

   This method returns all the tags in the database associated to the control with the specified identifier. To get these tags, we run the following query on the database: `SELECT * FROM tag WHERE id INSIDE $control_id<-tags.in''`.

6. `get_tags_batch(control_ids: Vec<String>) -> Result<Vec<Vec<Tag>>>`

   This method enables us to obtain, given a vector of control identifiers as parameter, another vector where for each control identifier we have a vector containing the tags associated to that control. In order to obtain these, we run the following query: ``SELECT * FROM $control_ids<-tags.in.*''.

7. `get_tag_control_association() -> Result<HashMap<String, HashSet<String>>>`

   This method returns a map where for each tag identifier present in the database, we have an associated set containing the identifiers of all the controls that are tagged by that tag. We run the following query in order to access this data: ``SELECT in as tag, out as control FROM tags''.

8. `get_measure_tag_ids_batch() -> Result<HashMap<String, Vec<String>>>`

   This method returns a map in which the identifier of each measure present in the database is associated to the list of identifiers of the tags that tag such measure. Coherently with the definition established in Section 3.3.1.3, this means that we get all the controls associated to each measure and combine all the tags that tag such controls. We do this by running the following query: ``SELECT id, array::distinct(->satisfies.out<-tags.in) as tags FROM measure''.

Ranking

1. `add_ranking(ranking: Ranking) -> Result<String>`

   This method persists the provided ranking in the database, returning its identifier upon completion.

2. `get_rankings() -> Result<Vec<Ranking>>`

   This method returns a vector containing all the rankings present in the database.

3. `get_ranking(id: $str) -> Result<Option<Ranking>>`

   This method returns the ranking associated with the provided identifier -if such a ranking is present in the database-.

User

1. `add_user(user: User) -> Result<()>`

   This method creates a new user in the database using the provided data. The unit type is returned to signal a successful operation.

2. `does_user_exist(username: &str) -> Result<bool>`

   This method returns a boolean value indicating wether a user with the provided username is present in the database.

3. `get_user(username: &str) -> Result<Option<User>>`

   This method returns all the information about a user given its username -if it is present in the database-.

Token

1. `add_token(token: &AuthToken) -> Result<()>`

   This method creates a new token in the database with the provided data, returning the unit type to indicate a successful creation.

2. `get_token(token: &str) -> Result<Option<AuthToken>>`

   This method retrieves the information of a token -i.e., it's associated user and expiry date- given the token string itself, which is unique to this token.

3. `get_token_for_user(user: &str) -> Result<Option<AuthToken>>`

   This method retrieves the information of a token -i.e., it's associated user and expiry date- given the token string itself, which is unique to this token. The result is an `Option` since there may be no token for the provided username.

4. `delete_token(token: &AuthToken) -> Result<()>`

   This method deletes the token from the database identified by the provided token string parameter.

4.1.2.4 Surrealist

During the process of developing the interactions between the backend and the database, we found it useful to have access to a graphical client for SurrealDB to check the state of the database directly, without passing through the backend.

SurrealDB plans to offer a graphical interface but -at the time of writing this thesis- it has not been released yet[8]. Nonetheless, we found an existing product that offered the exact functionality we were looking for: Surrealist[9].

Surrealist is an open source project -licensed with the MIT license- that provides a graphical client for SurrealDB. It has both a desktop and a web client, with the former offering more possibilities of interaction with the database -see Figure 4.2a-. For our use-case, the web client was perfectly adequate.

Functionalities The Surrealist web client has two main views: one for queries and one for exploring the database.

Query View The query view -visible in Figure 4.1- allows to send one or multiple queries to the database and view the result of each of them either as JSON or in a tabular format. If there is an error in the query, it is shown in red text in place of the result, making it obvious that something was not specified correctly. This client also supports parametrized queries by specifying the values in the underlying "variables" section and interpolating them in the query using the SurrealQL syntax of a dollar sign followed by the variable name -e.g., "SELECT * FROM control WHERE id = $control_id"-.

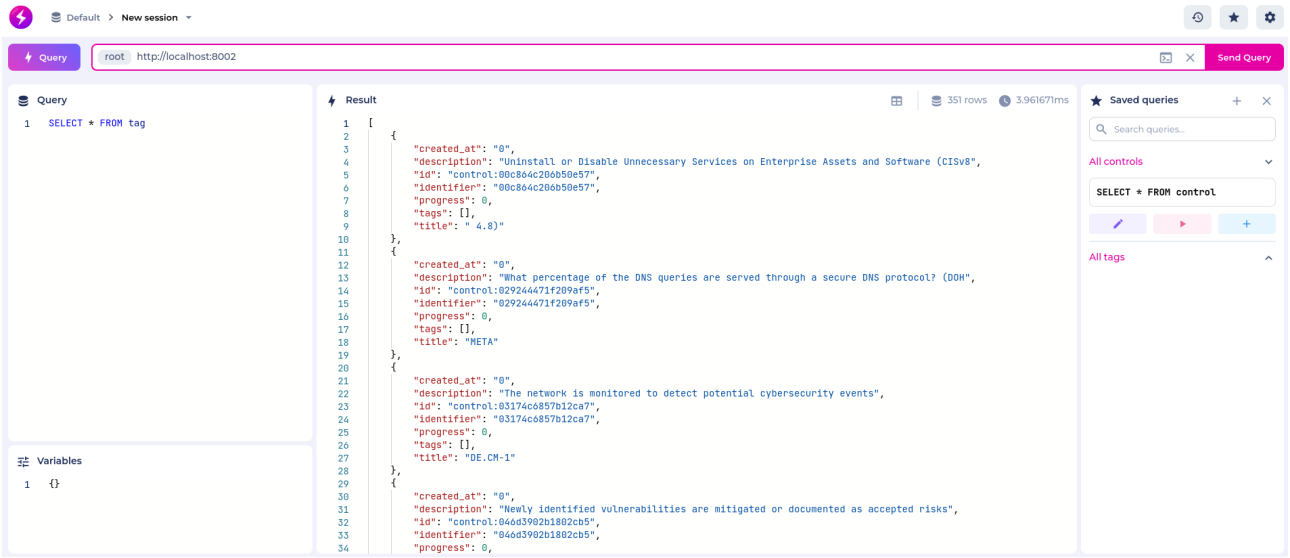


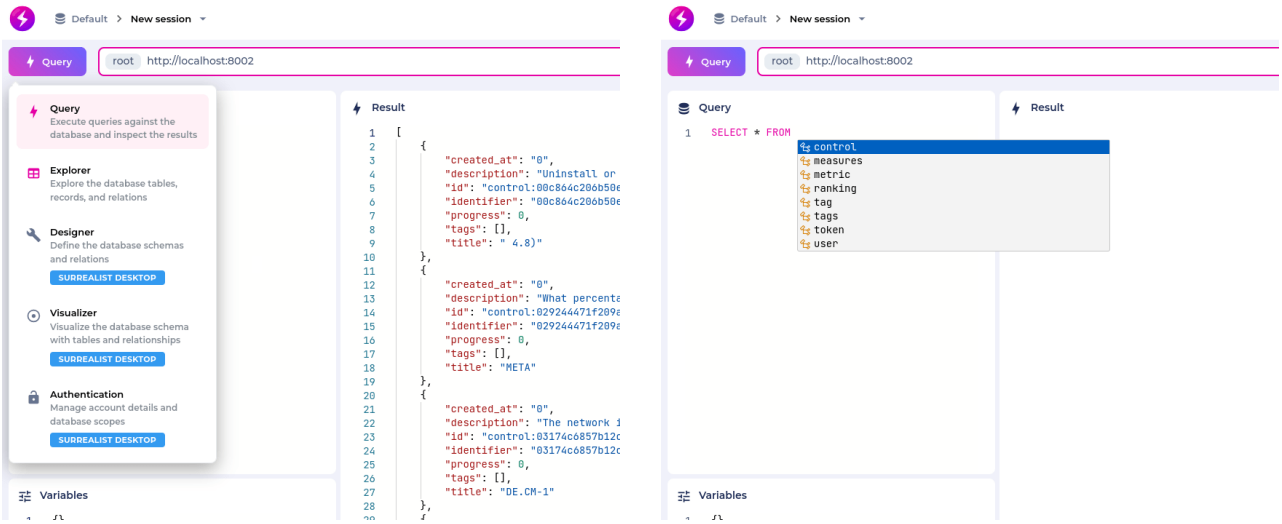
Figure 4.1: Surrealist web app. Query tab with a query and its result

In the query editor, a query autocompletion feature dynamically suggests the name of the tables as the user is typing based on those present inside the database -see Figure 4.2b-.

Database Explorer The other section of the UI is the database explorer. In this view, it is possible to see on the left the names of all the tables present in the database and, by clicking on one of them, show a detailed view in the center -as shown in Figure 4.3-.

8 `https://web.archive.org/web/20230917183228/https://surrealdb.com/features#tooling`

9 `https://github.com/StarlaneStudios/Surrealist`



(a) Menu interface (b) Autocompletion feature

Figure 4.2: Surrealist web app. UI details

If we then click on one of the rows present in the table view, a new dialog opens where it is possible both to see and edit the content of the record -Figure 4.4a- and to see a list of the relations that this record is part of -Figure 4.4b.

Instance Initially, we used Surrealist through the instance available publicly[10]. For this to work, we had to expose the container of SurrealDB -defined in Section 4.1.2.1- on a public IP address. To do so, we used the service offered by Serveo [11]. To expose a local service on serveo, it is sufficient to run this command: `ssh -R 80:localhost:5173 serveo.net`, replacing 5173 with the port on which the service is running on the current host. Serveo will reply with a public address that can be used as the hostname on the Surrealist instance.

After a couple of weeks, however, Surrealist stopped displaying the tables in the database explorer view. Inspecting the logs in the browser, we found no trace of errors. Supposing this may have been an upstream update that had introduced this bug, we decided to run locally a previous release of the service. We chose to use the image available on Docker Hub under the name `0xmimir/surrealist`, which packaged Surrealist version 1.7.5. The local instance, besides not exhibiting the problem, also had no need to access the database through a public URL. Therefore we also stopped using the Surrealist public instance and Serveo for this project, in favor of the local one.

4.1.3 Creating rankings

This section details the process of creating a new ranking on the server. It describes the implementation of the ranking concept specified in Section 3.3.1.5.

To create a ranking, the controls present in the database are first filtered by tag. This accommodates the possibility to let the user exclude from the ranking creation process the measures that would not benefit the filtered controls, if they so wish.

Then, we run the greedy weighted set cover algorithm on the set of measures associated with the filtered controls to determine which are worth implementing and in which order.

4.1.3.1 Filtering the controls

The filtering step on the controls is conceptually simple. First, we get all the controls from the database. Then, we retain only the controls that satisfy the filtering criteria, expressed as a set of tags. The user can ask us to include either only the controls that have all the selected tags, or all the controls that have at least one of the selected tags.

[10] `https://surrealist.app`
[11] `https://serveo.net`

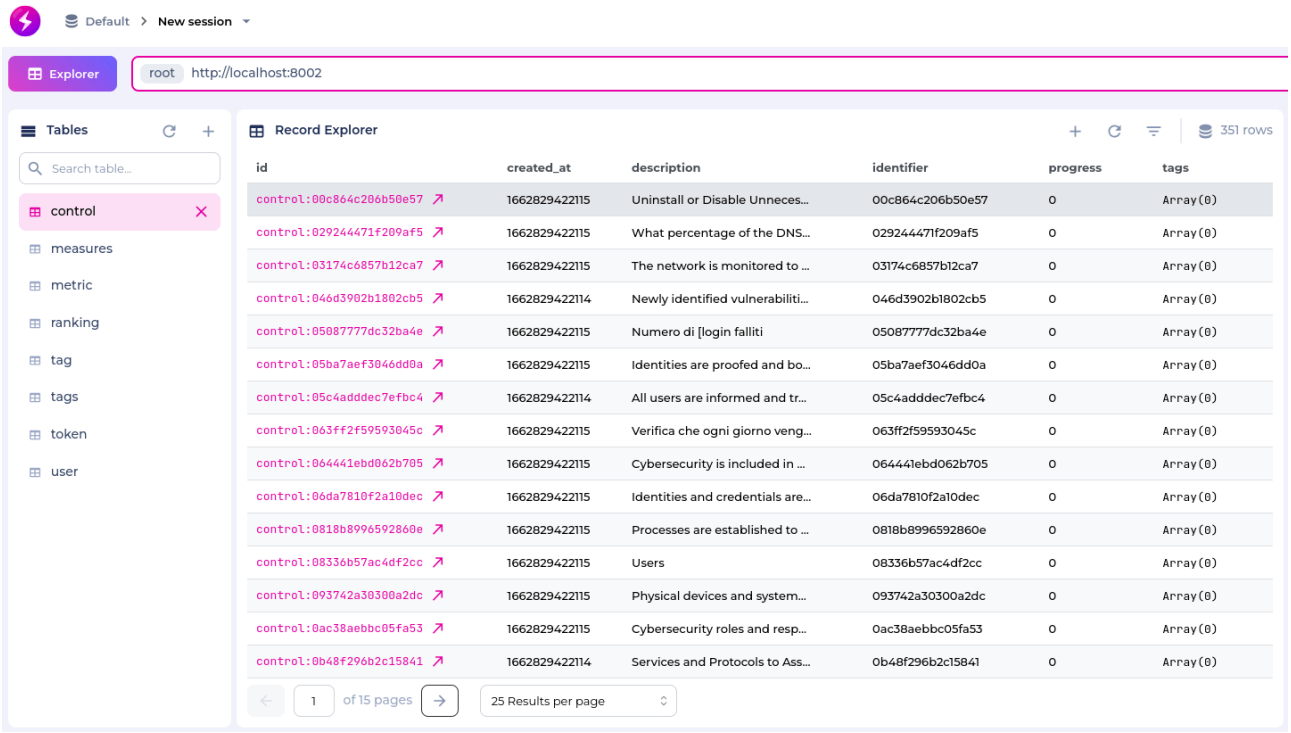


Figure 4.3: Surrealist web app. Tables tab with the controls table selected

4.1.3.2 Ranking algorithm

At this point, we run the greedy weighted set cover algorithm, similar to what is presented by Vazirani in [68]. The idea behind this algorithm is to build the solution iteratively. We start from an empty set, representing our ranking. First, for every measure that we have at our disposal, we compute the cost that would be associated to introducing such measure in the ranking. Then, we choose the measure with the smallest one. We add this measure to the solution, and repeat the process of computing the costs and adding the measure with the smallest one to the ranking with all the measures we have yet to include, until we reach a stopping condition -when either all controls are covered, we do not have any more controls to cover or we included all the measures-.

