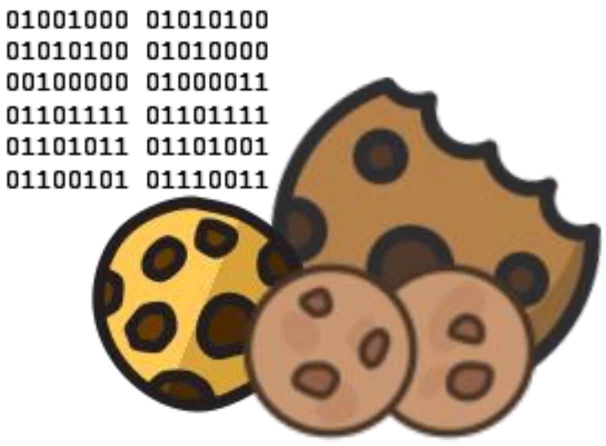


Platform to manage cookies

Marco André Melo Carneiro

Mestrado em Segurança Informática
Departamento de Ciência de Computadores
2021

Orientador
Luís Filipe Coelho Antunes
Professor Catedrático
Faculdade de Ciências da Universidade do Porto

Supervisor
Engº. Pedro Vasconcelos Castro Lopes Faria
TekPrivacy

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/_________

Abstract

Cookies have the fundamental function of providing a website state record mechanism between user-server transactions and connections. They can also improve the experience of a given user and help a website operator understand how it behaves during the visit. Since their introduction, these data structures have been solidified as integral components of the Internet as it is known today, and can be found on several websites serving different privacy goals. Despite this fact, there is next to no established standard that dictates what and how much information cookies and third-party cookies can collect and to which end. Their widespread use, coupled with the lack of standardization of data collection can lead to major security implications.

This thesis' purpose is to analyze a broad assortment of known cookie management platforms, including both open source and proprietary solutions, to provide the best sense of the array of features these platforms possess. Furthermore, cookie scanning and collection tools were also studied. The project aims at gathering a proper understanding of the expected functionalities of a feature-rich platform, how it helps a website owner in managing the cookies the website uses and how it can help achieve compliance with the current law, namely May 2016, European Union approved, General Data Protection Regulation (GDPR).

Resumo

As cookies têm o papel fundamental de fornecer um registo do estado do site entre as transações e conexões cliente-servidor. Podem também melhorar a experiência de um determinado utilizador e ajudam o proprietário do site a entender como utilizadores se comportam durante a visita. Desde a sua introdução, estas estruturas de dados foram solidificadas como componentes integrais da Internet, tal como é conhecida hoje, e podem ser encontradas em vários *sites* com diversos propósitos. No entanto, não existe um padrão estabelecido que dite qual e quanta informação as cookies, e de igual forma as cookies de terceiros, podem coletar e para que fim. O uso generalizado de cookies, juntamente com a falta de estandardização da recolha de dados, pode levar a importantes implicações de segurança.

O objetivo desta tese é analisar uma ampla variedade de plataformas de gestão de cookies, incluindo *software* livre e soluções proprietárias, para fornecer uma melhor compreensão da variedade de recursos que estas plataformas possuem. Além disso, ferramentas de recolha e *scanning* de cookies também foram estudadas. O projeto visa obter uma compreensão adequada das funcionalidades esperadas de plataformas ricas em recursos, dado que ajudam o proprietário de um website a gerir as cookies que este usa e como podem ajudar a cumprir a lei em vigor, nomeadamente o Regulamento Geral da Proteção de Dados, aprovado pela União Europeia em maio de 2016.

Acknowledgements

I want to thank Professor Luís Filipe Coelho Antunes for the support, clarification, and guidance which proved to be key at the beginning of this dissertation. I also want to thank Pedro Faria for his continuous support during the dissertation. Thanks are also extended to Ana Carvalho and Tânia Carvalho for the guidance they provided during the realization of this document.

Lastly, I want to thank my family who always kept pushing forward with me during the pandemic.

Contents

List of Tables

List of Figures

Listings

Acronyms

EU	European Union
CMP	Cookie Management Platform
CMS	Content Management Systems
GDPR	General Data Protection Regulation
GPDPL	General Personal Data Protection Law
CCPA	California Consumer Privacy Act
ePD	ePrivacy Directive
ePR	ePrivacy Regulation
EDPS	European Data Protection Supervisor
GPC	Global Privacy Control
DPA	Data Processing Agreement
DPO	Data Protection Officer
JSON	JavaScript Notation Object
YAML	YAML Ain't Markup Language
DOM	Document Object Model
URL	Uniform Resource Locator
CDN	Content Delivery Network
CSS	Cascading Style Sheets
RFC	Request for Comment
REST	Representational State Transfer
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
SPA	Single Page Application
API	Application Programming Interface
IE	Internet Explorer
MVC	Model-View-Controller
XML	eXtensible Markup Language
XSS	Cross-Site Scripting
CSRF	Cross-Site Request Forgery
URL	Uniform Resource Locator
WEC	Website Evidence Collector
SSL	Secure Socket Layer
DNT	Do-Not-Track

Chapter 1

Introduction

Cookies are implemented for various purposes. Examples are the "basket" in online shopping or auto-login. Thus, cookies and tracking functions may be essential for certain types of websites. Such websites need to save the status of a client during a visit. Furthermore, cookies serve the purpose of authenticating certain websites and services, namely, to provide sub-pages to authorized users. This constitutes examples of the functions of first-party cookies. Other cookies serve no real direct purpose to the website a user visits. Therefore they serve data to a third-party entity and can, at times, extract relevant or identifiable data from a user session for advertisement purposes, usually in violation of European regulations (e.g. lack of consent). Third-party cookies can make up the majority of present cookies on a website.

The problem of managing cookies stems mostly from the lack of insight the owner and visitor have on the website they use. A website owner, the data controller, faces the often daunting task of managing a myriad of scripts that help a website to function. A visitor, on the other hand, lacks power over the choice of what scripts should run on its browser. The next section of the problem is linked to the absence of tools that possess a GDPR compliant banner, mostly in part by not allowing the users to 'opt-in' to the use of certain cookies or not offering the option at all. Even if the option for 'opting-in' is provided, it often lacks transparency to what cookies and services are present, what is their purpose and only allows 'opt-in' per general category. Such an approach falls short of the conditions for consent in Article 7 [55], and transparency and granular control expected from GDPR compliant banners in accordance with Articles 5, nº1, line c), "(Personal data shall be:) adequate, relevant and limited to what is necessary (...)" [55]. The GDPR, however, is not the only legislation in place that regulates the use of cookies and, specifically, only mentions cookies once in Recital 30 [55] with "Natural persons may be associated with online identifiers provided by (...), cookie identifiers (...)." indicating cookies that are used to identify users qualify as personal data. The ePrivacy Directive (ePD) passed in 2002 [53] and amended in 2009 [54] are equally, if not more, important that the GDPR. The ePD addresses important factors such as the confidentiality of e-communications and the tracking of user activity online. Its introduction was responsible, much like with the GDPR, for the rise of cookie consent banners across the Internet. For this reason, the ePD is often called the "cookie law". In an effort to revise the ePD, in 2017, the European Commission introduced the ePrivacy Regulation (ePR) [9] and it was supposed to be passed alongside the GDPR, however, this did not happen. The ePR aims to expand the already in effect ePD by updating it increasing the scope by addressing browser fingerprinting as a form of user tracking, accounting for news ways of communication, and more protections for metadata [37]. Aside from the European Union (EU) and its legislations, the State of California in the United States of America enacted the California Consumer Privacy Act (CCPA) [48] to cover similar concerns over the handling of user data by businesses and other entities. The CCPA differs in scope and reach territory-wise, mostly by being limited to only the State of California. One key difference is the definition of personal data. The CCPA differentiates personal data as data provided by a consumer and thus does not apply if the information is made public by the data subject itself. This makes the GDPR's definition broader as it applies regardless of the information's source [3, 25].

1.1 Proposed Solution

In order to create a more fair and compliant relationship between users and website owners, we propose an application that aggregates insight gathering functionalities such as cookies/script scanning, cookie management functionalities that allow for granular cookie consent, and proper cookie usage information disclosure that allows the visitor to understand which scripts are about to be set by the website.

The concern over unlawful data sharing stems from the irresponsible way organizations and website owners allow third-parties to set their cookies on their websites. In a broad sense, there are severe deficiencies in how public and private entities deal with this issue [8].

The proposal encompasses the use of already established tools to achieve distinct objectives. The application should, then, be an amalgamation of different tools, each with a different responsibility of providing the functionalities previously mentioned and an application developed to mitigate any shortcoming that may exist.

1.2 Contributions

The theme of this dissertation was proposed in partnership with TekPrivacy. TekPrivacy is an organization that focuses on the development of technological solutions to support the activity of the Data Protection Officer, and therefore, in addition to the continuous improvement in services and products in production, it seeks to develop new solutions.

Hence, the cookie management platform aims for data controllers to ensure that personal data is collected and processed in a lawful and auditable manner. Consequently, the result of the proposed solution will later be implemented as a complementary service of the existing solution. Furthermore, the proposed solution aims to be a valid contribution to the scientific community, who in itself, contributed with open-source tools and information to the fulfillment of the present proposal.

1.3 Thesis Structure

This document is structured in the following manner:

- **State of the Art** - This chapter goes into detail about what constitutes a "cookie" and its uses. Elaborates on some of the most well-known and used market solutions of today that aim to solve the management problem of cookies. Each solution's features are detailed to shed light on how they handle their data and how they can contribute to this proposal.

- **Design** - The chapter consists of the design process, describing the main concepts, specifying the requirements, and all the necessary components that comprise its inner workings.

- **Development** - This chapter details the steps taken in the development process focusing on explaining some of the major choices that were made and how they impact the features of the application. The chapter also explains how each feature utilizes all the technologies discussed in the Design chapter.

- **Conclusion** - This chapter summarizes the progress made during the time dedicated to the project and all the suggested future work that might help mature the developed solution.

Chapter 2

State Of The Art

This chapter will provide an overview of the currently available and studied solutions present on the market of cookie and consent management platforms. It will also compare these solutions and how they fare in building a proper open-source and complete solution for cookie management.

2.1 Brief History Of Cookies

The name ”cookie”, in this context [59], was created by Lou Montulli [40], a programmer who worked as an engineer at Netscape Communications in 1994. This designation derives from the term ”magic cookie” previously used by Unix developers to describe small data structures that a program receives and sends without any changes. Originally, cookies were used to verify that a visitor had visited the Netscape website before [39].

Support was added in the second version of Internet Explorer (IE) that had been launched in 1995 [29]. In that same year, Lou Montulli applied the patent for the cookie technology. The patent was guaranteed years later in 1998 [40].

Concern about the impact of this technology is not new, so much so that in 1996 the Financial Times published an article about the cookies. Since the use of cookies was not widely known by the public, in that same year it received proper attention on its possible risks, such as the possibility of being stolen and misused.