For this algorithm, the definition of cost is fundamental. In our case, the cost $\gamma_{(m,\ C)}$ of a measure m for a set of controls C is given by Equation (2.1).

Measure Cost Algorithm The pseudo-code of the algorithm to compute the cost associated to a measure is reported in Algorithm 2. It takes the following inputs:

1. `m`:

   The measure being considered -see Section 4.1.1.2-.

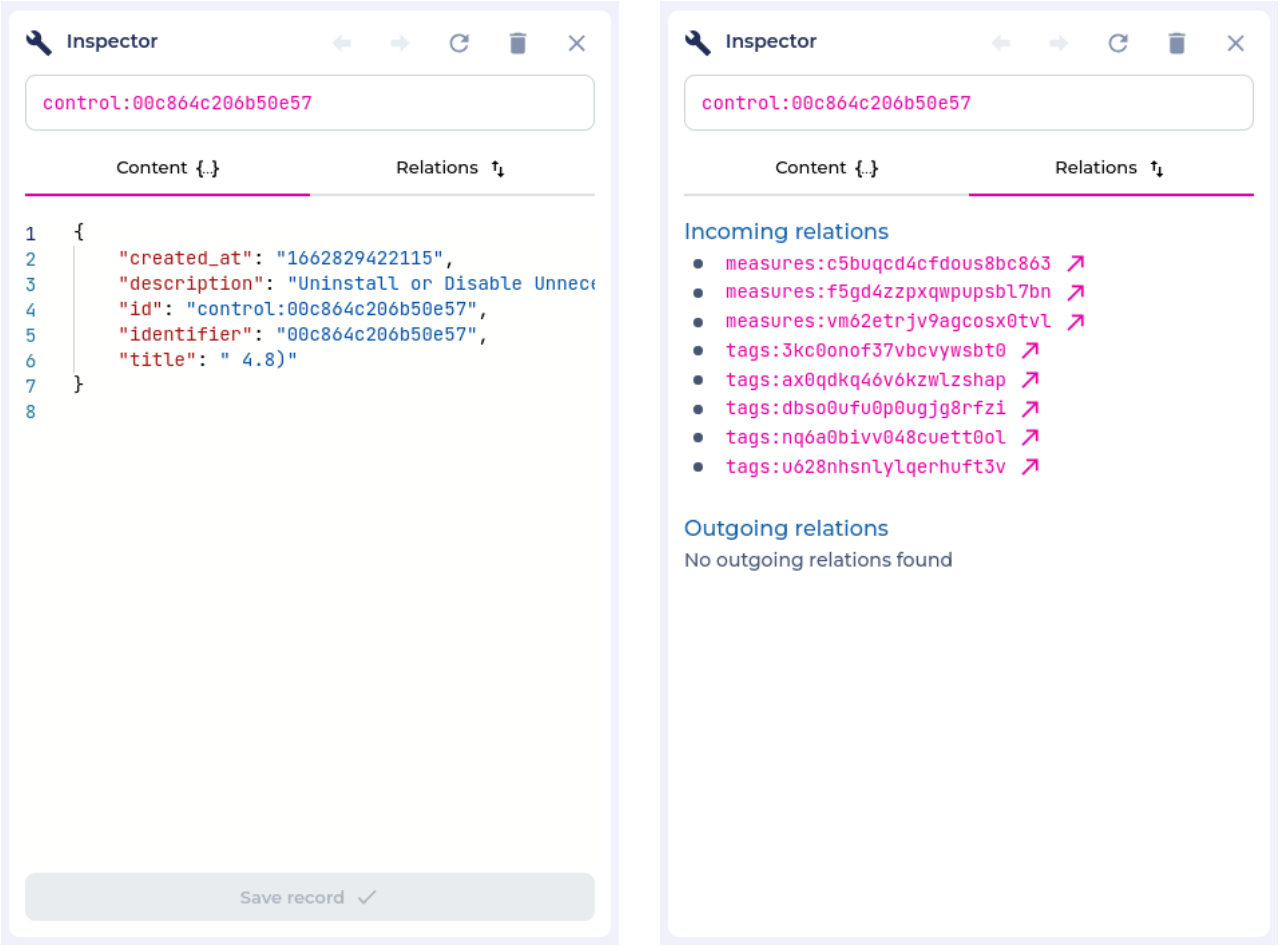
2. `controls_covered`:

   A list of tuples representing which controls are covered by the current measure and the relative coverage values -see Section 2.5.3.1-. Each tuple contains as its first element the identifier of a control that the measure satisfies and, as its second element, the coverage value relative to that measure and that control. The type of this parameter, therefore, is `[(String, Number)]`.

3. `completion_map`:

   A map containing the association of each control identifier and its current completion. Its type is `Map<String, Number>`.

4. `min_compl`:



(a) Content of this record

(b) Relations of this record

Figure 4.4: Surrealist web app. Detail of the data in a table record

A number representing the minimum amount of completion a control shall reach in order to be deemed complete. The type of this parameter is `Number` and its default value is 100.

Looking at its code, it is possible to see how it first computes the increase in completion that the measure would bring between line 6 and line 10, adding the contribution that the measure gives to a control only if it has not been fully satisfied yet -Line 7-. Then, it computes the final cost according to Equation (2.1) and returns it.

Ranking Generation Algorithm The pseudo-code of the ranking generation algorithm is reported in Algorithm 2. Its inputs are the following:

1. `C`:

   The list of the filtered controls. Its elements are of type `Control` -see Section 4.1.1.1-.

2. `M`:

   The list of the measures present in the database. Each of its elements is of type `Measure` -see Section 4.1.1.2-.

3. `coverage_map`:

   A map associating the identifier of each measure to a list of tuples. Each tuple contains as its first element the identifier of a control that the measure satisfies and as its second element the coverage value -see Section 2.5.3.1- relative to that measure and that control. The type of this parameter, therefore, is `Map<String, [(String, Number)]>`.

Algorithm 1 Compute measure cost

```
1: function COMPUTE_COST(m, controls_covered, completion_map, min_compl)
2:     if m.progress = 100 then
3:         return 0
4:     else
5:         completion_increase ← 0
6:         for (control_id, cov) in controls_covered do
7:             if completion_map[control_id] < min_compl then
8:                 completion_increase ← completion_increase + cov
9:             end if
10:        end for
11:        if completion_increase > 0 then
12:            cost ← m.effort ÷ completion_increase
13:        else
14:            cost ← +∞
15:        end if
16:        return cost
17:    end if
18: end function
```

4. `completion_map`:

   A map associating the identifier of each control to its current completion, as defined in Section 2.5.3.1. For each of the controls in C, this map contains the associated entry -even if the completion of said control is currently zero-.

5. `min_compl`:

   A number specifying the minimum amount of completion a control should have in order to be considered completed. Its type is a `Number`.

The algorithm first initializes a new ranking, represented by an empty array. Then, on line 3, we enter a while loop that progressively adds new measures to our ranking. In each iteration, to decide which measure to add, we iterate over all the measures that have not been added to the ranking yet, computing the cost of each one relatively to the controls that have not been fully satisfied to this point. After having found the measure with the minimum cost, we (i) add it to the ranking -line 16-, (ii) remove it from the list of measures -line 17-, and (iii) update the completion of all the controls it satisfies, removing the ones that we have completed from the controls list -line 19 to line 26-. When we have exhausted the pool of controls or measures -or when none of the remaining measures can grant us additional completion-, we exit the while loop. At this point, on line 30, we build a list containing the identifiers of all the controls that have been fully satisfied and return it as a tuple alongside the ranking on line 37.

We decided to return both the ranking and the control identifiers -and persist them to the database- so that if the measures are associated with additional, possibly unrelated, controls after the ranking has been created, these extra controls do not clutter the view of this ranking: the dashboard -see Section 4.2.3.1- already shows a view in which the user can see all the controls and measures.

4.1.4 Backend endpoints

Having described the logic to communicate with the database and create a new ranking, we now focus on making the functionalities of the backend accessible to external clients through a REST API.

The API exposes its endpoints under the route `/api/v1` to allow the clients to request a specific version of the API when they interact with it. The endpoints are organized according to their role in five categories: controls, measures, tags, rankings and auth. A graphical representation of this high-level endpoint structure can be found in Figure 4.5. In the following sections, we provide a detailed description of the endpoints and their role. Unless otherwise specified, the endpoints do not return

Algorithm 2 Generate new ranking

```
1: function CREATE_RANKING(C, M, coverage_map, completion_map, min_compl)
2:     ranking ← [ ]
3:     while C ≠ ∅ and M ≠ ∅ do
4:         min_cost ← +∞
5:         best ← null
6:         for m in M do
7:             controls_covered ← coverage_map[m.id] or [ ]
8:             incomplete_covered ← C ∩ controls_covered
9:             cost ← COMPUTE_COST(m controls_covered, completion_map, min_compl)
10:            if |incomplete_covered| > 0 and cost < min_cost then
11:                best ← m
12:                min_cost ← cost
13:            end if
14:        end for
15:        if best is not null then
16:            ranking.push(best)
17:            M.remove(best.id)
18:            controls_covered ← coverage_map[m.id] or [ ]
19:            for (c_id, coverage) in controls_covered do
20:                coverage_increase ← coverage * (100 − best.progress)/100
21:                new_compl ← completion_map[c_id] + coverage_increase
22:                completion_map[c_id] ← new_compl
23:                if new_compl ≥ min_compl then
24:                    C.remove(c_id)
25:                end if
26:            end for
27:        else
28:            break
29:        end if
30:    end while
31:    completed_controls ← [ ]
32:    for (c_id, cov) in completion_map do
33:        if cov ≥ min_compl then
34:            completed_controls.append(c_id)
35:        end if
36:    end for
37:    return (ranking, completed_controls)
38: end function
```

any data in their response but only signal the successful or failed request through their HTTP status code. For the sake of brevity, we omitted the request guard present on each of these endpoints -except for the one to register and create a new token under `auth`- that ensure they can only be called by a user with appropriate permissions. The system we use to enforce RBAC is detailed in Section 4.1.5 and the roles are listed in Section 4.1.5.5.

4.1.4.1 Controls

This route -`/api/v1/controls`- provides 8 endpoints. They are represented graphically in Figure 4.6.

1. GET `/`

   This endpoint returns all the controls present in the database as a JSON object containing an array of `Control` objects.

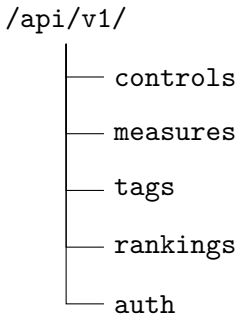


Figure 4.5: Backend endpoints structure

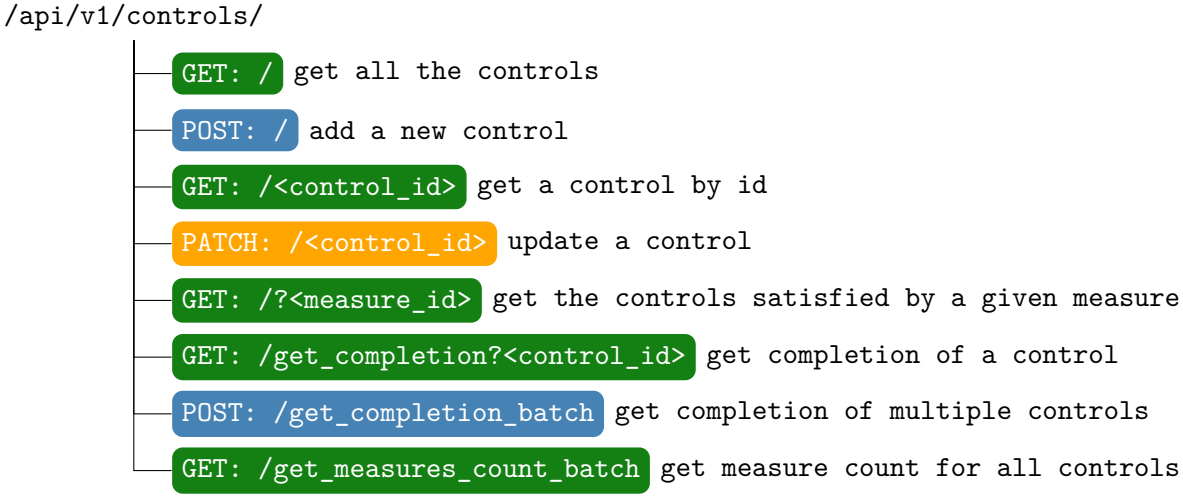


Figure 4.6: Backend control endpoints

2. POST `/`

   This endpoint is responsible for creating a new control. It accepts a POST request with a payload encoded as form data[12] that respects the following structure:

   - **title**: a string of length comprised between 3 and 30, representing the title of the new control;
   - **description**: a string of length comprised between 3 and 300, representing the description of the new control.

3. GET `/<control_id>`

   Returns a `Control` object serialized as JSON if a control with the given identifier exists in the database. The identifier is specified using the `control_id` as the final component of the path of the request's URL.

4. GET `/?<measure_id>`

   Returns the array of controls satisfied by a given measure, whose identifier is specified in the query parameters of the request as the `measure_id` parameter.

5. PATCH `/<control_id>`

   This endpoint allows the user to update an existing control. The identifier of the control must be specified as the final part of the URL path. The payload of the request shall be encoded as form data and contain the fields detailed below. They are all optional. If a field is missing, the corresponding property of the control will remain unchanged.

[12] `https://developer.mozilla.org/en-US/docs/Web/API/FormData/Using_FormData_Objects`

- **title**: an optional string of length in the range $[3, 30]$ representing the new title of the control;
- **description**: an optional string of length in the range $[3, 30]$ representing the new description of the control.

6. GET `/get_completion?<control_id>`

   Returns a floating point number representing the completion of the control whose identifier is specified as the `control_id` query parameter. If the control does not exist, it returns zero.

7. POST `/get_completion_batch`

   Accepts a request containing as payload an array of control identifiers -of type `String`- encoded as JSON. It returns an array of floating point numbers with the same length of the one in the request containing the completion of each of the specified controls.

8. GET `/get_measures_count_batch`

   Accepts a request containing as payload an array of control identifiers -of type `String`- encoded as JSON. It returns an array of positive integers with the same length of the one in the request containing the number of measures associated to each of the specified controls.

4.1.4.2 Measures

The route `/api/v1/measures` provides 8 endpoints. A graphical representation is provided in Figure 4.7.

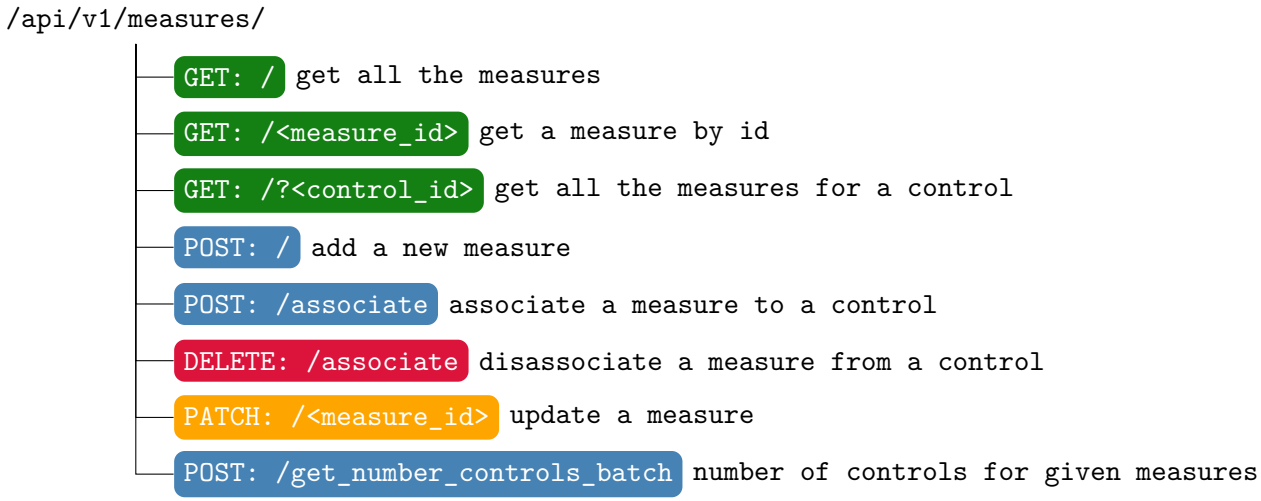


Figure 4.7: Backend measures endpoints

1. GET `/`

   This endpoint returns all the measures present in the database. The response is encoded as a JSON array where each element is of type `Measure`.

2. POST `/`

   The user can create a new measure through this endpoint. It accepts a request with a form-encoded payload with the following attributes:

   - **title**: a string of length comprised between 3 and 30, representing the title of the new measure;
   - **description**: a string of length comprised between 3 and 300, representing the description of the new measure;

- **effort**: a number between 0 and 256 -limited by the fact that we store it in a `u8`- representing an estimate on the difficulty of implementing the measure across the company -see Section 4.1.1.2-. This field is optional. If absent, the fallback value is 1.;

3. GET `/<measure_id>`

   This endpoint returns a specific measure from the database. The requester specifies the identifier of the measure it wants to retrieve using the `measure_id` a the final component of the URL path.

4. GET `/?<control_id>`

   Through this endpoint, the user can ask for all the measures that satisfy a given control. They specify the control by including its identifier under the `control_id` query parameter. The response is a JSON array containing the `Measure` objects requested.

5. POST `/associate`

   This endpoint allows to associate a measure to a control, representing the fact that the measure satisfies -see Section 2.5.1.1- the control. The payload of the request must be encoded as form data and contain the following fields:

   - **control_id**: a string identifying the control;
   - **measure_id**: a string identifying the measure;
   - **coverage**: an integer in the range $[1, 100]$ representing the coverage associated to this measure for this control -see Section 2.5.3.1-.

   .

6. DELETE `/associate`

   This endpoint allows to disassociate a measure from a control. The payload of the request must be encoded as form data and contain the following fields:

   - **control_id**: a string identifying the control;
   - **measure_id**: a string identifying the measure;

   .

7. PATCH `/<measure_id>`

   This endpoint allows the user to update an existing measure. The identifier of the measure must be specified as the final part of the URL path. The payload of the request shall be encoded as form data and contain the fields detailed below. They are all optional. If a field is missing, the corresponding property of the measure will remain unchanged.

   - **title**: an optional string of length in the range $[3, 30]$ representing the new title of the measure;
   - **description**: an optional string of length in the range $[3, 30]$ representing the new description of the measure;
   - **effort**: an optional number in range $[0, 256]$ representing the new effort associated to the measure;
   - **progress**: an optional number in range $[0, 100]$ representing the new progress associated to the measure.

8. POST `/get_number_controls_batch`

   This endpoint accepts a request with a JSON payload that contains an array of strings. Each of these strings represents the identifier of a measure. The endpoint returns an array of equal length encoded as JSON where for each identifier in the request is present a non-negative integer representing the number of controls that are associated to the corresponding measure.

4.1.4.3 Tags

The route `/api/v1/tags` provides 8 endpoints which allow the clients to interact with the tags present in the program. A graphical representation of the endpoints is provided in Figure 4.8.

Figure 4.8: Backend tags endpoints

1. GET `/`

   This endpoint returns all the tags present in the database as a JSON array of `Tag` objects.

2. POST `/`

   This endpoint allows to create a new tag. It accepts a request with a payload encoded as form data, with the following fields:

   - **name**: a string representing the name of the new tag;
   - **color_hex**: a hexadecimal string representing the color code of the new tag.

3. GET `/?<measure_id>`

   Returns all the tags relative to a specified measure. The identifier of the measure is included as the `measure_id` query parameter. The response is a JSON array of `Tag` objects.

4. GET `/?<control_id>`

   Returns all the tags relative to a specified control. The identifier of the control is included as the `control_id` query parameter. The response is a JSON array of `Tag` objects.

5. POST `/tag_control`

   Adds a tag to a control. It accepts a request with the following payload, encoded as form data:

   - **control_id**: a String containing the identifier of the control to tag;
   - **tag_id**: a string containing the identifier of the tag to associate to the control.

6. DELETE `/tag_control`

   Removes a tag to a control. It accepts a request with the following payload, encoded as form data:

   - **control_id**: a String containing the identifier of the control to un-tag;
   - **tag_id**: a string containing the identifier of the tag we want to remove from the control.

7. POST `/get_tags_batch`

   This endpoint allows the client to retrieve the tags for multiple controls at once. The request shall contain a payload with an array of control identifiers encoded as JSON. The response is a JSON array where for each of the control identifiers, the server provides an array containing the `Tag` objects associated to such control.

8. GET `/get_measure_tag_ids_batch`

   This endpoint allows the client to retrieve the tag identifiers for all the measures at once. The response is a JSON map where for each of the measure identifiers present in the database, there is an array containing the identifiers of all the tags that are associated to that measure -coherently with the definition of tagging in Section 3.3.1.3-.

4.1.4.4 Rankings

The route `/api/v1/rankings` provides 3 endpoints which allow the clients to interact with the rankings present in the program. A graphical representation of the endpoints is provided in Figure 4.9.

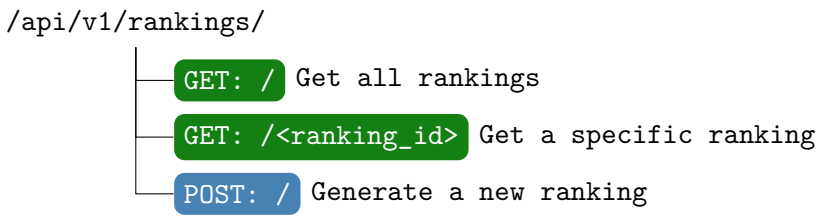


Figure 4.9: Backend rankings endpoints

1. GET `/`

   This endpoint returns all the rankings present on the server as a JSON array of `Ranking` objects.

2. GET `/<ranking_id>`

   This endpoint returns a specific ranking present on the server as a `Ranking` object serialized to JSON. The identifier of the ranking is specified as the final part of the URL path.

3. POST `/`

   This endpoint allows the user to create a new ranking. It accepts a request where the payload is encoded as form data and has the following fields:

   - **name**: a string representing the title for the ranking. Its length shall be between 1 and 100;
   - **minimum_coverage**: an integer representing the minimum completion that a control must have in order to be considered complete -see Section 2.5.3.1-.
   - **filter_tags**: an optional array of tag identifiers represented as strings. If this field is not null, the controls will be filtered before proceeding with generating the ranking -see Section 4.1.3-;
   - **all_tags**: a boolean flag that is considered only if the filter_tags array is not null. If this flag is set to `true`, the ranking will proceed only with the controls that have been tagged with all the tags specified in filter_tags. Otherwise -if all_tags is set to `false`- the ranking will include in the creation process all the controls that have been tagged by at least one of the tags specified in the filter_tags field. This field is optional and its fallback value is `false`.

4.1.4.5 Auth

We implemented a simple authentication and authorization mechanism for our service, which we cover in Section 4.1.5.

The route `/api/v1/auth` provides 4 the endpoints which are necessary to perform the authentication procedure. A graphical representation of the endpoints is provided in Figure 4.10.

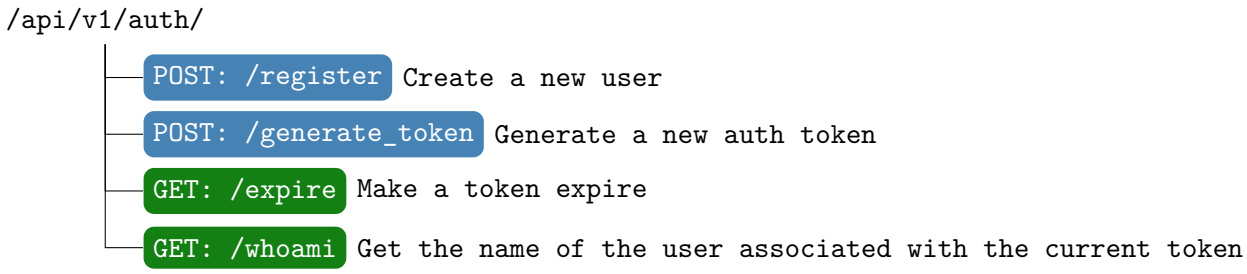


Figure 4.10: Backend auth endpoints

1. POST: `/register`

   This endpoint allows to create a new user on the service. It accepts a request having as payload the following data in form encoding:

   - **username**: a string representing username of the user we want to create. In order to be valid, it must comply with the requirements reported in Section 4.1.1.5;
   - **password**: a string long at least 8 characters and at most 256 representing the plaintext password of the user.

2. POST: `/generate_token`

   The client can call this endpoint to authenticate as a user and obtain as response a bearer token for that user represented as a string, as explained in Section 4.1.5. A valid request shall have a payload containing the following form-encoded data:

   - **username**: same as in the `/register` endpoint
   - **password**: same as in the `/register` endpoint

3. GET: `/expire`

   A request to this endpoint causes the bearer token associated with it in the `Authorization` HTTP header to expire, ensuring it cannot be used for authorizing subsequent requests.

4. GET: `/whoami`

   This endpoint returns a response containing the name of the user corresponding to the token provided in the request `Authorization` HTTP header if one exists, otherwise it returns a response with the status code 401: unauthorized.

4.1.5 Authentication and access control

This section details how we implemented a basic authentication and authorization system. The goal was to add a minimum layer of security to the server through a basic RBAC system [24].

4.1.5.1 User accounts

The first step is user authentication. Each user has their own account, to which we can associate the RBAC roles.

A user account, as shown in Section 4.1.1.5, has as its only properties the name, password hash and salt and the array of roles that the user has.

Account creation To create the account, an anonymous user has to supply an unregistered username to the `/api/v1/auth/register` endpoint -see Section 4.1.4.5- along with a password. The password is immediately hashed along with a salt that is randomly generated by the server to obtain the password hash, which is then stored in the user account.

We employ an established password hashing algorithm, argon2id [8]. To adopt it in the backend, since it is written in Rust, we use the crate `argon2`[13]. The crate provides all the necessary functionality to generate a secure password hash, including the generation of the salt using the entropy provided by the system. The resulting hash is stored as a string in a format known as PHC String Format, which includes the hashed password, the salt, the algorithm and the parameters in a single string [55].

Login When a user wants to authenticate, they send a request to the authentication endpoint `/api/v1/auth/generate_token`, including their username and password in the body of the request -see Section 4.1.4.5-. On the backend, the `argon2` crate provides the functionality to securely verify if the provided password matches the hash in PHC String Format that we stored in the database for that username during the registration. If they match, a new token is generated and sent to the user.

4.1.5.2 Tokens lifecycle

Our application uses bearer tokens to authorize the user's requests to protected endpoints. The model representing this data in the backend, `AuthToken`, is described in Section 4.1.1.6.

When a new `AuthToken` is generated, the token string is initialized with a random string of 64 hexadecimal digits -using a secure random number generator provided by the crate `rand_chacha`[14], seeded with the entropy provided by the operating system at runtime-.

The expiration date of an `AuthToken` is set at seven days after its creation. If it was necessary to revoke a token before this expiry date, it can be done by calling the endpoint `/api/v1/auth/expire` -see Section 4.1.4.5-.

4.1.5.3 AuthToken request guard

The rocket framework offers a powerful feature to ensure that a request handler is not called with erroneous data: request guards. A request guard[15] is a type, any type that implements the `FromRequest` trait. When a request handler wants to use a request guard for a type `T`, it simply includes a parameter of type `T` between its parameters, like in Listing 2, where the value for the user parameter will be obtained automatically from the request using the method defined by the `FromRequest` trait if `User` implements the `FromRequest` trait -else, it will not compile-. A request handler can also specify as one of its parameters an optional request guard by wrapping it in an `Option<T>`. In that case, if `T` cannot be derived from the request, the handler will still be invoked, but that parameter will have a value of `None`.

The rocket framework ensures that if a request guard cannot be obtained from a request, the handler is never called and an appropriate error message is returned. This enables the programmer to use the request guards to check the incoming requests according to their requirements. As the official documentation describes it[15], "a request guard is a type that represents an arbitrary validation policy".