The lack of standardization led to the emergence of various formats of cookies to ensure the state in HTTP communications. It was decided that the Netscape format would be used as a starting point and RFC 2109 [43] was also created. This specification dictates that the use of ”third party” cookies should not take place at all or, at least, should not be allowed by default.

In 2000, RFC 2965 specification was created to escape the restrictions established by RFC 2109 that neither IE nor Netscape decided to follow [44]. However, even this specification was eventually replaced by RFC 6265 in 2011 [1].

2.2 Cookies And Trackers In The Present

Activities such as automatic login, online shopping cart, or authorized access to subpages are categorized as stateful. A server has no way of performing these tasks without certain information being retained between sessions or requests. Therefore, cookies are used to give the server the information necessary to perform those operations.

However, not all cookies have the same purpose. Tracking cookies (also called third-party cookies) are used to record the use and behavior of visitors to websites, including the history of visits. This type of record is seen by the EU as a potential violation of online privacy and thus mobilized itself to create protective measures. For this reason, many websites currently require users to provide consent to store non-essential cookies on their system. Other cookies may have a more vital function for the functioning of the Internet, such as authentication cookies. These cookies are used by the servers to determine whether a user is logged in or not. Through these cookies, a server can provide authorized pages only to users whose permissions guarantee them access. Failures in the integration of these cookies can lead to attacks such as ”cross-site scripting” attacks and ”cross-site request forgery” attacks [33, 34].

Albeit cookies, as a technology, receive the most attention from both users and website owners, being the most widely used, they are not the only ways a website can store data on a user’s browser. Browsers possess a small *key-value* database that depending on each use case might be called either LocalStorage or SessionStorage. LocalStorage is read-only and persists between sessions. SessionStorage does not persist between sessions and is then deleted once one such session ends [61].

Such web storage types were introduced with HTML5 in 2008 [30], and they possess the following advantages compared to cookies:

- Simple, as it uses JavaScript only,
- 10MB data limit per origin (Firefox and Chrome),
- Doesn't have to be sent in every request/response with the server,
- Apps are sandboxed, only accessible from the domain that created them.

For use cases such as a user's mailbox or user files, not having this data being always sent to the server with each request largely boosts performance, cookies on the other hand possess a maximum size of 4kB. However, web storage options also have disadvantages compared with cookies [61]:

- Vulnerable to Cross-Site Scripting (XSS) due to being purely JavaScript,
- Not as standardized as cookies yet (Cookies have access to HttpOnly and Secure flags).

In contrast to the above-mentioned methods of cookies and WebStorage, which by design are not meant for tracking but can be used as such, there is another method that has no other purpose than tracking user access to specific content [49]. Originally, web beacons were named "Web bugs" or "1-pixel images/gifs" for the way they were implemented into webpages. By loading a page with a web beacon, a client would send a request to the location where the "1-pixel image" was located. This request would contain basic data about the browser and thus relaying both the webpage from where the request was made and the aforementioned information. As it became more common, web beacons could be small sections of a page or even buttons that reported the activity to the entities that employed them [58]. This last method of employing web beacons has been discontinued since the introduction of HTML 5 [30]. Web beacons were also embedded in emails, usually by phishers or spammers, in an effort for it to behave as a "reception receipt" and know if and when a user opens the fraudulent email. The introduction of the Web Beacon Application Programming Interface (API) has introduced a level of standardization without diverting resources from the website that uses such methods for tracking [26, 46].

2.3 Basic Structure

Cookies are text structures that store small amounts of data separated by semi-colons and therefore do not have a complex or large structure. A cookie is made up of *key-value* pairs named attributes. A simplified cookie possesses a name attribute and a related value (<cookie-name>=<cookie-value>) [1, 45]. An example of such a simplified cookie:

- PHPSESSID=302qd17hg323fh5
- sid=1:cF3CnluKkK2Mwp9yQQqbz5zMdLC (...) DvOH5

It's possible to have more attributes with information related to the cookie such as Expiration time, the Host from which the cookie originated, Path and Domain to determine the exact scope of the cookie, or other flags, for example, HttpOnly or Secure. The expiration date, host, domain/path, and flags can also determine which type of cookie it is, as the next section will elaborate. An example of a cookie with the aforementioned attributes:

- NAME=VALUE; Domain=domain.com ; Path=/pathtopage ; Expires=Sun, 22 Aug 2021 13:00:00 GMT; HttpOnly;

Cookies are transmitted by one of two fields in the headers of an HTTP message,

- Cookie;
- Set-Cookie.

The field **Set-Cookie** is used by HTTP messages sent by the server in order to set one or more cookies on the client. The syntax is as follows [1, 45]:

Set-Cookie: <cookie-name>=<cookie-value>;<attribute>=<attribute-value>;

The field **Cookie** is normally sent on requests by the client in which the server requires information that only the client has, for example, authentication information. The simplified syntax is as follows:

Cookie: <cookie-name>=<cookie-value>;<cookie2-name>=<cookie2-value>;

2.4 Types Of Cookies

There are different types of cookies depending on their use and where they are implemented. As described before, the basic structure of a cookie possesses a name attribute as a key and an attributed value. Although cookies can be separated into the following categories, they can be inserted into several categories simultaneously due to having different attributes [1], purposes, and which entity sets them.

Cookie Lifetime

- Permanent: Unlike session cookies, persistent cookies have an **Expiration date** and, as such, persist for the specified time. This feature is for the cookie to store information that can be accessed whenever a resource belonging to the original website detects it, such as, for example, advertising on other websites. Due to this functionality, these cookies are also called tracking cookies, precisely because they maintain the status of visits through different websites. However, they are also used to store login data between sessions, which ensures that a user does not need to enter their credentials each time they visit the website [1].

- Session: A session cookie is a type of non-persistent cookie that exists only while a user visits a website. If a cookie does not have an **Expiration date** then the browser treats it as non-persistent and therefore will be deleted at the end of the session [1].

Cookie Provenance

- First-Party: A first-party cookie is a cookie that belongs to the domain the user is currently visiting. Such cookies are used for analytics, performance, and/or language data. First-Party cookies serve, at times, the purpose of including authentication credentials or other preferences on the website that set them so they aren't used to track a user's activity across different websites [1].

- Third-Party: A third-party cookie belongs to a different domain than the user is visiting at the time. These cookies allow it to be possible to track what a user views, as they allow the appearance of content external to the current website, for example, for advertising. Assuming that a user visits a website ”www.X.com” with advertisements from ”pub.foo.com”, the latter sets a cookie. If the user visits another website ”www.Y.com” with advertisements from ”pub.foo.com”, another cookie is set. Eventually, both are sent to the advertising owner, and the owner will learn about the user’s online habits [1].

Cookie Security

- Secure: Cookies that can only be transmitted over HTTPS connections are called Secure Cookies. Depends on the **Secure** attribute being set in the cookie [1]. Using HTTPS adds overhead to each transmission, however, the security advantages are far too many to ignore, and for that reason, these cookies are prime targets for collection and exploitation given that they might contain sensitive information depending on their use.

- HttpOnly: Http-Only cookies are cookies with the addition of the **HttpOnly** flag. This flag prevents the cookie from being accessed by client-side APIs. Therefore, it is not vulnerable to XSS. However, it does not prevent CSRF attacks [1].

- SameSite: In an attempt at mitigating CSRF attacks, in 2016, Chrome introduced a new type of cookies. Other browsers such as Firefox and Edge started to support these cookies at a later date. Same-site cookies add an attribute called **SameSite**. This attribute allows greater control over the use of cookies in addition to the domain that created them. **SameSite** can have 3 levels - **Strict, Lax, or None** [45, 62].

  - Strict: the browser only sends these cookies on requests that originated on the same domain as the domain being accessed. This level prevents CSRF attacks;
  - Lax: the browser does not restrict the site where requests originate but enforces what is being accessed to be the same as the domain of the cookie. In this way, third-party cookies would be blocked;
  - None: allows any cross-site cookies.

The implementation of this type of cookie (Lax Mode) on a wider scale has been delayed as of the writing of this thesis. The reasons behind this decision are the COVID-19 pandemic and the switch from None level to Lax would break several websites that rely on cross-site/third-party cookies [62]. One implemented, it will update RFC 6265 [1].

Cookie Purpose

- "Strictly necessary" cookies or Essential cookies: An essential cookie is labeled as such because it is necessary for the proper browsing experience of the website that employs it. Necessary features that make use of this type of cookie might be authentication credentials and the "shopping cart" feature on online marketplaces. These types of cookies are set by the website the user might be visiting and as such are considered first-party cookies as well [37].
- Preference cookies: Cookies that deal with preference data can also be called "Functionality" cookies due to providing data to certain website features that depend on user specification. One example of a preference cookie is the language setting preference [37].
- Statistics cookies: Cookies that collect information about the usage of the website are also called "Performance" cookies. Cookies with this purpose have the task of collecting information such as the number of pages visited. This information helps improve a website's functions and usually cannot be used to identify a specific user. Some websites might use third-party services for this task, however, such cookies are only in this category if they restrict their usage to the website the user is currently visiting [37].
- Marketing cookies: Marketing cookies exist to control the marketing material that is displayed on any given website. They track the online activity of a user to increase the relevancy of displayed ads. These cookies share this information with the other parties, and, because of this, they are persistent in nature and also classified as third-party cookies [37].

Other Examples

- Supercookie: Supercookies are normally blocked by browsers. The reason for this course of action stems from the fact that an attacker in control of a website can set a supercookie and take control of requests from other users who share the same top-level domain or suffix. This is possible because these cookies originate from top-level domains or public suffixes (ex: ".com") [1].

- Zombie cookie: As the name implies, a zombie cookie is a cookie that is added even after it has been deleted. A cookie can be a zombie cookie if it is stored in different locations (client-side or server-side). These cookies are recreated when it is detected that they have ceased to exist [1].

2.5 Tracking

Online tracking is a way that websites have to store and later share the online habits of visitors on the Internet. The biggest attraction of this practice is being able to determine a user's preferences, to provide relevant or influenced content. Tracking can be used, for example, for advertising, web analysis (performance), or even usability tests. When performed without the user's consent, this activity can be considered a security breach of the browser [41].

The use of cookies is not always a synonym of web tracking, as first-party cookies may provide legitimate insight to website owners. As mentioned before, there are more uses for cookies than targeted advertisement and it's also important to note that the majority of first-party cookies do not focus on advertisement altogether. It's relevant, nevertheless, to make the distinction between the real impact first-party cookies have versus third-party cookies. According to [5] published in April 2016, upon visiting the top 100.000 Alexa websites, the researchers registered that 94.6% of the landing page visits resulted in at least one cookie being set on the browser. Additionally, the research showed that third-party cookies comprised roughly 2/3 of all collected cookies, while first-party cookies only made up 1/3 of all results. From these results, we gain the insight that, from the pool of first-party cookies, the majority are performance-related cookies and that the majority of these performance cookies and targeted advertisement cookies are of the persistent type. From the gathered third-party cookies, the overwhelming majority are dedicated to targeted advertisement, persistent, all the while having the **Domain** and the **Path** attributes set at a root path. This indicates the third-party cookies can monitor and track users for a substantial amount of time.