Listing 2 Example of how to use a request guard for a type `User` in a rocket handler

```
1 #[get("/example")]
2 fn my_handler(user: User) {
3     /* ... */
4 }
```

[13] https://docs.rs/argon2/0.5.2/argon2/index.html
[14] https://docs.rs/rand_chacha/0.3.1/rand_chacha/index.html
[15] https://rocket.rs/v0.5-rc/guide/requests/#request-guards

In our case, we created a custom request guard for `AuthToken` that extracts the token from the `Authorization` HTTP header of the request and checks its validity with the database. If the token does not exist or is expired, the server will not process the request further and reply with a status code of 401: unauthorized. The code for this request guard is reported in Listing 3. There, we can see on line 14 how it is possible for the request guard to access the database by virtue of it being include in the state of the server, which is managed by rocket -we initialized the corresponding variable in Section 4.1.2.2-.

Listing 3 Rocket request guard implementation for `AuthToken`

```
1  #[rocket::async_trait]
2  impl<'r> FromRequest<'r> for AuthToken {
3      type Error = AuthTokenError;
4
5      async fn from_request(req: &'r Request<'_>) -> Outcome<Self, Self::Error> {
6          match req.headers().get_one("authorization") {
7              None => {
8                  println!("The Authorization HTTP header is missing");
9                  Outcome::Failure(401)
10             }
11             Some(authz_header) => {
12                 let db = req.guard::<&State<Surreal<Client>>>().await.unwrap();
13                 match authz_header.strip_prefix("Bearer ") {
14                     Some(user_supplied_token) => {
15                         match get_token(user_supplied_token, db).await {
16                             Ok(Some(token)) => {
17                                 if token.is_expired() {
18                                     Outcome::Failure(401)
19                                 } else {
20                                     println!("A valid token is present in the
                                        ↪  request");
21                                     Outcome::Success(token)
22                                 }
23                             }
24                             Ok(None) => Outcome::Failure(401)
25                             Err(_err) => Outcome::Failure(500)
26                         }
27                     }
28                     None => Outcome::Failure(401)
29                 }
30             }
31         }
32     }
33 }
```

Having implemented this request guard is enough to protect some endpoints ensuring only users with a valid token can access them. We did, however, implement an additional request guard to obtain the `User` from a request. Its code is shown in Listing 4, and it is possible to see how it uses -on line 6- the `AuthToken` guard we just defined to try extracting a valid token from the request and then lookup the associated user.

With the `User` request guard in place, the rocket frameworks guarantees that every handler that accepts a parameter of type `User` -e.g., Listing 2- will be invoked only if a valid user could be extracted from the request.

Listing 4 Rocket request guard implementation for `User`

```
#[rocket::async_trait]
impl<'r> FromRequest<'r> for User {
    type Error = AuthTokenError;

    async fn from_request(req: &'r Request<'_>) -> Outcome<Self, Self::Error> {
        let token_from_req = req.guard::<AuthToken>().await;
        if token_from_req.is_success() {
            let token = token_from_req.expect("Can get token from req guard");
            let db = req.guard::<&State<Surreal<Client>>>().await.expect("Can get
            ↪  db from req guard");
            match get_user(&token.username, db).await {
                Ok(Some(user)) => Outcome::Success(user),
                Ok(None) => Outcome::Failure(401)
                Err(err) => Outcome::Failure(500)
            }
        } else {
            Outcome::Failure(token_from_req.failed())
        }
    }
}
```

4.1.5.4 Roles

Having the `User` request guard in place, we proceeded implementing Role-Based Access Control (RBAC) [24]. As anticipated in the `User` model -Section 4.1.1.5-, there is a predefined list of the roles that a user can have, represented as a Rust `enum`.

To implement RBAC, we had to ensure that a request handler would be invoked only if the caller has all the roles required to call that endpoint. To do so, we decided to create a request guard tasked with extracting the current `User` using its request guard and return a successful result if and only if such user has in its roles all the ones contained in a predefined list -one list for each endpoint-. In order to create such a request guard programmatically, we wrote a Rust macro that takes as arguments the name of the request guard struct we want to generate and a set of roles representing the ones required to make that guard pass. The code of the macro is shown in Listing 5. There, we can see the check on the roles being performed on line 19, where it checks if the set of required roles r_{req} is an improper subset of the user's roles r_{user} ($r_{req} \subseteq r_{user}$).

With the `generate_endpoint_roles` macro defined, Listing 6 shows how it can be used to generate a new request guard on line 1 and how it can be used in an endpoint on line 3.

4.1.5.5 Roles management

The following list details all the roles currently defined in the program, with a brief description of each one. If needed, it is possible to define others by changing the source code in the `Role` enum.

- `CreateControls` Can add new controls
- `GetControls` Can see existing controls
- `EditControls` Can modify existing controls
- `CreateTags` Can add new tags
- `GetTags` Can see existing tags
- `EditTags` Can modify existing tags
- `CreateMeasures` Can add new measures
- `GetMeasures` Can see existing measures
- `EditMeasures` Can modify existing measures
- `AssociateMeasures` Can associate an existing measure to an existing control

- `DisassociateMeasures` Can remove an association between a measure and a control
- `CreateRanking` Can generate a new ranking
- `GetRankings` Can see existing rankings

A new user on the database only gets the roles allowing them to see the data present in the program: `GetControls`, `GetTags`, `GetMeasures`, and `GetRankings`. The only way to change the roles assigned to a user is by interacting directly with the database since the backend does not offer the functionality to manage the users and their role at the moment. This is an inconvenient solution, but it ensures that only the administrator of the service can change the roles: they should be the only one with the credentials to access the database through a client like surrealist -see Section 4.1.2.4-. We acknowledge this to be a sub-optimal solution, and justify its adoption only in light of the fact that adding a more comprehensive functionality here would have hindered our ability to focus on the core components of the program due to time constraints. We hope that a future version of the program will have a more proper user management system enabling the administrators to manage the users and the roles directly from the program's web UI.

4.2 Frontend

This section details the frontend, provides is the web client end-users adopt to communicate with the kampas backend -covered in Section 4.1-. It is written in TypeScript[16], using the web framework SvelteKit[17].

To distinguish the backend of this service from the backend of Section 4.1, we refer to the former as "backend" and the latter as "server" in this section.

4.2.1 Authentication

Since the majority of the endpoints provided by the kampas server require an account to access -see Section 4.1.5-, authentication is the first step.

We provide a page to register and one to login. In each of these pages, the user can see a form where they can enter the username and password for the account. When they submit the form, the credentials are sent to the backend.

There, we perform an initial validation of the data we received to ensure it complies with the requirements for a username and password of the kampas server. If they are acceptable, they are forwarded to the kampas server. If the credentials are correct, the backend obtains a bearer token as response, which gets stored in the user's session. The sessions on the backend are managed in redis in-memory and periodically persisted to disk. Every time the backend needs to make an authenticated request to the kampas server for a user, the bearer token is extracted from the session and included in such request.

Each session is bound to its respective client using a cookie containing the identifier of the session. The cookie has the `httpOnly` flag set to `true`, `sameSite` as `strict`, a max age of seven days and, if we are running in with the `NODE_ENV` environment variable set to "production", the secure flag is also set.

We wrote a custom SvelteKit hook[18] that automatically redirects any client which tries to interact with our service unauthenticated to the login page.

4.2.2 Models

Representing the data in the frontend through models posed no issues: as reported in Section 4.1.1.1, a Rust crate allows to automatically convert the models on the kampas server to their equivalent TypeScript type definitions. Ensuring consistency between the two representations. Listing 7 shows as an example the model for `Measure` that was generated programmatically using the `ts-rs` crate. In the listing, the Rust code is on the left and the equivalent TypeScript type declaration that was generated for it is shown on the right.

[16] `https://www.typescriptlang.org`
[17] `https://kit.svelte.dev`
[18] `https://kit.svelte.dev/docs/hooks`

4.2.3 Routes

SvelteKit offers a very pragmatic method for defining the routes of a web application[19]. Starting from the `src/routes` directory, every file whose name starts with the '+' characters is considered a route file. The most common is '+page.svelte', which allows to define the svelte page that will be shown when the user visits that path. The path of a route is given by its place in the directory structure of the project. `src/routes/+page.svelte` is the page that is shown when the user opens the app, `src/routes/controls/+page.svelte` is the page that is shown when the user navigates to the route `/controls` in the frontend.

SvelteKit also supports dynamic components in the route. If, for example, we want to create a page for a specific control, we can create a file on the following path: `src/routes/controls/[id]/+page.svelte`. When the user visits a route like `/controls/nist_id_1` on the frontend, the page we just defined will be rendered -and inside the page the code will have access to a value named 'id' with the value 'nist_id_1'-.

Figure 4.6 represents graphically the directory structure of the files we just mentioned. We can notice that, besides the page files, we also have some `+page.server.ts` files. These files contain TypeScript code that runs on the server whenever we request the relative page. This code can fetch any data the page needs and make it available to it during the server-side rendering.

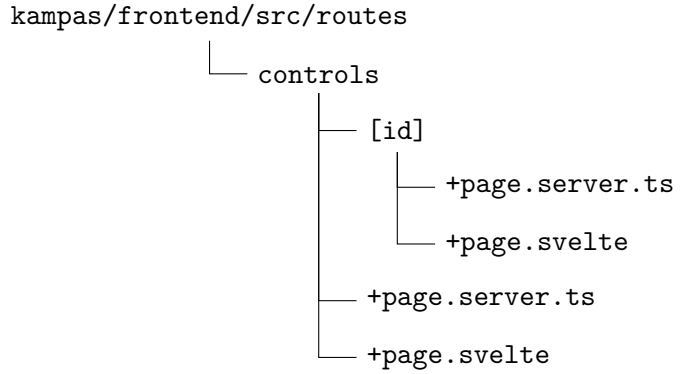


Figure 4.11: Frontend controls routes

4.2.3.1 Homepage dashboard

The most prominent feature of the frontend is surely the dashboard -visible in the screenshot in Figure 4.12-. This page aggregates and displays most of the information available on the server.

The dashboard is divided in multiple sections: first we have the statistics for the controls, displaying the total number of controls present in the database, the medium and median completion of the controls and a bar chart. The bar chart presents the distribution of how complete the controls are by showing on the horizontal axis the completion percentage and on the vertical axis the number of controls that have said completion percentage.

Then, we have an equivalent section showing the statistics for the measures present on the kampas server, followed by a table which lists all the measures present on the server sorted by title. By clicking on the table header of each column, it is possible to change the ordering from ascending to descending and vice-versa and to change the column we are sorting on. Therefore, this table also enables the user to see which measures have the greatest or smallest progress or number of associated controls.

Subsequently, a second table lists all the controls present in the database, offering the same interactive sorting functionality as the first one.

Finally, on the top of the page, there is a button with the text "Add tag". Through this button, the user can choose to restrict the controls, measures and statistics shown on the page to only the ones that are tagged with all the tags added above. This makes it possible to assess the performance of the company both at a holistic level, when no tags are specified, and for specific topics -provided the relative controls are tagged adequately-.

[19] Official documentation available at `https://kit.svelte.dev/docs/routing`

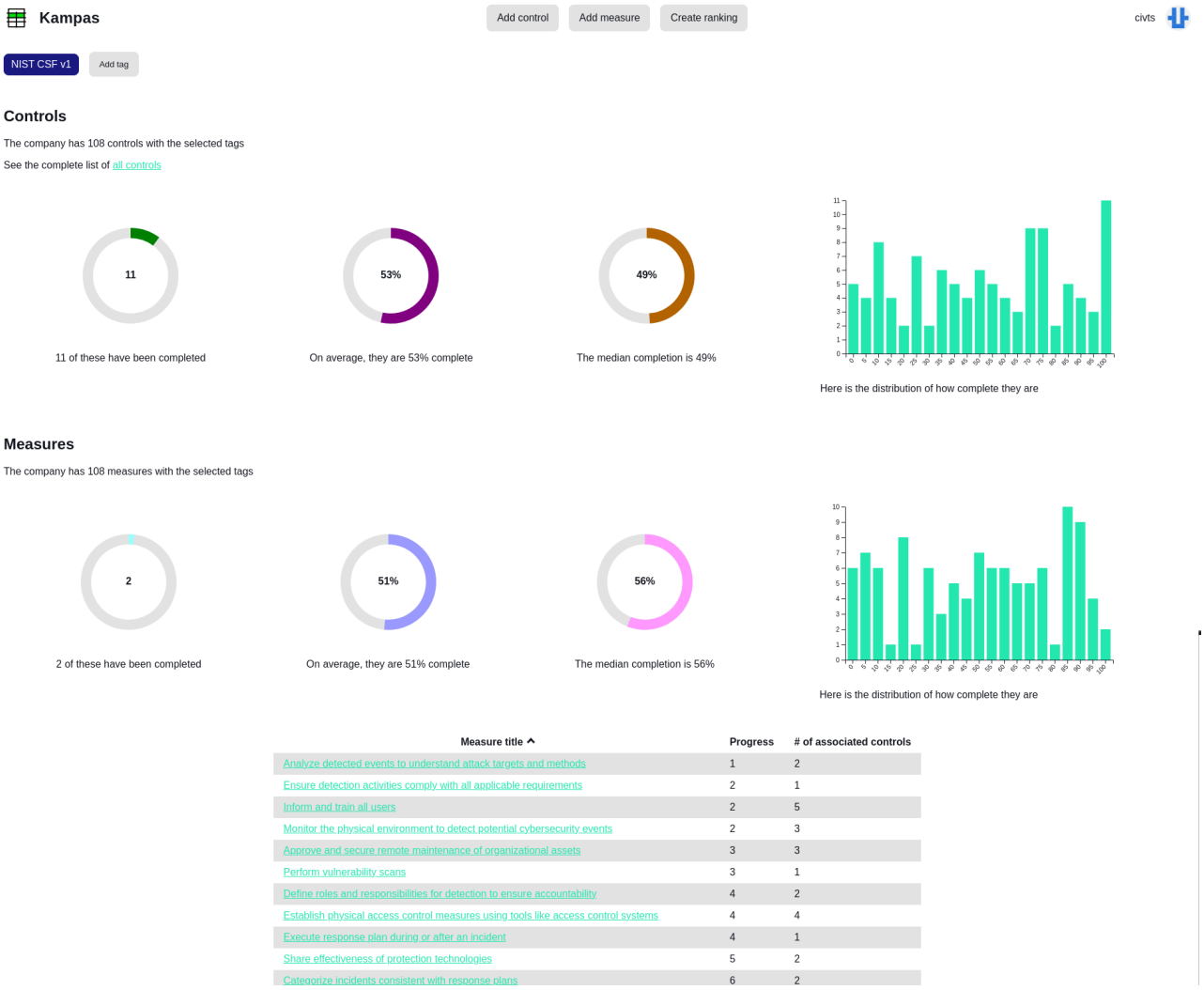


Figure 4.12: Dashboard of the frontend

4.2.3.2 Controls list

Another important page is the one showing the control list -Figure 4.13-. The user can reach it by going to the route `/controls`, or by clicking the dedicated link in the homepage.

The page with the controls list enumerates all the controls present in the kampas server and presents every one of them complete with its description and tags. The controls are sorted alphabetically and clicking one of them allows the user to go to its dedicated page.

In this view, it is possible to quickly add a new tag to an existing control by simply clicking the "Add tag" button and selecting the tag from the overlay that appears.

Through the "Add tag" on the top of the page, instead, it is possible to filter the controls shown on the page to display only the ones that have all the specified tags. It is possible to add multiple tags to this filtering set. To remove a tag from the tags we are filtering by, it is sufficient to hover with the mouse over the tag we want to remove to show a button with an X over it -see Figure 4.14-. Clicking it removes the tag from the set of currently active tags.

It is also possible to quickly un-tag a control. The interaction is the same as in Figure 4.14, provided that the user has the permissions necessary to remove a tag from a control.

4.2.3.3 New ranking

The user can access this route by clicking on the "Create ranking" button on the navigation bar or browsing to `/new_ranking`. A screenshot the page is shown in Figure 4.15.

There, they can specify tha name for the ranking they wish to create, specify the minimum

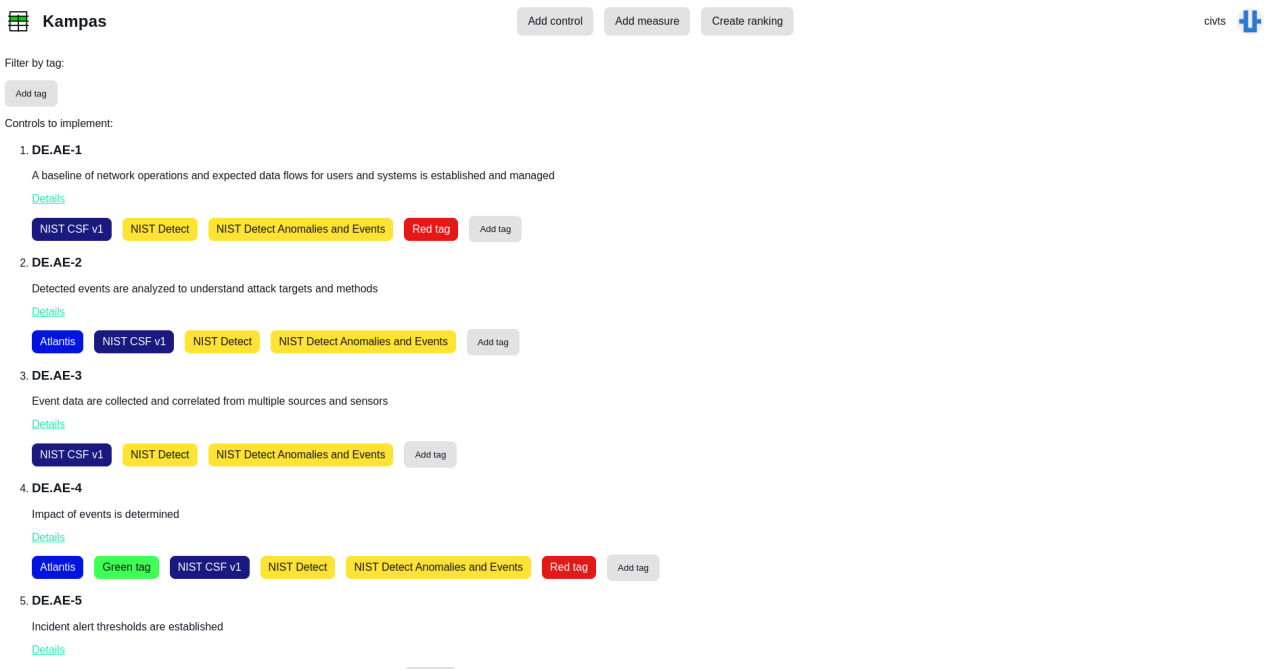


Figure 4.13: List of controls in the frontend

Figure 4.14: Micro-interaction to remove a tag in the frontend

completion a control needs to reach in ordered to be considered completed -see Section 2.5.3.1-, and filter the controls to include for the ranking creation by tag -choosing to keep the controls that have either all the specified tags or at least one of them through a dedicated toggle button-.

Once they are satisfied with the parameters they chose, they can submit the form to start the ranking generation. Once that is complete, they are automatically redirected to the detail page of the new ranking.

4.2.3.4 Ranking detail page

This page is located under the route `/rankings/<id>`, where `<id>` represents the identifier of the ranking we want to view the details of. As it is visible in Figure 4.16, the ranking details page shows all the information that is present on the kampas backend about a specific ranking.

The first portion of content reports the parameters that were used in the creation of the ranking -i.e., the ranking algorithm and the minimum_coverage value-, the username of who created the ranking and the creation time.

Underneath this section, the interface displays the “Add tag” button that the user can utilize -in the same way they do in the other pages- to filter according to the specified tags the controls and measures shown in the subsequent sections.

Next, the page includes a list of all the measures and the controls present in the ranking. The measures are displayed in order so that the ones that bring the maximum increase in completion of

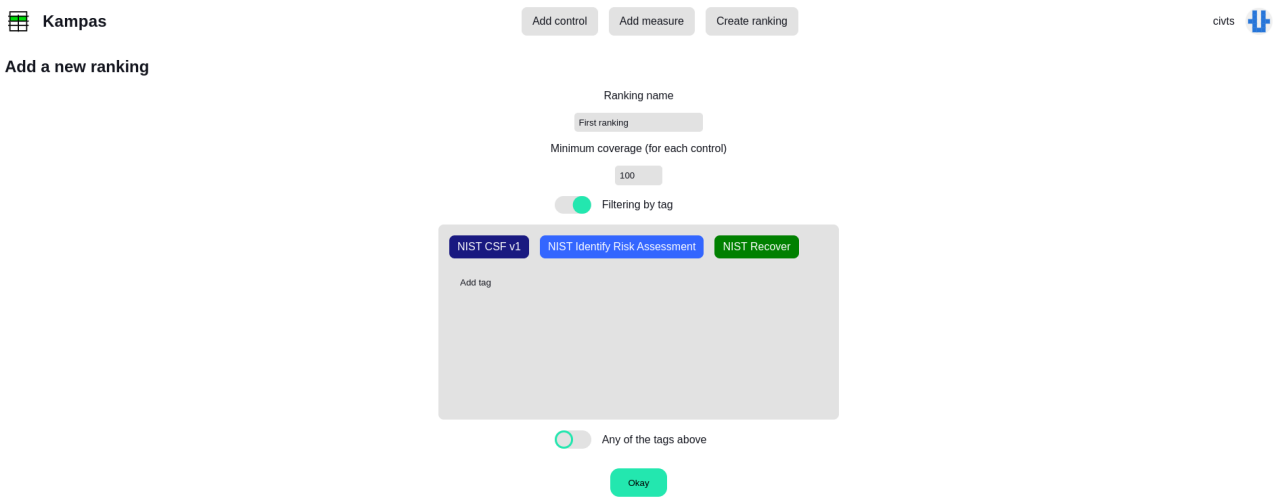


Figure 4.15: Ranking creation in the frontend

the controls at the minimum cost are shown first -see Section 2.5.2.1-. The progress of each of the measures is shown besides their title, and the page also reports the aggregate medium progress of the measures that are being displayed.

It is possible to see that, in the case of Figure 4.16, we are filtering by tag using the tag named "NIST Identify Risk Assessment". Therefore, the control list shows only the controls that possess that tag and the measures section lists only the measures that satisfy at least one control tagged with the selected tag -including any control that is tagged with the tag "NIST Identify Risk Assessment" but was not included in this ranking-.

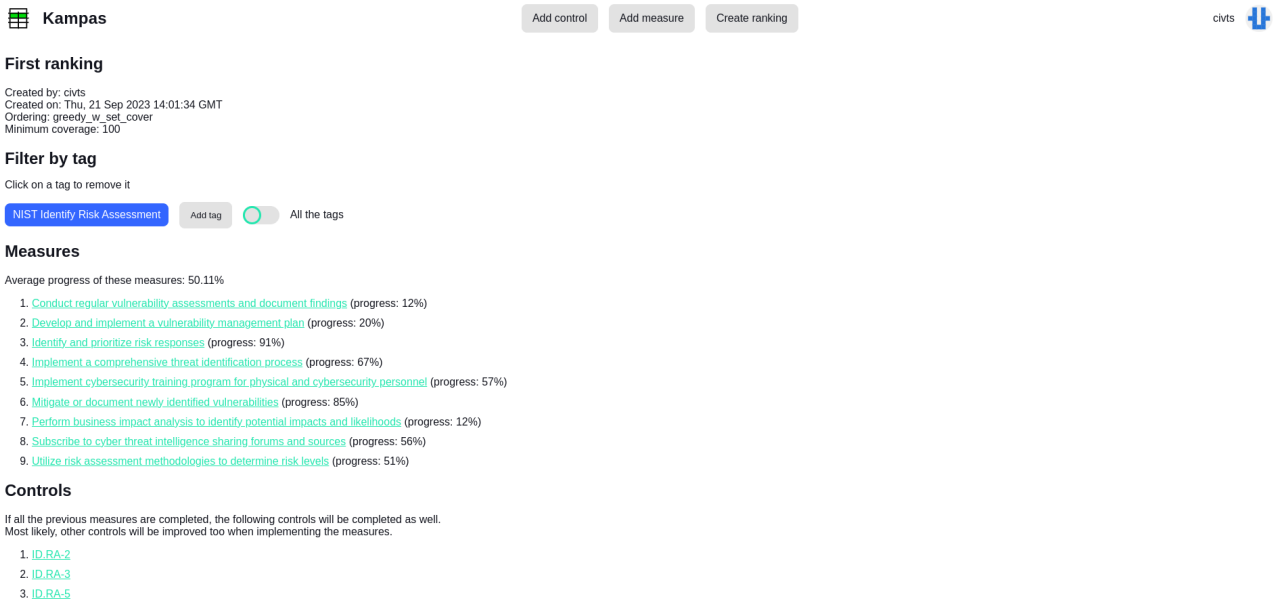


Figure 4.16: Ranking details page in the frontend, with tag filtering

4.2.3.5 Graph view

Another page worth of attention is the one available through the route `/graph`. By visiting this page, the user can see the dataset of controls and measures represented as a graph -as in Figure 4.17-.

The nodes of the graph are either a measure or a control. The measures are colored in blue, while the controls are colored in gray. The edges connect a control to all the measures that satisfy it.

This graph is rendered by a third-party library and is a force-directed graph drawing [30]. With this technique, the drawing can be imagined as an evolving physics simulation in which the nodes repel one-another and the edges bind some of them together. As the iterations advance in the simulation, the graph naturally tends to lay itself out in a stable configuration, where the attractive and repulsive forces are balances. This often results in an aesthetically pleasing representation which keeps the nodes relatively close to their nearest neighbors.

The idea behind the inclusion of this graphical view is to let the user visually explore the controls and the measure that are present in the database, perhaps identifying that some measures which appear close to a control but are not connected to it can in fact contribute to satisfying that control and therefore represent great candidates for new edges in the graph.

If the graph shows many components with few nodes in each one, perhaps this may indicate that it would be worth to check if the associated measure really only apply to these controls.

Since the notion of proximity is important for the scenario we described, we decided to represent the graph not in two but in three dimensions using the library `3d-force-graph`[20] to allow each node an additional degree of freedom in its movement.

The library we used draws the graph using Three.js in an HTML canvas, and the user can interact with the graph using the mouse for moving the view, zooming, panning and dragging the vertices. If the user hovers on a node, its title is shown. If a node is clicked its detail page opens in a new tab. It is also possible to draw the edges with different thicknesses to represent the different coverage values that each measure has with respect to its associated controls -see Section 2.5.3.1 for the notion of coverage-.

Figure 4.17: Graph page in the frontend

4.2.3.6 Other pages

The frontend also provides the following minor pages:

1. New control page:

   This page is reachable on the `/new_control` route and contains a form that allows the user to create a new control by specifying its title and description.

[20] `https://github.com/vasturiano/3d-force-graph`

2. New measure page:

   This page is available on the route `/new_measure` and provides a form which allows the user to create a new measure by specifying its title, description and effort.

3. Tag management page:

   This page is available on the route `/tags`. Here the user can see the existing tags and create new ones.

4. Ranking list page:

   This page is available on the route `/rankings`. It provides a list of all the rankings that have been generated on the server ordered with the most recent ones first.

5. Control details page:

   This page is available on the route `/controls/<control_id>`, where `<control_id>` is the identifier of the control. It displays the details of the control: its title, description, tags, and completion. It also shows a list of all the measures that are associated with this control. In this page, it is possible to edit the control's title and description.

6. Measure details page:

   This page is available on the route `/measures/<measure_id>`, where `<measure_id>` is the identifier of the measure. It displays the details of the measure: its title, description, tags, effort and progress. It also shows a list of all the controls that are associated with this measure. In this page, it is possible to update the measure's title, description, effort and progress.