The overreliance and overuse of third-party cookies is a problem also being tackled by browser developers. Specifically, Google announced its deviation from the use of third-party cookies by the end of the year 2023. Google's plan revolves around the creation of a Privacy Sandbox [18, 32, 60]. The introduction of such a technology does not imply the end of user tracking through the Internet. This mostly means the already in use third-party cookie providers will simply resort to workarounds to continue tracking. The implementation of the Privacy Sandbox is facing strong scrutiny by the European Commission [23] as it can, at times, make identification even easier by relying on unique identifiers for users. Additionally, unless all browser developers stop supporting third-party cookies, the goal of any technology as the Privacy Sandbox will never come to fruition.

These emerging issues will persist even after the eventual discontinuation of third-party cookies due to the profitability of tracking information. Because of this, consent management is and will remain an important task for website owners, not only for third-party cookie management but also for the management of any information that the website may process or store.

2.6 Cookie Management

With the strong use of third-party cookies from many websites, including for advertising and login information, website owners may feel overwhelmed with the number of scripts, cookies and trackers present while surfing the Internet that they have to manage.

Essentially, a CMP is a software that allows a website owner to manage the state of use of cookies and to keep their website in compliance with the GDPR and other legislation related to cookies and user information. Additionally, they allow websites to inform their visitors about what data is used and whether these visitors consent to its processing. Therefore, CMPs allow:

- The use of consent warning messages or warning banners;
- Allow or deny third-party cookies based on user consent;
- Prevent the automatic use of cookies by the website until consent is given.

2.6.1 Cookie Management Platforms

After the introduction of GDPR, CMPs became even more important for website owners. Entities responsible for managing websites that cannot or do not wish to integrate specific consent solutions for their website can implement CMPs. In this way, they comply with the current legislation.

Currently, there are several CMPs with different functionalities. The following section is an analysis of each of these CMPs.

Osano's CookieConsent

Osano's [52] option is divided into two parts: CMP and banner. Osano makes the banner available for free and open-source, allowing any owner to enter a message of their choice to inform visitors to their website. In addition, the banner has customization options that support 'opt-in' for the use of cookies. The banner possesses no granularity in the cookie selection. However, it is up to each website owner to decide what to do with these options if they choose not to use Osano's CMP or none at all. Finally, this banner can be integrated with a proprietary CMP and sold.

Osano's CMP is a solution that aims to combine compliance with legislation and ease of use. CookieConsent is also able to use geolocation to determine which is the local legislation [52].

The main features of CookieConsent are as follows:

- Automatic blocking of third-party cookies;
- Use of geolocation to block cookies based on local legislation;
- Uses simple Javascript events to determine consent status;
- Automatically detects new third-party cookies that try to add themselves to the website;
- Records the visit of each user;
- The user's visit is registered on a private blockchain ensuring validity.

Osano offers a free version of this CMP, however, its functions are relatively restricted to only one domain and a limited number of monthly visits. While maintaining the blocking of cookies, it does not keep track of visits nor automatically detects new cookies unless the owner manually indicates them [52].

CookieBot

CookieBot [15] is a CMP as a cloud service provided by Cybot [17]. This solution complies with the ePD [53, 54] and the GDPR [55]. The goal of CookieBot is to establish a solution that is easy for website owners to implement and at the same time make a simple experience for end users.

Cybot separates the CMP's functionalities into three categories - Core, Efficiency and Configuration [16].

- Core functionality: CookieBot performs automated cookie scans and declares the results alongside the banner for users to see. These results also integrate with the privacy policy. CookieBot will save a user's consent for 12 months after which the banner will reappear for a user to renew its consent if it so wishes.

  From a website owner's perspective, CookieBot will automatically block all first-party and third-party cookies and trackers present. It is up to the website owner to decide which scripts are necessary and which are not. The code for this is accessed through a JavaScript object exposed by the CookieBot CMP. Furthermore, CookieBot leverages a global cookie repository with statements about specific cookies and their purpose. After a scan, such descriptions are made available for the website owner to freely change locally to better reflect a determined cookie's purpose in the website's context.

- Efficiency: CookieBot supports bulk consent for multiple domains and sub-domains in the case the website owner operates several. Such a feature allows the CMP to prompt users for consent only once. CookieBot supports geo-targeting to effectively determine which visitors require a consent banner from those who do not. This data is added to the monthly report that CookieBot makes available to the website owner. The report contains additional info about the trackers found on the website in the latest scan. Security-wise, all consent records are communicated through SSL-protected channels and stored as encrypted keys.

- Configuration: CookieBot has an easy process of deployment with just some lines of JavaScript onto the website. By default, CookieBot will hold back the loading of all first-party and third-party scripts on the website. Strictly necessary scripts are up to the website owner to whitelist. Cookie also supports a wide array of browsers as long as they support HTML 4 and HTML 5, JavaScript and Secure Socket Layer (SSL). CookieBot respects Do-Not-Track (DNT) browser settings and, thus, it does not set the cookie which enables bulk consent effectively rendering this feature unusable.

CookieYes

CookieYes [47] is a CMP with support for several Content Management Systems (CMS) like Drupal, WordPress, Joomla, and Magento. It allows the automatic blocking of third-party cookies (Google Analytics, Facebook Pixel, Hotjar, and Youtube).

CookieYes maintains an extensive record of unique cookies with detailed descriptions and categorizes them according to their purpose and function.

This CMP automatically detects cookies and scripts and adds them to the website's list of cookies. After categorizing these cookies, it allows the website owner to personalize the consent banner to reflect this list of cookies. A user can then granularly provide consent. Each user's consent is kept and maintained by the CMP [47].

This CMP is the engine used by the cookie scanner, CookieServe [13].

CookieFirst

CookieFirst [12] provides compliance with the GDPR, General Personal Data Protection Law (GPDPL) [24] and CCPA [48] through the use of banners and consent management.

CookieFirst [12] allows the management of third-party cookies and scripts on the main portal by selecting these services and adding the code elements necessary for their operation. CookieFirst is responsible for the addition and everything the website owner needs to do is to add the CookieFirst script to his website. For example, a WordPress website is fully supported, and therefore, only needs the addition of the CookieFirst plugin and the website owner will have both the banner and the consent panel in full operation.

Monthly, CookieFirst automatically analyzes the website for cookies and updates the consent declaration. The results of this analysis can be viewed on the portal. After adding the scripts that the website owner wants on the website, the CookieFirst portal automatically updates the cookie declaration to reflect the changes. In addition, the consent given by visitors is kept in an encrypted and anonymous database. The website owner can check the consent status of each user at any time.

This CMP offers a superior degree of customization for consent actions. One of the features aims to offer the possibility of consent being changed to include more cookies even after the 'opt-out' option has been chosen. A particular visitor, for example, can give consent to essential cookies but deny performance or advertising cookies; the owner can define a period before the consent banner is shown again to this specific user to enable him to give consent again [12].

OneTrust PreferenceChoice

OneTrust's management solution called PreferenceChoice allows website owners to ensure compliance with hundreds of privacy regulations by requesting users to opt-in. The software is capable of analyzing mobile applications, in addition to websites, providing greater control over the data that is sent and processed.

PreferenceChoice uses scans to detect where data is sent by both applications and websites. These scans can be scheduled and performed periodically to detect third-party cookies. The results are compared with the Cookiepedia [50] database to categorize the cookies found.

In addition, PreferenceChoice automates the process of requesting and changing consent from the moment it is given until the final report. The entire process is carried out taking into account the legislation in the user's location [51].

CookieConsent

CookieConsent is a solution that focuses on offering a free choice and simple CMP to manage cookies. This CMP sets itself up as a streamlined, step-by-step, arrangement by letting any website owner create a banner with little effort. With four steps, a website owner can choose a compliance preference, customize the banner, add the necessary scripts responsible for setting cookies (or any other) and export the code needed to add to a website [11].

CookieConsent allows for consent per category of cookies (Advertisement, Analytics, etc.), but for its simplicity, it doesn't provide enough granularity for proper transparency [11].

Klaro

Klaro [36] and open-source CMP developed by KitProtect. This CMP allows the granular consent of cookies and leaves it to the website owner to identify the trackers and cookies used.

Klaro's emphasis is on the simplicity of use. Any changes to the settings can be made by editing a JavaScript object. Average knowledge of this language is the only barrier to the use of this CMP.

Klaro has a small footprint – about 50kB. This ensures that it loads quickly and the end-user does not notice any difference in the use of the website [35].

2.6.2 Cookie Scanners

CookieChecker

CookieChecker [47] is a website that allows checking of cookies present on a domain. This website uses CookieBot for analysis, and therefore, it is not a complete system on its own. It is very similar in operation to CookieMetrix and CookieServe and thus allows for an analysis of a designated website.

CookieMetrix

CookieMetrix [56] is a tool that should be used as a starting point for improvements and not to carry out a rigorous analysis. The tool simulates a user's visit to a specific page with a typical browser. The tool does not respect 'Do Not Track' requests (since most browsers do not use this feature by default) and does not perform the simulation with any addons or extensions. The simulation generates a brief report with four parts:

- Cookies found;
- Third-party scripts analysis;
- Banners on the page;
- First level links found.

The engine that enables the simulation/analysis is not available to the public, however, there are plans to be made available soon. The paid plan differs from the free plan in the type of analysis, simulation, and report that is generated. The paid plan allows more pages to be scanned, can be used in a commercial environment and the report is more extensive [56].

Disconnect.me Private Browsing Plugin

Disconnect.me [20] is similar to the options already studied but is not focused only on cookie management. Additionally, not being a CMP but a browser plugin, Disconnect.me works best for personal purposes, allowing a user to control cookies on the websites it visits.

The plugin separates cookies into four categories: 'Advertising', 'Analytics', 'Social', and 'Content'. Blocking cookies from the 'Content' category can sometimes prevent the proper functioning of a website that uses external resources. If the design of a website is considered poor, any blocking of cookies may prevent the website from functioning properly.

Disconnect.me supports Global Privacy Control (GPC) [10], but is still in the experimental stage. When this option is selected, the plugin sends a 'Do Not Sell' request to the website according to the CCPA [48].

This plugin fundamentally differs from the "incognito" mode of most browsers by preventing trackers and blocking websites/servers from reading the user's browser history.

CookieServe

CookieServe [13] is a tool that uses the CookieYes cookie management platform as an engine. The way it works is similar to CookieMetrix in that a user can analyze a website through the URL. Like the aforementioned alternative, CookieServe generates an analysis report.

CookieScanner

CookieScanner is a utility for analyzing a given website and generating reports on the status of cookies and compliance with GDPR. It makes use of a database specifically designed to identify cookies called CovenantSQL. The database is an alternative to Amazon's database and is open-source, decentralized, and uses Blockchain technology. The use of this database is exclusive to the "Cookie Database" functionality [14].

Website Evidence Collector

The Website Evidence Collector (WEC) is a tool for automating the process of collecting data on websites (cookies and data processing methods) [22].

The WEC was developed by the European Data Protection Supervisor (EDPS) as a ”standalone” tool to enable any individual - after a brief introduction - to collect information about how a website processes data. WEC uses ’headless’ Chromium to create the necessary environment to acquire information about a given website. Additionally, the tool makes use of the ’puppeteer’ automation tool to provide a control API to the Chromium environment.

WEC produces several result documents and a final report in HTML. The report aggregates information about cookies, links, and screenshots of the target website [21].

2.7 Comparison

The comparison was made based on four criteria: ease of installation, features, price of use, and the license to use. These criteria are relatively uneven and favor the type of license and the features that allow better interoperability with the system to be developed. In the comparison, criteria such as ’installation’ and ’features’ are scored in three levels. Installation score is dependant on the difficulty and knowledge needed to install and get a banner displayed to users; for instance, if the CMP or Scanner requires the website owner to go through several steps then it scores the lowest, conversely, a CMP that boasts a near plug-and-play experience scores the highest. Features are linked to the payment plan chosen and, thus, only the free version is accounted for this comparison; the software scores the highest if it features high granularity of results, allows for high customization of the scripts present on the webpage, and possesses tools that automate the process managing the cookies present, and it scores the lowest if the solution compared is limited in usage, restricts the number of domains it can manage and possesses weak transparency and granularity in the results that it is meant to produce.

CMP Comparison

When comparing CMPs, an important note must be made. While several of the solutions explored contain numerous features, these are sometimes behind restrictive payment. Certain restrictions are related to the number of domains and subdomains that the CMPs manages. The comparison for CMPs is represented in Table 2.7. The table does not portray the differences in granularity between solutions as they are a part of the feature set. Nevertheless, it's a key feature and deciding factor.

	Installation	Features	Cost	Open-Source
Osano CC Free	**	*	Limited Free Version	Yes (Free Version)
CookieBot	***	**	Limited Free Version	No
CookieYes	***	***	Limited Free Version	No
CookieFirst	***	**	Limited Free Version	No
OneTrust PC	***	***	Limited Free Version	No
CookieConsent	***	**	Free	Yes
Klaro	*	**	Free	Yes

Table 2.1: CMP comparison table.

Scanner comparison

Cookie scanners have comparison criteria similar to CMPs. From the perspective of the system to be developed, a scanner distinguishes itself in comparison if it has few restrictions on use and provides a comprehensive analysis of the website's cookies. The comparison is represented in Table 2.7.

	Installation	Features	Cost	Open-Source
CookieChecker	***	**	Free	No
CookieMetrix	***	**	Free	No
Disconnect.me Plugin	**	***	Free	Yes
CookieServe	***	**	Free	No
CookieScanner	*	**	Free	Yes
WEC	**	***	Free	Yes

Table 2.2: Cookie scanner comparison table.

2.8 Summary

The analysis of solutions and alternatives that are currently available allows for a better understanding of what needs to be done and what objectives need to be addressed. Furthermore, it highlights solutions that, although competitive, might lack some features. The analysis also concludes that there are features that are common or expected to be present in all of the alternatives such as the option to manually disable cookie usage.

Presently, open-source CMP solutions tend to be harder to use despite posing a far lesser financial strain to website owners (excluding running costs as remotely hosted solutions can be cheaper in the long run and can be more robust).

Cookie scanners follow a similar pattern such that scanners that use certain CMPs as backends also have the same advantages and shortcomings as their respective CMPs if they happen to use any.

For this analysis, open-source solutions take priority and are favored. For this reason, solutions such as Klaro and WEC have a more favorable approach as they can be self-hosted and more configurable.

Chapter 3

Design

The main objective of this thesis is the development of a prototype for a cookie management application. The application was designed to possess a friendly user experience, ensuring the management of cookies through the creation of configuration files and exporting the files to a banner. Furthermore, the process of creating and editing configuration files demands a rich user interface [42]. This chapter provides the application of technological concepts and design patterns including the chosen technologies and their main features.

3.1 Concepts

The application employs technological concepts, standards, and design patterns. Concepts describe the type of technology that was favored to achieve the solution's goals. A choice was made early into the solution's conception to be web-based. This was mainly chosen for its flexibility, and the solution should be able to be accessed through a computer with a browser installed. In the next enumeration are concepts found to be more advantageous to provide an easy-to-use experience and the addition of features pertinent to a cookie management solution:

- **SPA** is a solution that allows the dynamic change of the information present on a web page with new information coming from the server [2]. This avoids the loading of new web pages giving a more fluid experience to the user. In this way, the operation is closer to a normal application than a traditional website. Therefore, it should be noted that the page is never reloaded during operation as Figure 3.1, however, elements such as the HTML5 History API may change to preserve the browsing history. This can be done with three elements: HTML, Cascading Style Sheets (CSS) and JavaScript.

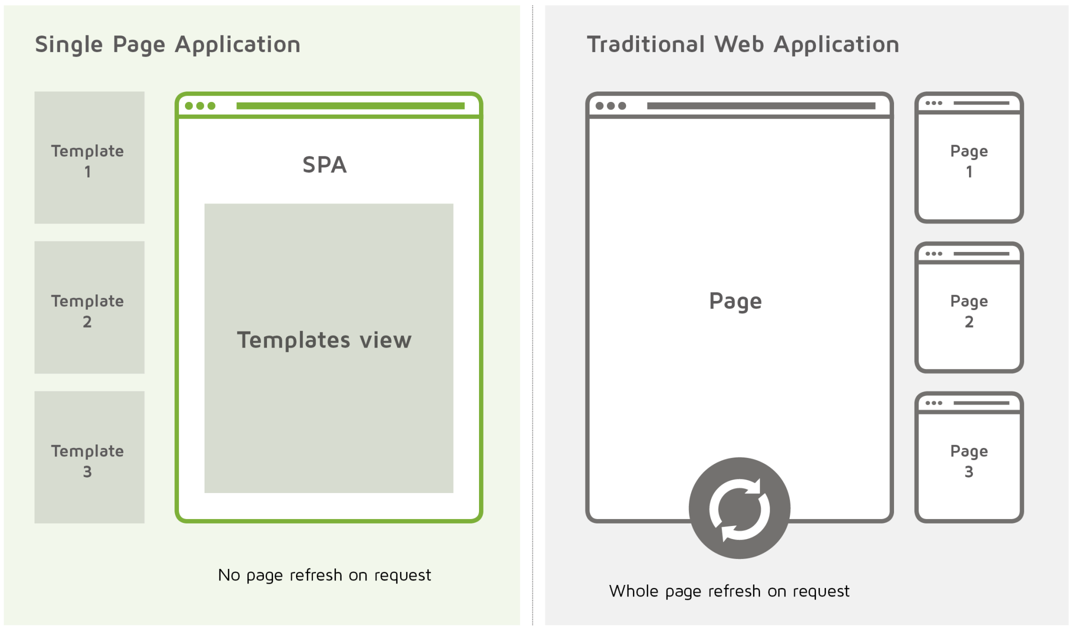


Figure 3.1: SPA vs Traditional Webpages [2, 6].

- **JSON** is an easy-to-understand data exchange format based on the structure of key/value pairs as Listing 3.1 exemplifies. It is used as an alternative to eXtensible Markup Language (XML) and is a subset of the JavaScript programming language. However, it is completely independent of it and can be used relatively easily with other languages such as C, C ++, and Python.

```
    {
        "firstName": "John",
        "lastName": "Smith",
        "isAlive": true,
        "age": 27,
        "address": {
            "streetAddress": "21 2nd Street",
            "city": "New York",
            "state": "NY",
            "postalCode": "10021-3100"
        }
    }
```

Listing 3.1: JSON structure example [4].

- **API** is a form of software interface made to provide a specific set of services to other pieces of software while abstracting underlying code implementation [57]. The most relevant types of APIs for this project are Web APIs and software libraries. Web APIs expose certain specifications, for example, as HTTP end-points and exchange information in JSON. Software libraries are interfaces to actual implementations of code. Such implementations can share the same programming interface to simplify their use. Such an example is the Puppeteer software library used to control a headless version of Google Chrome as seen on Listing 3.2. Puppeteer is a NodeJS library and as such, with help from other tools like WEC, can run through the command line.

```
const browser = await puppeteer.launch();

const page = await browser.newPage();
await page.goto('https://google.com');

console.log(await page.content());
await page.screenshot({path: 'screenshot.png'});

await browser.close();

```

Listing 3.2: Puppeteer visit to a webpage example [27].

The technological concepts and standards mentioned previously can be implemented in certain design patterns. What follows are the architectural design patterns relevant to the developed solution:

- **REST** is an architectural standard that allows communication between computer systems on the web. A REST system is called RESTful and separates client and server responsibilities. One of the great advantages of this standard is the ease of communication between these systems. There are four HTTP messages that when used allow for resource exchange with other REST systems: GET, POST, PUT, DELETE. Such interactions between a client and REST server are shown in Figure 3.2.

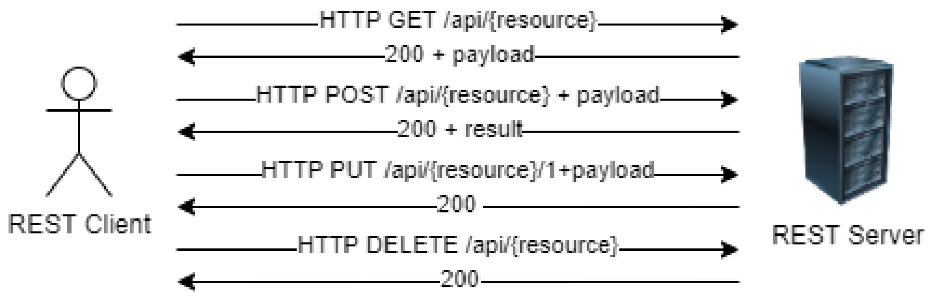

HTTP GET /api/{resource}
200 + payload
HTTP POST /api/{resource} + payload
200 + result
HTTP PUT /api/{resource}/1+payload
200
HTTP DELETE /api/{resource}
200
REST Client
REST Server

Figure 3.2: REST HTTP messages example [1].

- **MVC** is a design pattern that separates the logic of an application into 3 parts - Model, View, and Controller. Each part is responsible for a different representation of data [38]. The Model deals with data independently from the user interface and possesses any data-related logic that the user might work with. The View is any representation of the data to a user and the visual elements that make up the user interface. The Controller mediates commands from the View to the Model and vice-versa. The interaction can be represented in a simplified manner in Figure 3.3. Because of the relative ease of use, this pattern is favored in the design of user-interface heavy applications such as web applications [19].