Listing 5 Macro to generate the required roles request guards

```
#[macro_export]
macro_rules! generate_endpoint_roles {
    ($struct_name:ident, { $($role:expr),* }) => {
        pub(crate) struct $struct_name {}

        #[rocket::async_trait]
        impl<'r> FromRequest<'r> for $struct_name {
            type Error = &'static str;

            async fn from_request(req: &'r Request<'_>) -> Outcome<Self,
            ↪  Self::Error> {
                let required_roles = vec![$($role),*];

                let user_from_req = req.guard::<User>().await;
                match user_from_req {
                    Outcome::Success(user) => {
                        let required_roles_set: HashSet<_> =
                        ↪  required_roles.into_iter().collect();
                        let user_roles_set: HashSet<_> =
                        ↪  user.roles.into_iter().collect();

                        let has_all_roles =
                        ↪  required_roles_set.is_subset(&user_roles_set);

                        if has_all_roles {
                            Outcome::Success(())
                        } else {
                            Outcome::Failure(403)
                        }
                    }
                    _ => Outcome::Failure(401),
                }
            }
        }
    };
}
```

Listing 6 Example usage of the `generate_endpoint_roles` macro

```
generate_endpoint_roles!(GetControlsMeasuresRole, { Role::GetMeasures,
↪  Role::GetControls });
#[get("/get_measure_control_association_batch")]
pub(crate) async fn get_measure_control_association_batch(
    _required_roles: GetControlsMeasuresRole,
    db: &State<Surreal<Client>>,
) -> status::Custom<String> {
    /* ... */
}
```

Listing 7 Measure model in TypeScript generated from Rust

```rust
// Rust model in kampas backend
pub(crate) struct Measure {
    pub(crate) identifier: String,
    pub(crate) title: String,
    pub(crate) description: String,
    pub(crate) progress: u8,
    pub(crate) effort: u8,
}
```

```typescript
// TypeScript generated model
export interface Measure {
    identifier: string,
    title: string,
    description: string,
    progress: number,
    effort: number
}
```

5 Evaluation

In this chapter, we perform an initial assessment of our solution in the light of the requirements detailed in Section 3.1. The aim of this preliminary evaluation is to determine if the programs we produced in this thesis are capable of satisfying the needs of the users.

Whilst a proper assessment would require deploying the application in a realistic scenario -perhaps a medium to large size enterprise- and observing the performance of the system over a sustained period of time and in multiple use-cases [61, §8.4], this initial simulation allows us to verify some core functionalities of the software we produced for this thesis coherently with the time frame set for this project.

To perform this assessment, we decided to see what the process would look like if we were a company of approximately 500 people looking to implement the controls of the NIST *Cybersecurity Framework (CSF) v1.1* [49] and, subsequently, to be also in compliance with the CIS *Critical Security Controls v8* [12].

In these sections, unless specified differently, we interact directly with the frontend -see Section 4.2-.

5.1 NIST Cyber Security Framework

The objectives of this first phase of the test is to:

- add the controls of the NIST CSF to the system,
- add the measures associated to each control. Each measure shall have its own effort
- modify a measure to update the progress
- create a ranking of the controls
- modify a control to correct a hypothetical typo
- tag the controls
- filter by tag

5.1.1 Account creation

The first step is represented by creating a user on the system. This is mandatory given that, unless we are authenticated, we are redirected to the login page. We can easily create a user by navigating to the register page and providing a new username and password. We see that the registration fails if we provide an existing username.

Once we have authenticated, we are redirected to the homepage of the application, showing a dashboard in which we can see that we currently have no controls nor measures.

5.1.2 Adding the controls

At this point, we can start adding the controls from the framework by clicking on the "Add control" button in the navigation bar. And filling out the title and the description for the first control of the NIST CSF: "ID-AM.1".

When we submit the form, it is immediate to notice that the request did not succeed since we get an error message. This is expected as new users only get read permissions by default, not write. In order to provide the user with the permission necessary to create a new control, we need to assign them the corresponding role. As mentioned in Section 4.1.5.5, the only way to assign roles, as of now, is to do so manually in the database. This ensures that only the administrator of the service -which should be the only one able to access the database- can modify the user's roles. This limitation is

reported also as a possible future improvement in Section 6.2. Once the user has been assigned the required roles, they can successfully create a new control.

We also tested the functionality to create multiple controls simultaneously by uploading a CSV file containing in each row a new control's title and description. This worked as well, enabling us to quickly create a control in the database for each of the NIST CSF's controls.

5.1.3 Adding the measures

At this point, we proceeded in creating the measures. Since we are considering a fictitious company, we asked a Large Language Model -ChatGPT version 3.5- to generate the measures that an hypothetical company may take in order to satisfy the controls [18]. While this was fine for generating the measures for this demonstration, we would advise against proceeding similarly in a real scenario. Choosing the measure(s) necessary to implement even a single cyber security control requires a deep understanding of the company and a considerable level of knowledge and critical reasoning that we find unlikely to be matched by models like ChatGPT. These models -in their current form- do not appear to have a real understanding of the underlying semantics of the language and the requests they are given. In light of this, some researchers have labeled them as "stochastic parrots" [6].

To create the measures programmatically, we provided ChatGPT with the following prompt:

> I am going to send you a series of controls from the NIST CSF.
>
> For each of the NIST CSF controls I will give you, propose a way in which a fictitious financial company of 500 people may comply with it.
>
> Do not omit any of the original controls in your answer. Try to come up with plans that are as specific and actionable as possible. Do not bluntly rephrase what is asked from the control but think about what you could do to implement it in the company if you were its CISO.
>
> If, when describing the tools, you have in mind the name of a specific one that may be suitable, include it.
>
> Reply with a JSON array where every object is in this format:

```
{
    // The control we want to satisfy
    "control_title": "ID.AM-1",
    // How we satisfy it
    "plan_title": "Perform monitoring for unauthorized connections, devices,
    ↪  and software",
    "plan_description": "Integrate the Suricata and Snort intrusion detection
    ↪  systems to monitor for unauthorized personnel, connections, devices,
    ↪  and software within the organization's networks.",
    // How many days it will take to do this (realistically)
    "effort": "40"
}
```

In the subsequent messages, we sent a series of JSON arrays, each one containing some controls from the NIST CSF, structured like this one: `{"title": "ID.AM-4", "description": "External information systems are catalogued"}`.

The response contained a JSON array with exactly one measure for each control. We noticed that we could include around 12 controls in each message we sent. If we sent more than that, the response generation was at times automatically terminated before the response was complete since it got too long.

Having obtained a JSON file with all the measures, we proceeded to add them to the database. At first we used the web UI to add them individually, verifying that the component was functional. Then, we imported them in the database directly by running a batch of queries using Surrealist Section 4.1.2.4. Having a way to upload the measures in batch as we did with the controls would have been convenient in this case, so we include it as one of the possible future improvements.

5.1.4 Updating the progress

We then proceeded to update the progress of the measures, first from the web UI, then by interfacing directly with the database. We set the progress of each measure to a random number between 0 and 100, included.

5.1.5 Associating the measures

Next, we proceeded associating each measure to the control it was generated for. We initially did so using the web UI to test the functionality, confirming it worked. Then, we proceeded programmatically to speed the process up. We assign to each relation between a measure and its control a random coverage between 80 and 100 -since the measure was conceived explicitly for that control, we give it a high coverage-. Then, we assigned, randomly, one half of the existing measures to existing controls with a random coverage between 1 and 50.

5.1.6 Creating a ranking

At this point, we proceeded in creating a ranking through the web UI. We did not specify any tag to filter by -as we did not include any tag yet-. The ranking generation -with 108 controls, 108 measures and 162 associations between the measures and the controls- took approximately 180 milliseconds -on a Intel i7-8550U, with other workloads running on the system-.

In the ranking, we noticed how the measures with progress equal to 100 were all listed first. This is coherent with the definition of cost given in Equation (2.1): if a measure is complete, there is no additional cost required to implement it.

Then, we see how the measures which are listed first tend to exhibit a low effort and high coverage values, whilst the one listed last show a higher ratio between the effort and the coverage values.

5.1.7 Editing a control

We proceeded testing the functionality of editing a control to update its information. To do so, we opened the detail page of a control, clicked the "Edit Control" button and proceeded in modifying the title and the description. Upon submitting the form, the UI refreshes, displaying the updated information.

5.1.8 Tagging

In the page with the list of all the controls, we proceeded adding new tags and assigning them to multiple controls. First, we tested the functionality manually, creating a tag for the "NIST Identify" category of controls and tagging all the appropriate controls. Then, we sped the tag generation and control tagging by interacting directly with the database. We tagged all the controls as "NIST CSF v1", then created and applied another set of tags for each of the five NIST functions -identify, protect, detect, respond, and recover- and third one for each category of controls -ID.AM, ID.BE, &c.-

At this point, we went back to the web UI and checked that the controls were tagged approximately. We also checked that each measure showed as tags the union of the tags of the controls it satisfied by looking at the measure details page.

5.1.9 Filtering

We tested the tag filtering functionality by using it in the homepage, in the controls list page and in a ranking page. In all of these, the filtering worked correctly.

5.2 Adding the CIS Security Controls

The objectives of this second phase of the test is to:

- add the controls from the CIS' framework,
- associate some existing measures to these controls.
- compare the progress between the frameworks

5.2.1 Adding the controls

We added all the controls from the CIS *Critical Security Controls v8* [12] by uploading a second CSV file containing their titles and descriptions.

We tagged each of these controls with a tag "CIS v8" and the tags "IG1", "IG2" and "IG3" according to the implementation group(s) of every control. Our source also contained, for each controls of the framework, the function in the NIST CSF to which it belonged. Therefore, we also tagged each CIS control with the corresponding, preexistent, NIST function tag. This tagging operation was performed through Surrealist.

In the homepage, it is now possible to filter the 261 total controls to only show the ones belonging to a particular framework, CIS implementation group or NIST function -as shown in Figure 5.1-, and the relative measures. Specifying multiple tags it is possible, for example, to see only the controls of the implementation group one of the CIS framework that are concerned with the NIST function of "Respond"

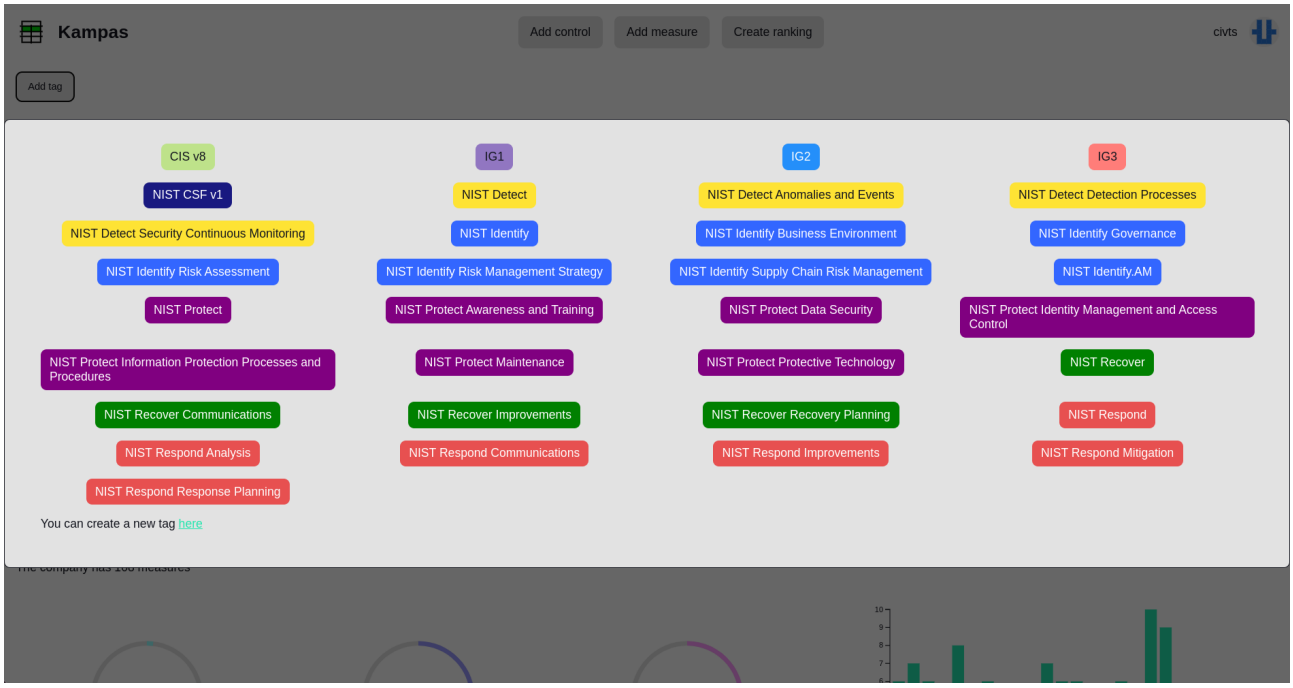


Figure 5.1: Tag selection containing both the NIST and the CIS tags

5.2.2 Associating existing measures

At this point, we randomly associated half of the NIST controls with measures that were tagged with the same NIST function with a random coverage between 1 and 100.

5.2.3 Comparison of the progress

In the homepage dashboard, it is now possible to decide, using the tag filtering, if we wish to see the completion of the controls of only one framework, both, or refine further our filtering.

Having tags which are shared across the controls of more than one framework -in our case, the five NIST functions- is quite useful in determining which preexisting measures for the controls of one framework may be associated with the controls of the other.

6 Final considerations

This chapter provides an overview of the results reached in this work. It includes a summary of the work that was performed during this thesis and the lessons learned in Section 6.1, and suggestions about the direction in which this project may evolve in the future in Section 6.2.

6.1 Discussion

In this work, we acknowledge the need for companies to achieve and maintain a proper cybersecurity posture to manage the risk of sustaining damages in their reputation, operations or capital.

We show how securing a company is a complex task which requires a vast amount of knowledge in multiple areas of expertise since, for each measure that the company may think of adopting, it is insufficient to consider it only from the technical standpoint, as the cost associated with implementation and its impact on the productivity of the employees and the processes of the company cannot be overlooked.

We consider how frameworks from reputable institutions like the NIST and the CIS aim to provide guidance to the businesses by defining a taxonomy of the multiple facets of cybersecurity at a commercial scale and suggesting on which ones to focus. In this regard, we highlight how these work provide valuable information to the companies, yet, at the same time, may end up leaving them perplexed.

Part of the reason is arguably the fact that these frameworks often contain a conspicuous amount of guidelines and suggestions -the cybersecurity controls- and a company may be undecided regarding which of them are applicable to their reality. Another hurdle is represented by the inevitable compromise that the authors of the framework have to make: they can either define the controls very specifically -and render the framework very useful to certain companies or industries-, or they can adopt more general guidelines so as to target a broader audience. This can easily lead to controls that appear too general, abstract or even ambiguous in their definition, hindering the implementation planning.

Another layer of complexity is introduced by the fact that a company may want to comply with multiple frameworks, from multiple agencies, each with its own terminology, conventions and scopes.