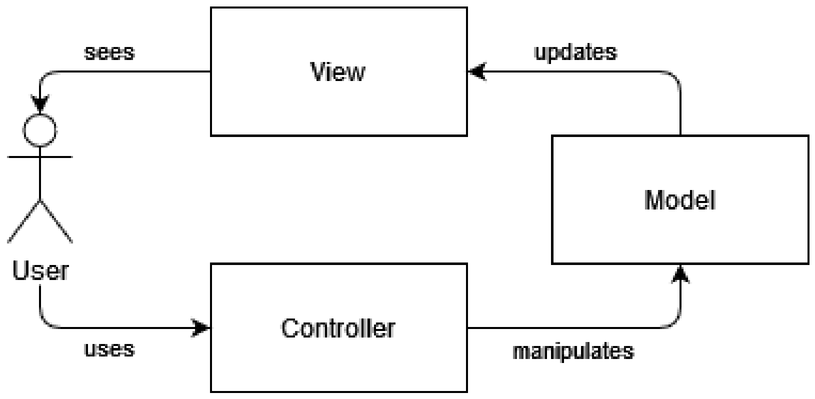


Figure 3.3: MVC interactions.

3.2 Elements

The cookie management application will be divided mainly by two elements: the frontend and a REST server backend. The first is responsible for making the information available to the user. The second is responsible for performing website analysis and safeguarding user preferences. The separation of the solution into two distinct servers allows for the segregation of responsibilities, separates certain functionalities that interact directly with server functionalities, and permits the addition of more features that may not be directly shown to the user such as better scanning capabilities or user access control. The downsides for such an approach are the increase in overall complexity and features directly linked with user input might need adjustments on both ends. Regardless, the REST server possesses functionalities that, for security reasons (i.e. received user input for the website analysis target) should be separated from the frontend. The exchange of resources between servers will be done using the JSON format in HTTP messages.

Additionally, the application requires additional elements aside from the two mentioned above. Two extra elements are required: a consent banner provider and a cookie scanner. These elements work together with the application as follows:

- The consent banner interprets the configuration generated by the application and displays the consent message to the website visitor.
- The scanner targets a website and lists the cookies in use. This is particularly useful for website owners to know what cookies are being used on their own websites.

3.3 Chosen Technologies

The application communicates with the additional elements in the manner described in the previous section. The application, divided into the frontend and REST server backend, uses the Angular 11 and Express NodeJS frameworks respectively. Such frameworks were chosen for their ease of implementation, alongside the availability of different tools for the development and customization of the user experience. The application is organized as pictured in Figure 3.4.

The NodeJS framework is an open-source, multi-platform JavaScript execution system. NodeJS operates on the JavaScript V8 engine, the same JavaScript engine as the one used for Google Chrome.

The additional elements – the consent banner and the cookie scanner – chosen and used together with the application are Klaro and EDPS Website Evidence Collector

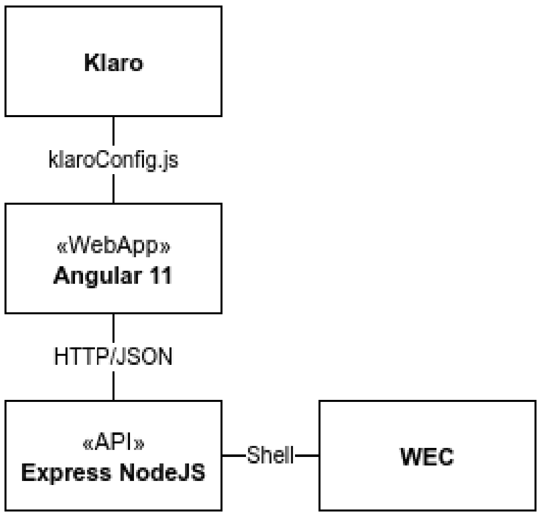


Figure 3.4: Simple architecture of the elements.

respectively.

The workflow, taking into account Figure 3.4, is as in Figure 3.5.

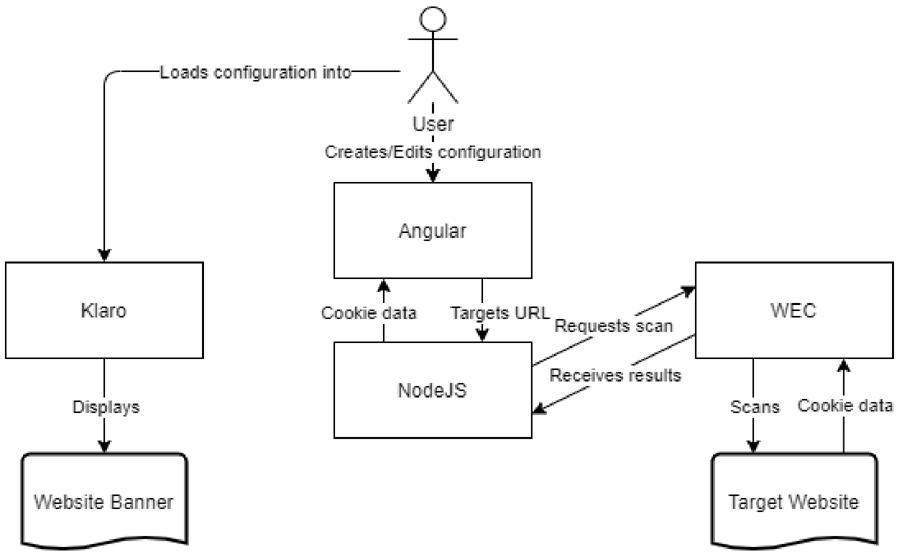


Figure 3.5: Application workflow.

3.4 Overview

The developed application is divided into three main pages or sections: Dashboard, Configuration, and Scanner. In addition, a Login page is considered for the application to provide access control, although it's not a priority as the application is a toolbox in nature.

3.4.1 Configuration Page

The Configuration page gives the website owner the possibility to create a configuration for the consent banner or to load an existing configuration. This page also provides the function to export the configuration after it has been edited.

The desired concept should divide the page into two distinct sections. The first section should create a new configuration file. The second section should allow the website owner to load a Klaro configuration file and edit its properties.

The configuration editing step would be the same for both sections to keep consistency between the create and load sections. Figure 3.6 is a concept of how the page is to be organized.

Conceptually, this page comes rather close to the way one of the studied solutions – CookieConsent – works. With a few steps, the website owners should be able to set categories, the scripts it wishes the CMP control, and export the configuration.

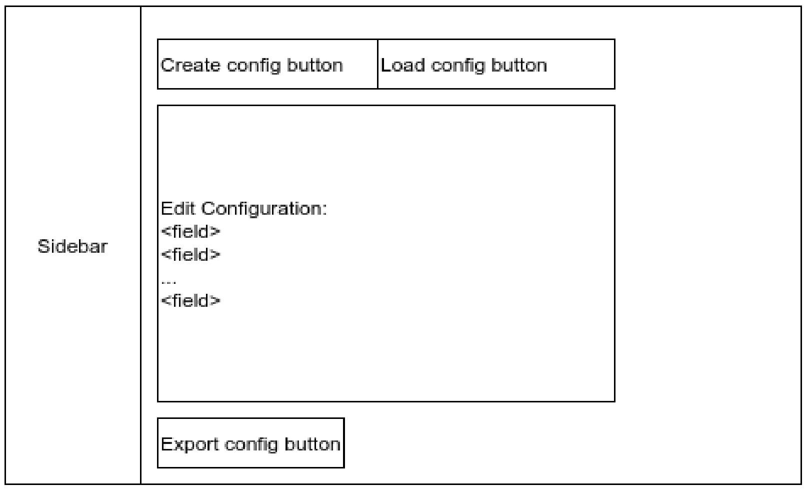


Figure 3.6: Configuration page concept.

3.4.2 Scanner Page

The Scanner page allows the analysis of a website provided a Uniform Resource Locator (URL). After the user initiates the analysis, the backend server uses the additional website analyzer/scanner element. This element, in turn, visits the website and returns the cookies present. The cookies should present all the information that they contain within in a way conceptualized in Figure 3.7.

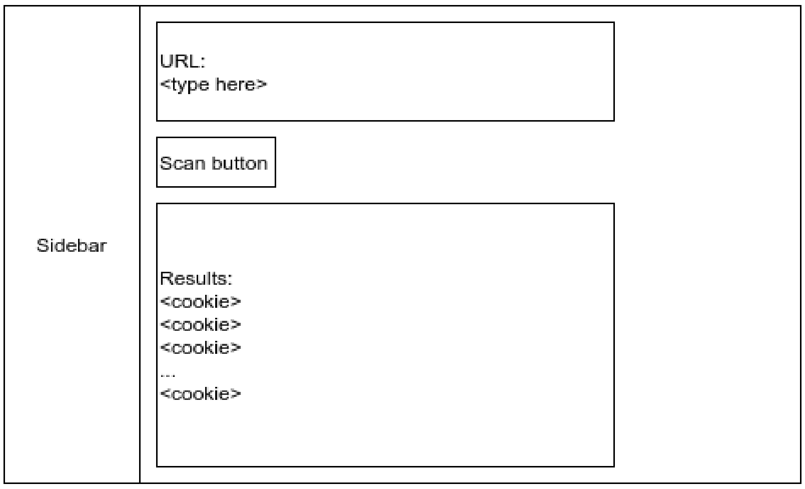


Figure 3.7: Scanner page concept.

3.5 Requirements

The Configuration page is responsible for three essential user stories, and the Scanner page is responsible for the last user story. More requirements specific to each page will be introduced in the this section. From a general perspective, the application must, at least, provide the essential functional requirements for the user stories illustrated in the Figure 3.8.

In regards to non-functional requirements, the application should take into account any future redesign that aims to increase its availability. Furthermore, as part of the application's objectives, it must accelerate the process of usage from configuration to result. Non-functional requirements are listed in Table-3.5.

3.6 Summary

The application should provide a set of tools to allow website owners to understand what cookies are used and to immediately provide further tools to properly categorize and manage said cookies in accordance with their needs. The tools are meant to

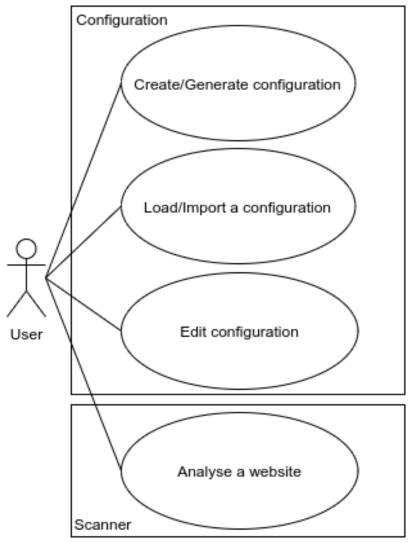


Figure 3.8: Essential user stories.

Usability
Simple yet appealing design
Customization options should be easy to understand
Configuration must be able to be saved locally

Reliability
Configuration creation should function even if the scanning function fails

Scability
Modularity of components for future implementation of features
Separate configuration features and scanning features
Scanning features are implemented server-side to implement future custom features

Table 3.1: Non-functional requirements.

contextualize and modify the way data collection is done with cookies and third-party cookies by giving a procedure that's easy to follow, understand and complete. The application should make up for the advanced knowledge needed to use tools like Klaro and WEC.

Chapter 4

Development

The application, as elaborated in Chapter 3, was developed using Angular and Express NodeJS. This design was chosen to allow it to be a service that sits between the context/cookie data collection with WEC and the presentation of the banner. This way, the application and website owner can have a better understanding of the cookies that are used and how to customize the banner. In this chapter, we present the details regarding the development process.

4.1 Klaro

The Klaro software functions by loading a JavaScript file into the website and a JavaScript configuration (more specifically a variable). Klaro can be imported using the script 'src' tag in HTML with either the website owner hosting the JavaScript file or KitProtect's Content Delivery Network. This makes Klaro relatively easy to integrate, however, without a correct syntax JavaScript configuration file, Klaro cannot present its banner properly and may even cease to function altogether [35]. Klaro can be imported in the following manner:

<script defer src=”.../klaro.js”></script>

The 'defer' element refers to the loading method of the script. This specific method forces the page to load before Klaro. Such an element is needed to not have any script load after Klaro can manage it [35].

The modifications to the website aren't limited to the ones mentioned. Each script the website owner wants Klaro to manage needs to have its HTML element changed from *script* to one that Klaro recognizes. This step allows Klaro to stop the loading of this script until the user gives consent. The modifications are as follows:

- Inline scripts
  - Inline scripts should change their 'type' attribute to 'text/plain';
  - Previous 'type' attribute should be place in a dedicated property called 'data-type';
  - An additional property is to be added called 'data-name'.

  Changes like the 'type' attribute prevent a script from loading unintentionally.

- External scripts
  - External scripts need the same modifications as the inline scripts;
  - Change 'src' property to 'data-src'.

  The change of the 'src' attribute is also to prevent a script from loading into the page unintentionally.

The 'data-name' property must match the name set in its service section on the configuration file Klaro uses. The website owner needs to include two scripts in the website. The first is Klaro itself and the configuration [35]. Although these changes are manually done by the website owner, elements such as the configuration will be generated by the developed application to streamline its usage and avoid loading errors that might appear due to bad formatting.

The manual changes end at those highlighted above. The web application handles configuration creation and modification. A minimal configuration can be structured in the following sections:

- Properties;
- Translations;
- Services.

The configuration properties (e.g. Listing 4.1) are string and boolean values that set certain behaviors and information of the Klaro banner and manager. String values define the text displayed on the banner, the type of storage Klaro will use (cookie or local storage) and the name of the storage chosen. Boolean values define how Klaro handles the manager, for example, should cookies be aggregated by purposes such as advertisement or analytics.

```
var klaroConfig = {
    elementID: 'klaro',
    storageMethod: 'cookie',
    storageName: 'klaro_cookie',
    groupByPurpose: true,
    acceptAll: false,
    /* truncated */
    translations: { /* truncated */ },
    services: [ /* truncated */ ]
}
```

Listing 4.1: Properties example.

The configuration possesses the Translations section. Translations exist here to modify the text displayed on the consent banner. This section should not be confused with the translations section in the Services, as the last are exclusive to the service in question and the prior does not interfere with its function. Furthermore, the Translations section also sets the purposes, or rather, the text displayed in each (see Listing 4.1). If any purpose is left out from the configuration, it will simply not appear in the banner.

```
var klaroConfig = {
    /* truncated */
    translations: {
        en: {
            privacyPolicyUrl: '/privacy',
            consentModal: {
                description:'Description'},
            purposes: {
                analytics: {
                    title: 'Analytics'
                },
                /* truncated */
            },
        },
    },
    services: [ /* truncated */ ]
}
```

Listing 4.2: Translations example.

The Services in the configuration are responsible for describing what scripts the website owner intends to monitor. Each service must specify a name, description, purpose, and the cookie(s) that it sets. The latter can be set using regular expressions to match dynamic named cookies. Alternatively, if the cookie name is always the same, the website owner can just specify the name outright with a string value. The domain of the cookie can also be specified.

```
var klaroConfig = {
    /* truncated */
    translations: { /* truncated */},
    services: [
        name: 'Example',
        default: true,
        translations: {
            en: {
                description: 'This cookie does X'
            }
        },
        purposes: ['analytics'],
        cookies: [
        /* Matches a cookie containing example in the name */
            [/^example.*$/, '/','ex.example.com'],
        ]
    ]
}
```

Listing 4.3: Services example.

The setup of these values is one of the main focuses of the developed application. The goal is to make the process of this setup as quick as possible. Even though some of the values and properties that Klaro supports can be omitted in the configuration, this leaves these values as default. Given this fact, the web application attempts to strike a balance between giving the website owner as much choice as possible without leaving settings to default and keeping the complexity at a minimum by not barraging the website owner with too many options that pose no real value.

4.2 Website Evidence Collector

The developed application provides context about the usage of cookies on a determined website. This feature is supported through the use of the Website Evidence Collector software. This tool displays its output information in a readable format, but most importantly, returns information about cookies and third-party requests for data. We will use the cookie gathering function to help understand which cookies are used. WEC is a tool that leverages the Puppeteer [28] NodeJS library high-level API to simulate and control a real browser. The necessary commands are provided by WEC and used on a terminal shell. The results will appear in the developed application as they are received.

The tool runs on any system that supports NodeJS and thus requires NodeJS to work. It presents results in different formats aside from human-friendly HTML such as JSON or YAML Ain't Markup Language (YAML).

The backend of the web application calls this tool through a spawned shell, as will be discussed in the next section, replacing the URL with the intended target:

website-evidence-collector [URL] –json –quiet –no-output

4.3 Express NodeJS

The backend of the web application uses Express for its ease of setup routing. Certain features might use an external application. Such an example is the cookies scanner feature. For this purpose, routes must be implemented that allow the web application to call the scanner.

The NodeJS project is organized between folders and the main file as shown in Figure 4.1.

```
/
├── /bin
├── /models
├── /node_modules
├── /public
├── /routes
├── /views
└── app.js
```

Figure 4.1: Express NodeJS project structure.

Folders such as bin and node_modules are used to contain the logic to start the project and the modules needed for its functions. The public folders contain all the publicly available resources, views contain all the pages and routes hold all the necessary routing for the server.

For the web application, most of the necessary logic will be present at the routes folder in a designated JavaScript file. The Figure 4.2 diagram shows the specific function of the scanning feature and its respective route. Each route's function is represented by numbers 1 and 2.

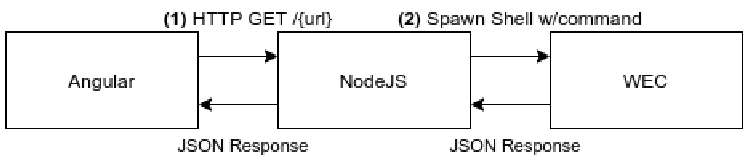


Figure 4.2: Route interactions

The route necessary to handle the cookie scanner and the respective 1 and 2 actions is shown in Listing 4.3 in a truncated form.

```
// (1)
router.get('/:_url', function (req, res, next) {
    let url = Buffer.from(req.params._url, 'base64').toString();

    // (2)
    exec(wecTemplate.slice(0, 27) + url + wecTemplate.slice(26,), (
   error, stdout, stderr) => {
        if (error) {
            //truncated error handling
        }
        if (stderr) {
            //truncated error handling
        }
        res.json(JSON.parse(stdout)['cookies']);
    });
});
```

Listing 4.4: Truncated scanner route.

On action 2, a shell is spawned, uses the received URL, and crafts the necessary command. The command is executed in this shell and returns the output. The HTTP response is made out of the 'cookies' section of the console output.

As of the writing of this document, only the dedicated route to the scanning of a URL is implemented. This is because this route is essential to the function of a feature in the web application.

4.4 Angular 11

The frontend of the web application makes use of the Angular 11 framework. The main advantage of using Angular 11 for the development is the opportunity to reuse certain page elements for pages that share the same functionality, and two-way data binding. The latter allows the sharing of information between a template and the component logic (see Figure 4.3). This property also permits that changes in the Document Object Model (DOM) can influence data on the component.

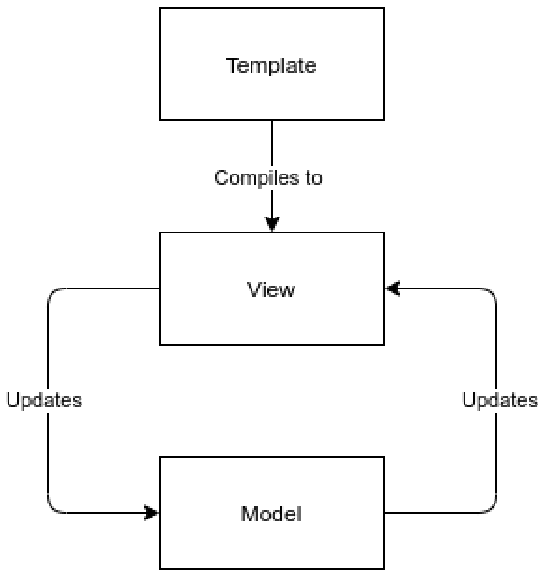


Figure 4.3: Two-Way Data Binding.

The main project is located in the app folder and it’s structured as follows in Figure 4.4.

```
/app
 ├── /model
 ├── /pages
 ├── /shared
 └── /styles
```

Figure 4.4: Main project structure.

The structure separates the page elements from the shared components. This choice allows for the shared logic of **shared** components and their persistence on the screen independently from whatever might be shown. Such elements might be the sidebar, header, and footer. Each component in an Angular project is comprised of, at least, three files (two, however, can be contained entirely within the Component file) – Component, HTML Template and Styles.

Components, depending on their complexity, must possess a module or be part of another. Modules aggregate similar logic or an architecture domain. This permits the importation and exportation of functionality between modules. Each module declares its main components and module imports that it may need. For instance, in the developed project, each page has its logic despite sharing certain components. So, each page also possesses its module and imports its needed modules. The only exception is the root module, that due to serving as the anchor for the rest of the application, also needs the routing file. This file dictates the routes for each page anchor component, for example, for the 'not found' page, scanner, and configuration pages. An example of the structure can be seen in Figure 4.5.

```
/app
 ├── app.module.ts
 ├── app-routing.module.ts
 ├── app.component.ts
 ├── /pages
 │    ├── /not-found
 │    │    ├── not-found.component.html
 │    │    ├── not-found.component.ts
 │    │    └── not-found.component.scss
 │    ├── /scanner-page
 │    └── ⋮
 └── ⋮
```

Figure 4.5: Component organization example

The example above shows the minimal elements for a component to be valid and their hierarchy with the main module. It's worth noting that pages such as the scanner possess their module.