We characterize the problem by providing a formal definition for it, and show how in its simpler variants, it can be reconducted to an instance of the set cover problem, which can be solved in polynomial time using greedy algorithms.

Considering this starting situation, we perform an analysis of the problem, identifying the people involved in the process of managing the cybersecurity of a company and its compliance with the frameworks. We documented some of the problems they may face and the needs they may have in the process, obtaining a series of functional and non-functional requirements for a potential solution.

We evaluate, in the light of the aforementioned requirements, existing solutions from both the academic and commercial world, critically assessing their fitness for the task identified. This analysis shows how some of the tools do provide, albeit partly, the functionalities needed to manage the problem. Nonetheless, we were unable to find an existing solution offering sufficient flexibility to the users in how they manage their security controls and, at the same time, help them in prioritizing which amongst the tens or, more likely, hundreds of controls are best to implement first.

Hence, we proceed drafting the architecture for a software system that could be capable of assisting a company in managing the implementation strategy of their cybersecurity controls. We design a solution, named kampas, where multiple users can collaborate to insert, update and rank the security controls.

Kampas has a modular architecture, with a backend written in Rust that manges the data and the core business logic and a frontend written in Svelte that controls the presentation and provides a web UI for the end-users.

The details of the development process are discussed thoroughly, delving into the details of the modeling of the data and the interactions with the database, the ranking creation logic, the API design and functionalities, the access management system and the organization of the routes in the frontend.

In addition to the continuous testing of the components we carried out during the development, we perform -and provide the summary of- a preliminary test during which we verified that all the functionalities of the application worked as intended. During the test, we noticed some possible improvements, which are reported in Section 6.2.

The outcome of test -having been able to complete all the tasks we set as objectives- highlights how the core functionalities have been implemented successfully. With kampas, the users should be able to manage their cybersecurity controls more efficiently.

6.2 Future work

Whilst we do think that the provided solution, in its current state, is able to provide its users with adequate capabilities to aid them in the organization of their security controls, we developed kampas with the intention of making minimum viable product rather than a fully-fledged production application in order to align with the time-frame for this thesis.

In light of this, we have identified several possible improvements for our work which would, in our opinion, represent some of the steps that would be worth integrating to create a second iteration of the software.

6.2.1 Features

Evolution in time The web UI may be enriched to show how the controls, measures and their progress evolves over time. For doing so, it would be possible to harness the ability of SurrealDB to handle timescale data -see Section 3.4.0.3-.

Integration with external tools Many companies use ticketing systems like Jira[1] to assign, manage and track tasks. Adapters could be added to kampas allowing, for example, to have a measure linked to a Jira issue and pull the progress automatically from it.

Community Some of the solutions we surveyed -notably, the CIS CSAT in Section 3.2.2.2- offer their users the possibility to see how other companies in their same sector are faring in terms of compliance with the various standards.

Adding the possibility for the users of kampas to share this kind of information may prove beneficial.

A feature like this, however, would require proper considerations of the trust relations in the system and -most probably- a change in the architecture of kampas with the introduction of some central, trusted, instances.

If companies decide to engage in sharing their data, they could reap the benefits of collaborating on the frameworks. It would become possible -for example- to build shared mappings of the controls from one framework to the others and to create a database of common measures associated to each control.

LLM integration Another future development may be represented by the addition of a LLM in the system. It may be used to suggest new measures for each control, efforts estimates, or even new tags and relations between controls and measures that satisfy them.

Metrics Kampas, in its current form, helps the users choose which security controls to implement and how to implement them. A next step may be represented by helping the users to monitor how the measures they implemented are performing. We propose to do this by including "metric". In our proposal, a metric represents a measurement that kampas can acquire automatically -after a first configuration-. Having metrics would allow the company to continuously monitor the empirical status of their compliance and afford the possibility to generate alerts when anomalies are detected.

[1] `https://atlassian.com/jira`

Alternatively, existing solutions that offer similar features -e.g., Prometheus- may be integrated with kampas.

Exploring the graph The graph view that is offered by the frontend may be enriched by adding the possibility to manipulate the graph in new ways. For example, the user may be allowed to filter the nodes in the graph -or color them- based on a set of tags they specify.

Uploads As of now, it is possible to create multiple controls simultaneously by uploading them in a CSV file. The users, however, may also want the possibility to upload the measures, the tags or the associations between the measures and the controls in batch. Still, this need should be verified before proceeding with the implementation to validate or disprove its presence.

GraphQL We observed how the frontend may at times need some data and at times others. Having a GraphQL API may reduce the amount of requests to the backend and improve performance. At the same time, this would mean giving the frontend wider access to the database, and proper considerations have to be made to avoid introducing vulnerabilities [1], [74].

6.2.2 Security

OAuth/OIDC support Adding support for Single Sign-On through OAuth 2 or OpenID Connect would enable the companies to use their existing identity management system to manage the users and their associated roles.

Logging As of now, neither the backend nor the frontend take a structured approach to logging. Proper logging would help monitoring the status of the application, through the use of tools like Parseable, during runtime [42]. Logging would also be instrumental to enable auditing of the activities performed on the system.

6.3 Conclusion

In this work, we designed and implemented from scratch a software system for helping companies streamline their management of cybersecurity controls.

The resulting program, kampas, supports the functionalities we aimed to provide and is arguably capable of starting to provide immediate value to a company that may decide to adopt it.

We regard its nature of being open source[2], its modular architecture, the fact that its core has been written in a performant, yet memory-safe language, and its independence from any single cybersecurity framework as key differentiators with respect to existing solutions that offer similar feature sets.

We hope that the work detailed in this thesis may serve as a starting point for a project that brings tangible value to its users.

[2] Source code available at `https://github.com/civts/kampas`

Bibliography

[1] N. Aleks, D. Farhi, and O. Chan, *Black Hat GraphQL: Attacking Next Generation APIs*. No Starch Press, May 2023, ISBN: 978-1718502840.

[2] M. Angelini, C. Ciccotelli, L. Franchina, A. M. Spaccamela, and L. Querzoni, *Framework Nazionale per la Cyber Security e la Data Protection, versione 2.0*, Feb. 2019. [Online]. Available: `https://www.cybersecurityframework.it/sites/default/files/framework2/Framework_nazionale_cybersecurity_data_protection.pdf` (visited on 09/14/2023).

[3] A. Asen, W. Bohmayr, S. Deutscher, M. González, and D. Mkrtchian, "Are you spending enough on cybersecurity?," 2019.

[4] P. Bartholomew, "Excel as a Turing-complete Functional Programming Environment," *arXiv preprint arXiv:2309.00115*, 2023.

[5] A. Basuchoudhary and N. Searle, "Snatched secrets: Cybercrime and trade secrets modelling a firm's decision to report a theft of trade secrets," *Computers & Security*, vol. 87, p. 101 591, 2019.

[6] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big? 🦜," in *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, 2021, pp. 610–623.

[7] S. Berlato, R. Carbone, A. J. Lee, and S. Ranise, "Formal modelling and automated trade-off analysis of enforcement architectures for cryptographic access control in the cloud," *ACM Transactions on Privacy and Security*, vol. 25, no. 1, pp. 1–37, 2021.

[8] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: new generation of memory-hard functions for password hashing and other applications," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2016, pp. 292–302.

[9] C. Boettiger, "An introduction to Docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[10] J. S. Brennen and D. Kreiss, "Digitalization," *The international encyclopedia of communication theory and philosophy*, pp. 1–11, 2016.

[11] British Broadcasting Corporation, "British Airways faces record £183m fine for data breach," Jul. 2019. [Online]. Available: `https://www.bbc.com/news/business-48905907` (visited on 09/30/2023).

[12] Center for Internet Security, *CIS Critical Security Controls, version 8*, May 2021. [Online]. Available: `https://www.cisecurity.org/controls` (visited on 09/08/2023).

[13] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[14] Clusit, *Rapporto Clusit 2023*, May 2023. [Online]. Available: `https://clusit.it/rapporto-clusit`; `https://clusit.it/pubblicazioni` (visited on 09/24/2023).

[15] M. Cygan, Ł. Kowalik, and M. Wykurz, "Exponential-time approximation of weighted set cover," *Information Processing Letters*, vol. 109, no. 16, pp. 957–961, 2009.

[16] E. Dolstra, *The purely functional software deployment model*. Utrecht University, 2006.

[17] E. Dolstra and A. Löh, "NixOS: A purely functional Linux distribution," in *Proceedings of the 13th ACM SIGPLAN international conference on Functional programming*, 2008, pp. 367–378.

[18] Y. K. Dwivedi, N. Kshetri, L. Hughes, *et al.*, "So what if ChatGPT wrote it? Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy," *International Journal of Information Management*, vol. 71, p. 102 642, 2023.

[19] Esentire, *2022 Official Cybercrime Report*, 2022. [Online]. Available: `https://www.esentire.com/resources/library/2022-official-cybercrime-report` (visited on 09/18/2023).

[20] M. Ettredge, F. Guo, and Y. Li, "Trade secrets and cyber security breaches," *Journal of Accounting and Public Policy*, vol. 37, no. 6, pp. 564–585, 2018.

[21] European Commission, *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*, May 2016. [Online]. Available: `http://data.europa.eu/eli/reg/2016/679/oj`.

[22] Federal Trade Commission, "FTC Imposes $5 Billion Penalty and Sweeping New Privacy Restrictions on Facebook," Jul. 2019. [Online]. Available: `https://www.ftc.gov/news-events/news/press-releases/2019/07/ftc-imposes-5-billion-penalty-sweeping-new-privacy-restrictions-facebook` (visited on 09/30/2023).

[23] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[24] D. Ferraiolo, J. Cugini, D. R. Kuhn, *et al.*, "Role-based access control (RBAC): Features and motivations," in *Proceedings of 11th annual computer security application conference*, 1995, pp. 241–48.

[25] A. Fijany and F. Vatan, "New approaches for efficient solution of hitting set problem," 2004.

[26] E. A. Fischer, *Cybersecurity issues and challenges: In brief*, 2016.

[27] U. S. Foerster-Metz, K. Marquardt, N. Golowko, A. Kompalla, and C. Hell, "Digital transformation and its implications on organizational behavior," *Journal of EU Research in Business*, vol. 2018, no. 3, pp. 1–14, 2018.

[28] A. Fox and E. A. Brewer, "Harvest, yield, and scalable tolerant systems," in *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, IEEE, 1999, pp. 174–178.

[29] Francis (Software Analyst), "The beginner's guide to cybersecurity," *The Software Analyst Newsletter*, Sep. 2023. [Online]. Available: `https://softwareanalyst.substack.com/p/the-beginners-guide-to-cybersecurity` (visited on 09/23/2023).

[30] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[31] S. Furnell, P. Fischer, and A. Finch, "Can't get the staff? The growing need for cyber-security skills," *Computer Fraud & Security*, vol. 2017, no. 2, pp. 5–10, 2017.

[32] Google, *Ep000: Operation aurora | hacking google*, Oct. 2022. [Online]. Available: `https://youtube.com/watch?v=przDcQe6n5o` (visited on 07/26/2023).

[33] P. N. Grabosky, "The Evolution of Cybercrime, 2004-2014," *Economics of Networks eJournal*, 2014. [Online]. Available: `https://api.semanticscholar.org/CorpusID:107580063` (visited on 09/12/2023).

[34] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao, "A framework for native multi-tenancy application development and management," in *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, IEEE, 2007, pp. 551–558.

[35] G. Hatzivasilis, S. Ioannidis, M. Smyrlis, *et al.*, "Modern aspects of cyber-security training and continuous adaptation of programmes to trainees," *Applied Sciences*, vol. 10, no. 16, p. 5702, 2020.

[36] S. M. Hawkins, D. C. Yen, and D. C. Chou, "Disaster recovery planning: a strategy for data security," *Information management & computer security*, vol. 8, no. 5, pp. 222–230, 2000.

[37] *ISO/IEC 27001:2022 Information Security Management Systems*, Standard, Geneva, CH, Oct. 2022.

[38] A. M. Joy, "Performance comparison between Linux containers and virtual machines," in *2015 international conference on advances in computer engineering and applications*, IEEE, 2015, pp. 342–346.

[39] R. Karp, "Reducibility Among Combinatorial Problems," vol. 40, Jan. 1972, pp. 85–103, ISBN: 978-3-540-68274-5. DOI: 10.1007/978-3-540-68279-0_8.

[40] V. G. Khalin and G. V. Chernova, "Digitalization and Cyber Risks," *Administrative Consulting*, no. 7, 2023. DOI: 10.22394/1726-1139-2023-7.

[41] B. Korte and J. Vygen, "Combinatorial optimization. Theory and algorithms. 2000," *Cited on*, p. 84, 2005.

[42] H. Li, W. Shang, and A. E. Hassan, "Which log level should developers choose for a new logging statement?" *Empirical Software Engineering*, vol. 22, pp. 1684–1716, 2017.

[43] Y. Li and Q. Liu, "A comprehensive review study of cyber-attacks and cyber security; Emerging trends and recent developments," *Energy Reports*, vol. 7, pp. 8176–8186, 2021.

[44] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, Inc., 2012.

[45] C. M. McRae, R. Wesley McGrew, and R. B. Vaughn, "Honey tokens and web bugs: Developing reactive techniques for investigating phishing scams," *Journal of Digital Forensic Practice*, vol. 1, no. 3, pp. 193–199, 2006.

[46] D. Merkel *et al.*, "Docker: lightweight linux containers for consistent development and deployment," *Linux j*, vol. 239, no. 2, p. 2, 2014.

[47] B. Middleton, *A history of cyber security attacks: 1980 to present*. CRC Press, 2017.

[48] M. Mollaeefar and S. Ranise, "Identifying and quantifying trade-offs in multi-stakeholder risk evaluation with applications to the data protection impact assessment of the GDPR," *Computers & Security*, vol. 129, p. 103 206, 2023.

[49] National Institute of Standards and Technology, *Framework for Improving Critical Infrastructure Cybersecurity (NIST CSF), version 1.1*. Gaithersburg, MD, Mar. 2018. DOI: https://doi.org/10.6028/NIST.CSWP.04162018.

[50] P. O'Kane, S. Sezer, and D. Carlin, "Evolution of ransomware," *Iet Networks*, vol. 7, no. 5, pp. 321–327, 2018.