Component Resources

The Angular framework supports the use of external modules. With this feature, we make use of the Angular Material Resources. These resources provide building elements for Angular and allow a more consistent look-and-feel and experience through the implementation of modules as wrappers for styles and functions.

Shared Components

Throughout the web application, certain elements are always present. Elements like a sidebar must always be available to allow the user to navigate the features. The shared components obey a set of positions defined in a base layout as shown in Figure 4.6. The only elements that are independent from this layout and define their positions by the current page are the footer and the dialog message with information about the configuration.

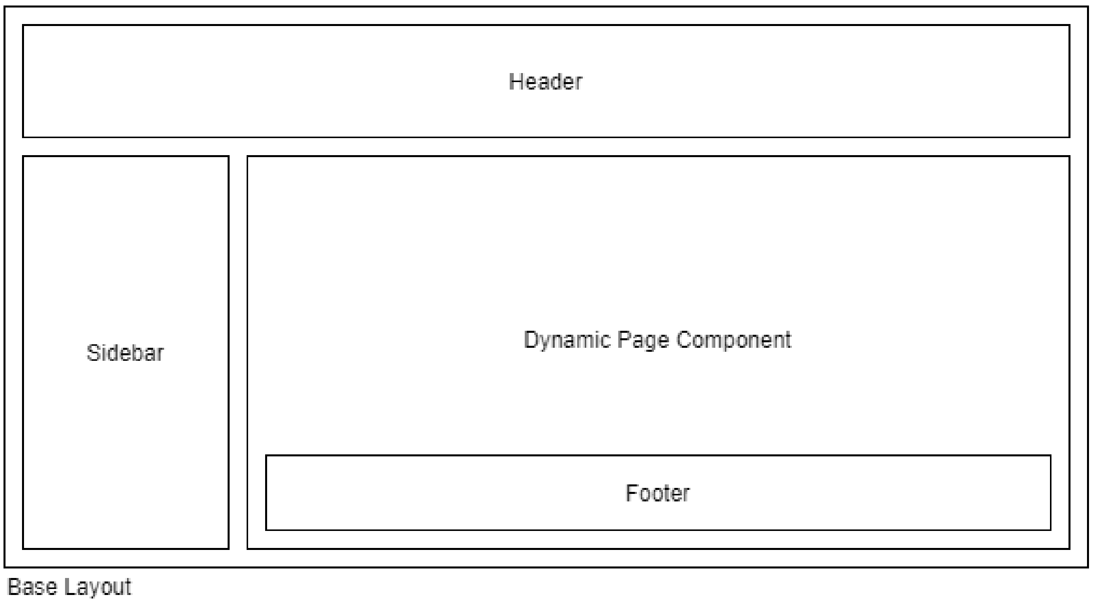


Figure 4.6: Base layout.

The header and base layout component are linked by function. The header has the button that provides the show and hide sidebar function. The sidebar component abstracts itself from this behavior since it needs only to be placed in the base layout.

The header has its own module to allow future features to be implemented and to export specific components. For example, by being responsible for the sidebar behavior, the header receives the current sidebar status from the base layout and emits an event change. This can be used to adapt the application in different ways such as showing or hiding the sidebar by "squeezing" the page space or making the sidebar appear on top of the page in case of a smaller window. The hierarchy is shown in Figure 4.7.

The shared components are anchored on the base layout, which in turn has a space

```
<router-outlet>
└─<app-[page]-page>
   └─<app-base-layout>
      ├─<app-header>
      └─<mat-sidenav-container>
         ├─<mat-sidenav>
         │  └─<app-sidebar>
         └─<mat-sidenav-content>
            └─<ng-content>
```

Figure 4.7: Base layout component hierarchy in HTML.

dedicated to receiving arbitrary components, namely the components that display the functions of the application. Through the use of the Angular HTML tag *<ng-content>*, any page that is navigated to can display its contents in it. For instance, the scanner page will display its component and logic in this tag, and the configuration page will do the same, all while not needing to reload the browser. When navigating to a page through the sidebar, the HTML template for that page is invoked. To leverage the dynamic page updates of Angular, each page merely needs to assemble its elements inside the *<app-base-layout>* tag. All the HTML elements within will be placed inside the *<ng-content>* tag of the base layout template. The reason behind the usage of *<ng-content>* instead of the tag defined in each page's templates is to allow any page to load its contents there. Assuming that we didn't need to account for different pages, we could forego *<ng-content>* for the specific HTML tag, for example, the use of the *<app-sidebar>* inside the mat-sidenav to insert the list of paths the user can navigate to. A visual representation of the shared elements can be seen in Figure 4.8.

Scanner Page

As discussed in Chapter 3, the scanner page is solely dedicated to the scanning and retrieval of cookie information as shown in Figure 4.9. The page is simple and easy to use with a single field for user input. This field receives the target URL the website owner intends to scan for cookies. Although WEC supports different URL formats, it allows the omission of elements such as the protocol. The application forces the inclusion of the HTTP protocol for simplicity's sake.

The target URL is base64 encoded in the Scanner Service and sent in the request URL. The service function expects an Observable type as it calls an external API for the

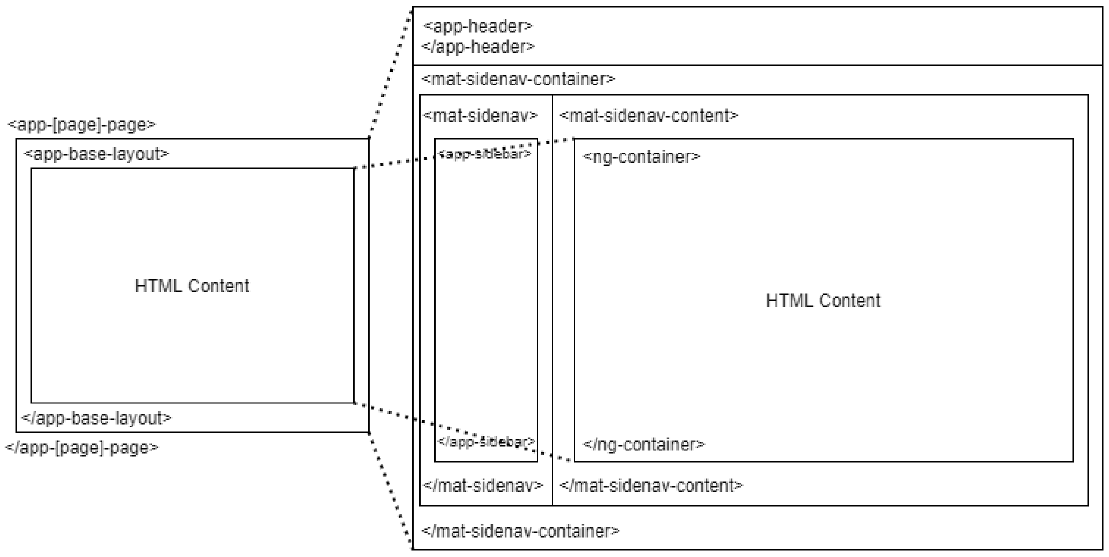


Figure 4.8: Visual representation of the placement of shared components.

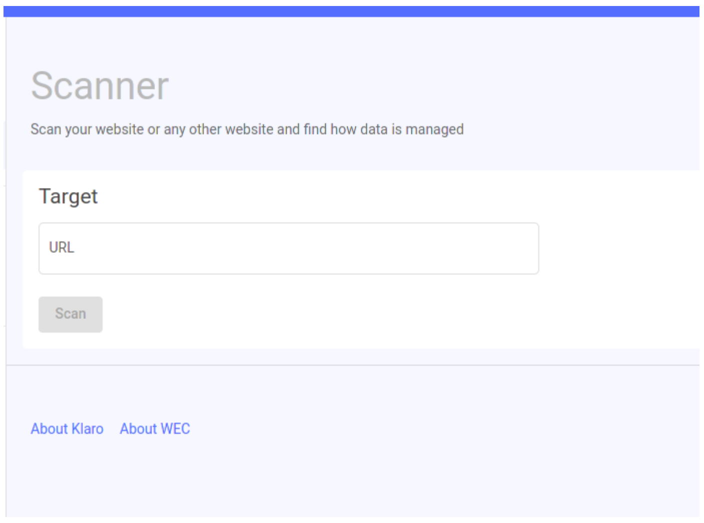
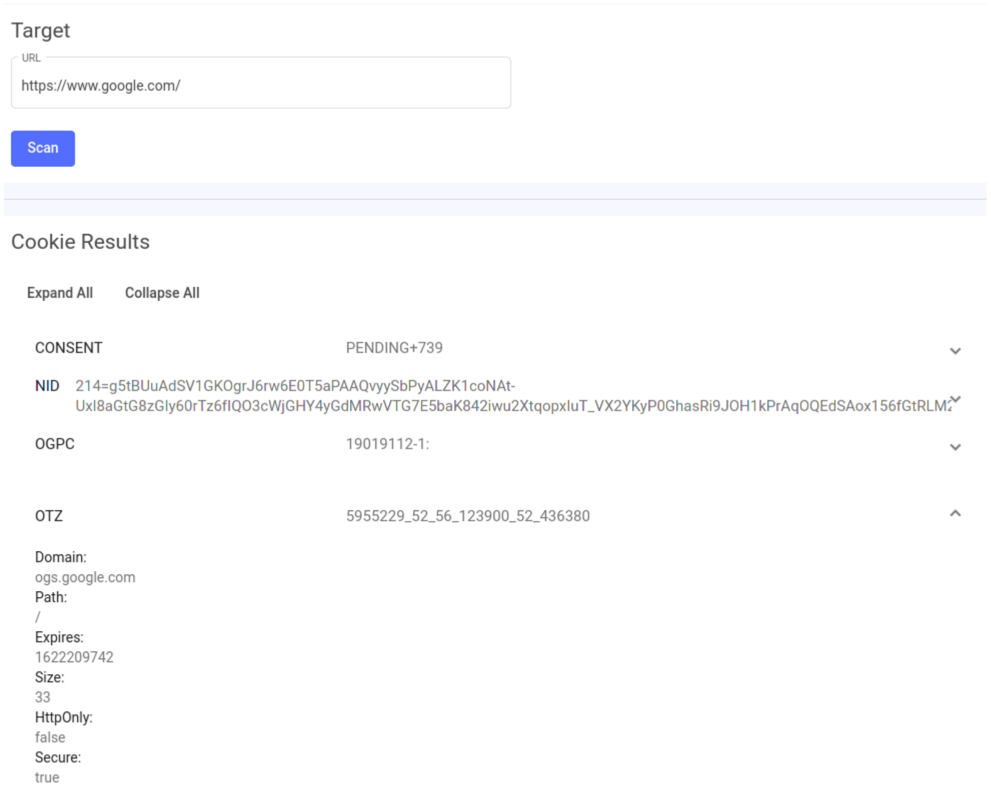

Scanner
Scan your website or any other website and find how data is managed
Target
URL
Scan
About Klaro
About WEC

Figure 4.9: Scanner page input field.

results. A request is formed and sent. The GET request is sent to the appropriate API route */scanner/'base64url'* as Listing 4.3 demonstrates. This action triggers the API to spawn a shell and execute WEC. The results are sent back in JSON format and presented to the user on a collapsible list as Figure 4.10 presents.