[51] Office of Cybersecurity, Energy Security, and Emergency Response, *Cybersecurity Capability Maturity Model (C2M2), version 2.1*, Jun. 2022. [Online]. Available: https://www.energy.gov/ceser/cybersecurity-capability-maturity-model-c2m2 (visited on 09/14/2023).

[52] H. S. Oluwatosin, "Client-server model," *IOSR Journal of Computer Engineering*, vol. 16, no. 1, pp. 67–71, 2014.

[53] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX annual technical conference (USENIX ATC 14)*, 2014, pp. 305–319.

[54] M. Paik, J. Irazábal, D. Zimmer, M. Meloni, and V. Padurean, *immudb: A Lightweight, Performant Immutable Database*, 2020.

[55] Password Hashing Competition, "PHC string format," 2021. [Online]. Available: https://github.com/P-H-C/phc-string-format/blob/master/phc-sf-spec.md (visited on 09/10/2023).

[56] PCI Security Standards Council, *PCI Data Security Standard (PCI DSS), version 4.0*, Mar. 2022. [Online]. Available: https://docs-prv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI-DSS-v4_0.pdf (visited on 09/14/2023).

[57] R. M. Rosenberg and E. V. Hobbs, "The spreadsheet," *Journal of Chemical Education*, vol. 62, no. 2, p. 140, 1985.

[58] D. Rumsfeld, *Known and unknown: a memoir*. Penguin, 2011.

[59] B. Schwartz, P. Zaitsev, and V. Tkachenko, *High performance MySQL: optimization, backups, and replication*. " O'Reilly Media, Inc.", 2012.

[60] P. Sharma, L. Chaufournier, P. Shenoy, and Y. Tay, "Containers and virtual machines at scale: A comparative study," in *Proceedings of the 17th international middleware conference*, 2016, pp. 1–13.

[61] I. Sommerville, "Software engineering (10th edition)," *America: Pearson Education Inc*, 2017.

[62] L. Spitzner, "Honeytokens: The Other Honeypots," *Symantec.com Security Focus*, 2010.

[63] K. Stouffer, T. Zimmerman, C. Tang, J. Lubell, J. Cichonski, and J. McCarthy, *Cybersecurity framework manufacturing profile*. National Institute of Standards and Technology, May 2019. DOI: https://doi.org/10.6028/NIST.IR.8183.

[64] M. Strangfeld, "Reproducibility of Computational Environments for Software Development," 2022.

[65] Swascan, *Report Threatland Q2 - Trend e scenari del Cybercrime*, Sep. 2023. [Online]. Available: https://www.swascan.com/it/report-threatland-q2 (visited on 09/14/2023).

[66] M. Syafrizal, S. R. Selamat, and N. A. Zakaria, "Analysis of cybersecurity standard and framework components," *International Journal of Communication Networks and Information Security*, vol. 12, no. 3, pp. 417–432, 2020. DOI: https://doi.org/10.17762/ijcnis.v12i3.4817.

[67] J. Toigo, *Disaster recovery planning: Preparing for the unthinkable*. Prentice Hall Professional Technical Reference, 2002.

[68] V. V. Vazirani, *Approximation algorithms*. Springer, 2001, vol. 1, ISBN: 9783662045657.

[69] Verizon, *2023 Data Breach Investigations Report*, 2023. [Online]. Available: https://www.verizon.com/business/resources/reports/dbir (visited on 09/27/2023).

[70] R. Vogel, "Closing the cybersecurity skills gap," *Salus journal*, vol. 4, no. 2, pp. 32–46, 2016.

[71] S. A. Waterman and R. Astley, "*Whenever You Need Somebody*," *RCA Records*, 1987. [Online]. Available: https://youtube.com/watch?v=dQw4w9WgXcQ (visited on 07/27/2023).

[72] A. Wiggins, *The Twelve-Factor App*, 2017. [Online]. Available: https://12factor.net (visited on 09/02/2023).

[73] H. Xu, Z. Chen, M. Sun, Y. Zhou, and M. R. Lyu, "Memory-safety challenge considered solved? An in-depth study with all Rust CVEs," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 1, pp. 1–25, 2021.

[74] S. Yazdipour, "GitHub data exposure and accessing blocked data using the GraphQL security design flaw," *arXiv preprint arXiv:2005.13448*, 2020.

Acronyms

API Application Programming Interface. 27, 28, 31, 46, 71, 72

CIS Center for Internet Security. 1, 9, 24, 66, 69–71

CISO Chief Information Security Officer. 1, 21–23

CSS Cascading Style Sheets. 32

CSV Comma Separated Values. 67, 72

DBMS DataBase Management System. 30, 31

GDPR General Data Protection Regulation. 1, 5, 7

HTML HyperText Markup Language. 32, 62

HTTP HyperText Transfer Protocol. 47, 53, 55

ICT Information and Communication Technology. 5

IDS Intrusion Detection System. 6

ILP Integer Linear Programming. 15

IP Internet Protocol. 43

IPS Intrusion Prevention System. 6

JSON JavaScript Object Notation. 23, 29, 47–49, 51, 52, 67

LLM Large Language Model. 71

NIST National Institute of Standards and Technology. 1, 11, 15, 34, 66, 69, 70

OIDC OpenID Connect. 72

PHC Password Hashing Competition. 36, 54

RBAC Role-Based Access Control. 47, 53, 56

REST REpresentational State Transfer. 46

SDK Software Development Kit. 30, 31, 33

SEO Search Engine Optimization. 32

SSG Static Site Generation. 32

SSO Single Sign-On. 72

SSR Server-side Rendering. 32

UI User Interface. 2, 32, 42, 57, 67, 68, 70, 71

URL Uniform Resource Locator. 37, 43, 48, 50, 52

Appendix A CIS CSAT Additional Screenshots

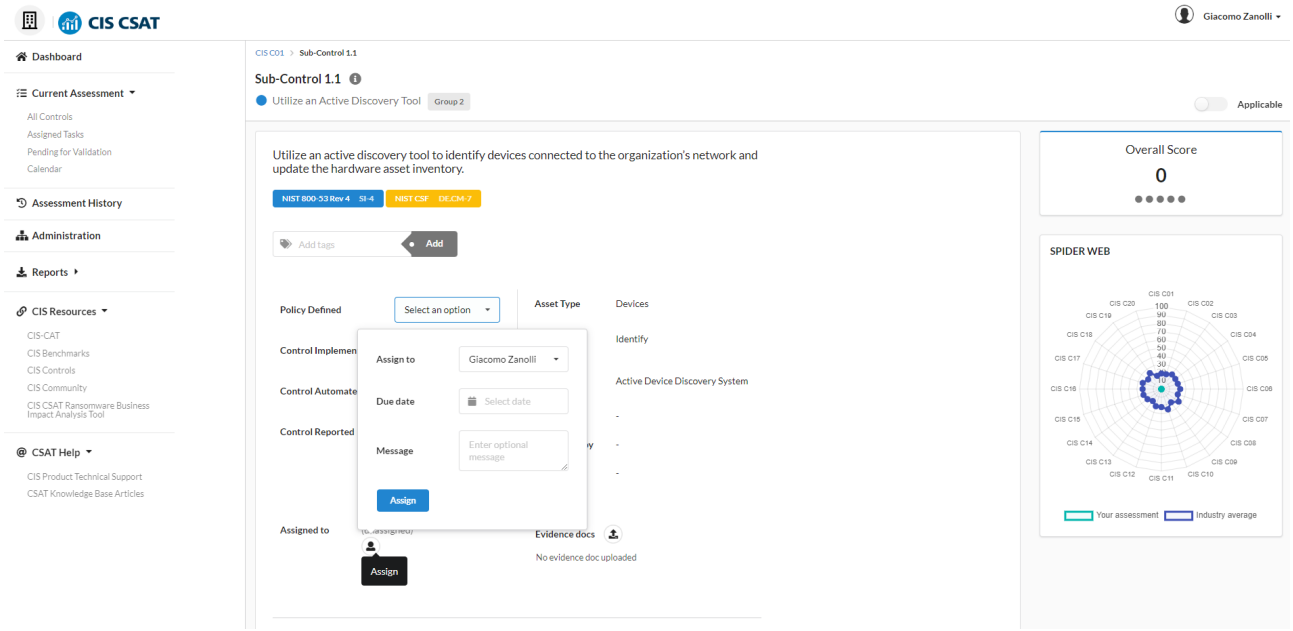


Figure A.1: CIS CSAT, assigning a control as task

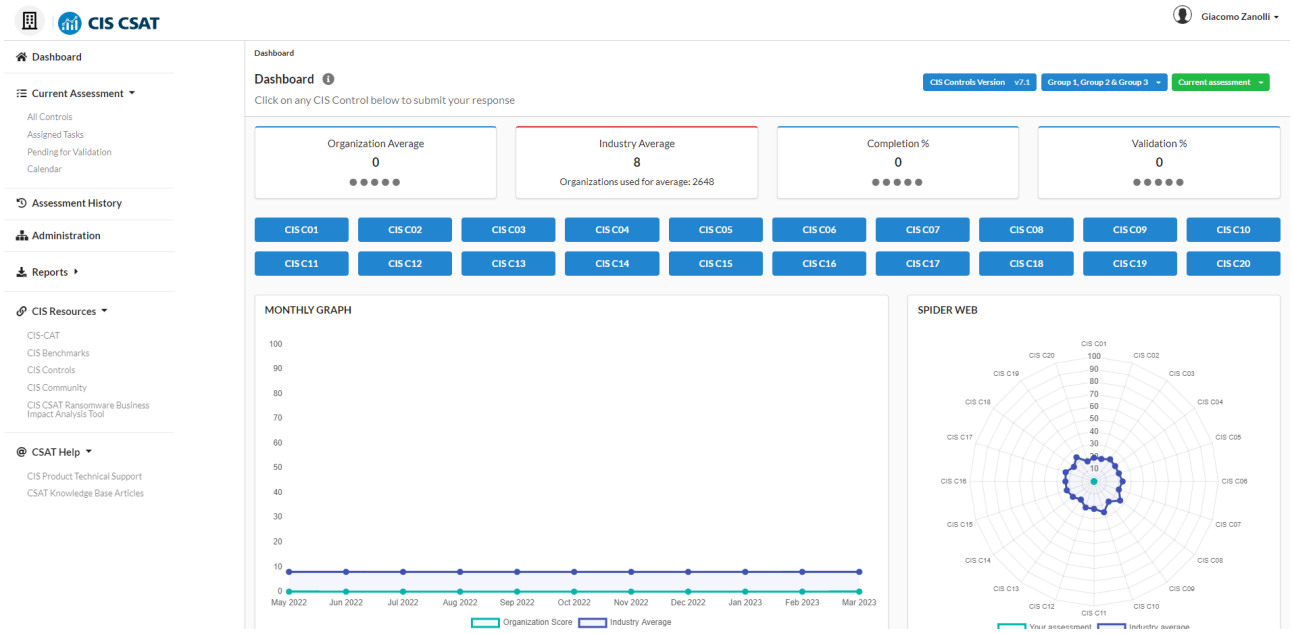


Figure A.2: CIS CSAT, dashboard page

Appendix B Hitting Set Problem

The problem discussed in Section 2.5 can also be seen as an instance of the hitting set problem. We report here the definition provided by Fijany *et al.*[1] in [25, §1]:

> A collection $S = \{S_1, \ldots, S_m\}$ of nonempty subsets of a set U is given. A hitting set (or transversal) of S is a subset $H \in U$ that meets every set in the collection S; i.e., $S_j \cap H \neq \emptyset$, for every $j = 1, \ldots, m$.

The mapping between our problem and the hitting set problem is direct. In our case, the set U coincides with the set of measures M and the set S can be obtained from the set of controls C by converting each of its elements, c, to the set of measures that satisfy c, which we can denote with S_c.

As an example, using the input described in Section 2.5.1.2, we can consider a set $U = C = \{a, b, c, d\}$ and a set S containing the following subsets: $S_1 = \{a, c\}$, $S_2 = \{d\}$, $S_3 = \{a, c, d\}$, and $S_4 = \{a, b\}$. In this case, there are several hitting sets: $\{S_3, S_4\}$, $\{S_1, S_2, S_4\}$, $\{S_1, S_3, S_4\}$, $\{S_2, S_3, S_4\}$, and $\{S_1, S_2, S_3, S_4\}$. Graphically, we can represent the input as in Figure B.1 -which we colored coherently with Figure 2.1-.

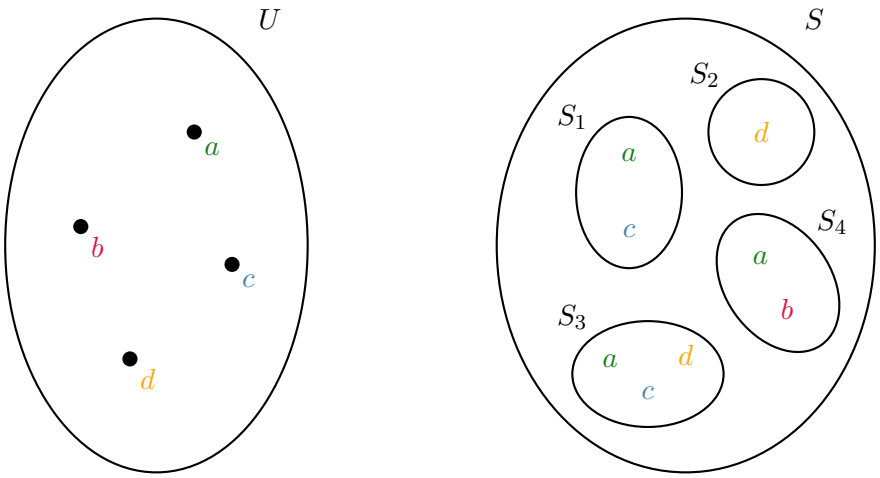


Figure B.1: Example of the hitting set problem

The hitting set problem exists in two versions: a decision version and an optimization version. In the optimization version, we are tasked with finding the smallest hitting set -meaning the hitting set with minimum cardinality-. In the decision version, we are given as an additional input parameter an integer k and have to decide if there is a hitting set with cardinality that is not greater than k. The decision version has been shown to be NP-complete by Karp in 1972 [39]. In the same work, it is shown how the problem can be reduced in polynomial time to the set cover problem.

[1] We changed the symbol M with U to make it consistent with the other problems' definitions