The list contains all the scanned cookies for the particular website and the information each cookie contains. The header of each element displays the name and value of the cookie. When expanded, the rest of the attributes, such as Http-Only and Secure



Target

URL

https://www.google.com/

Cookie Results

CONSENT	PENDING+739
NID	214=g5tBUuAdSV1GKOgrJ6rw6E0T5aPAAQvyySbPyALZK1coNAt-Uxl8aGtG8zGly60rTz6fIQO3cWjGHY4yGdMRwVTG7E5baK842iwu2XtqopxluT_VX2YKyP0GhasRi9JOH1kPrAqOQEdSAox156fGtRLM
OGPC	19019112-1:
OTZ	5955229_52_56_123900_52_436380

Domain:
ogs.google.com
Path:
/
Expires:
1622209742
Size:
33
HttpOnly:
false
Secure:
true

Figure 4.10: Scanner page cookie results.

flags, are shown. The HTML element that shows the list is hidden by Angular until actual results arrive from the realization of the request. This is achieved with the **ngIf* tag on the element that contains the list. If the results exist, the list is shown, otherwise, the element remains hidden. Assuming the existence of the results object, Angular provides the tag **ngFor* that simplifies the display of multiple elements by simulating an iteration.

*<mat-expansion-panel *ngFor="let cookie of cookieResults">*

With the usage of two-way binding, the scanned cookies can be displayed in the same manner as Figure 4.10.

Configuration Page

The Configuration page is composed of two main layers. The base, which defines the separation between the two actions the user can perform, and each of the actions, create a configuration or load one much like the Figure 3.6 roughly shows in two tabs.

Aside from these, several elements are shared between the two actions such as the configuration editor and certain information dialog boxes.

The **Create** configuration tab generates an entirely new Klaro configuration from scratch with some default values. However, this generated configuration should not be used as it lacks the specification of services that identify the cookies that need managing.

The **Load** configuration action tab gives a user the opportunity to load a preexisting configuration provided it's syntactic integrity is valid. The configuration (being itself a text declaration of a JavaScript variable) is loaded into the application using the box in Figure 4.11.

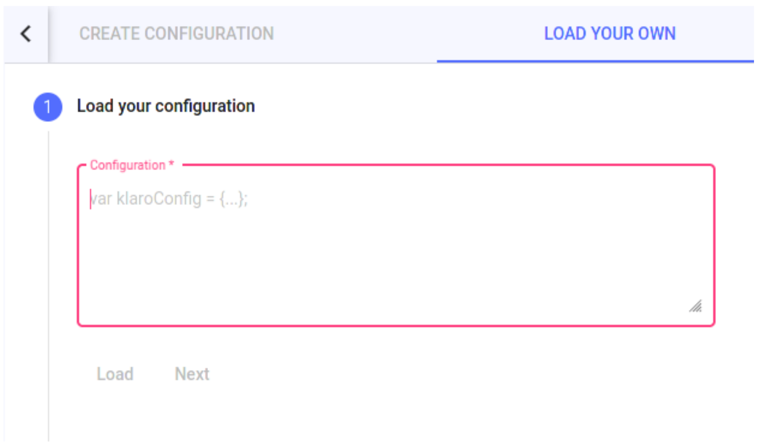
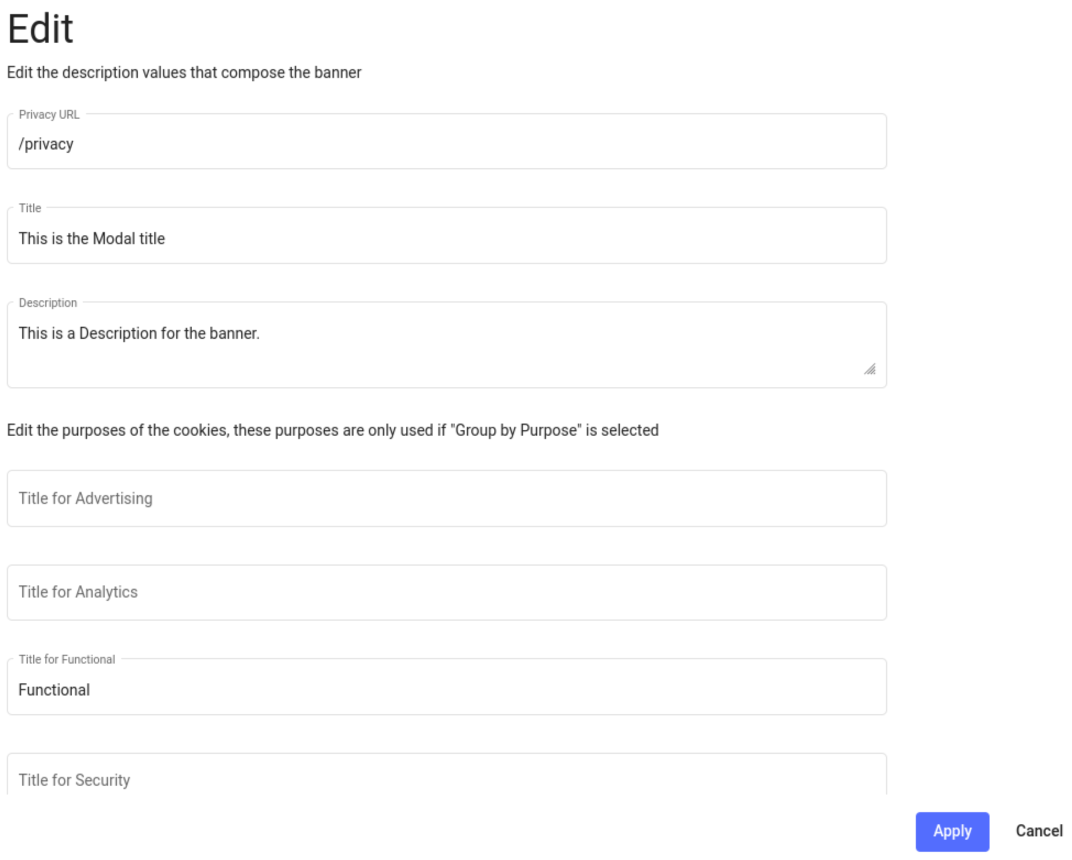


Figure 4.11: Import configuration input field.

The application proceeds with deep copying the inputted variable performing validity checks on each of the properties and ensuring the configuration is ready to be edited on the next stage. If one property fails to copy, the process is aborted and the user is prompted on the reason of the failure.

The **Edit** area is the same between the **Create** and **Load** tabs. Although it has the same functionalities for both tabs, we can edit a separate configuration for each. This allows a user to edit an entirely created configuration and edit an externally loaded one at the same time. The panel allows the modification of basic data such as the version, Klaro's cookie name and consent renewal expire date. Additionally, several checkboxes are provided to customize the behavior of the tool and how Klaro should behave and manage the cookies. An area for customization of the names of the four main cookie categories (Advertising, Analytics, Security and Functional) is also provided. The Services area is an area dedicated to the addition or deletion of the services and cookies a user intends on letting Klaro manage by giving the option to specify a regular expression to identify the cookie or service and it's purpose. Such customization options are given through small dialog windows as represented in Figure 4.12.



Edit

Edit the description values that compose the banner

Privacy URL: /privacy

Title: This is the Modal title

Description: This is a Description for the banner.

Edit the purposes of the cookies, these purposes are only used if "Group by Purpose" is selected

Title for Advertising

Title for Analytics

Title for Functional: Functional

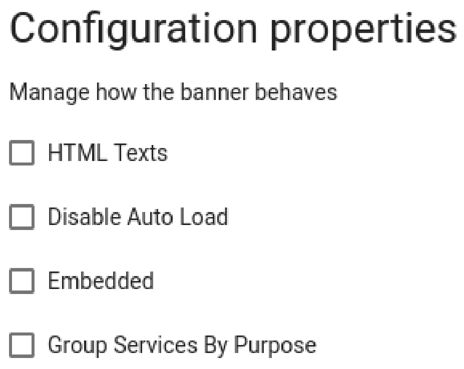
Title for Security

Figure 4.12: Edit categories dialog window.

The different Klaro functionalities can be configured through the aforementioned checkboxes. The checkboxes let the user specify certain behaviors of the CMP such as the automatic loading of Klaro upon page load, embedded rendering to allow the banner to be loaded inside other elements and even forcing the user to consent to the use of cookies before using the website. These options are available with short descriptions (e.g. as seen on Figure 4.13).

Constraints and Concerns

The Scanner page is responsible for the scanning of websites and the display of present


Configuration properties

Manage how the banner behaves

- [ ] HTML Texts
- [ ] Disable Auto Load
- [ ] Embedded
- [ ] Group Services By Purpose

Figure 4.13: Edit checkbox example options.

cookies. Currently, the analysis displays only the cookies an unauthenticated user would have set on its browser by the website. This constraint limits the full view of the used cookies and can miss some important data that can be collected post-authentication. This way, the functionality comes close to most of the already available market solutions for website cookie scanning. Like most scanners, cookies that are set only after consent is given are also not displayed as these require manual action.

Another intrinsic constraint of the scanner page functionality is the variability of the results upon several scans on the same target website. Although uncommon, this event can occur, mostly appearing as the lack of cookies when the website sets them on the browser. This can be circumvented by performing another analysis.

The Configuration page is responsible for the creation and edition of the configuration files used in Klaro.

The **Load Your Own** configuration feature reads an arbitrary JavaScript object provided by the user. This fact alone poses the issue of user-defined properties being loaded into the application and thus be an invalid configuration. Furthermore, cloning an object with arbitrary properties can force the app to malfunction and allows the user to add property fields not specified in the configuration file. To avoid this issue, each loaded configuration is deep copied into an already initialized object. The choice to opt for a deep copy is simply to preserve the object's fields per the only allowed fields. Other approaches were considered, but naive and shallow copies didn't preserve the non-enumerable fields of the JavaScript variable, didn't copy fields properly if the field itself was an object, and could, at times, still allow arbitrary fields to be added, such with the use of *'Object.assign()'* function. Additionally, the usage of the JSON library was considered, especially with the *.parse()* and *.stringify()* functions. However, these could not copy user-defined functions and regular expressions, the latter being key to the functioning of the application and configuration file.

4.5 Demonstration

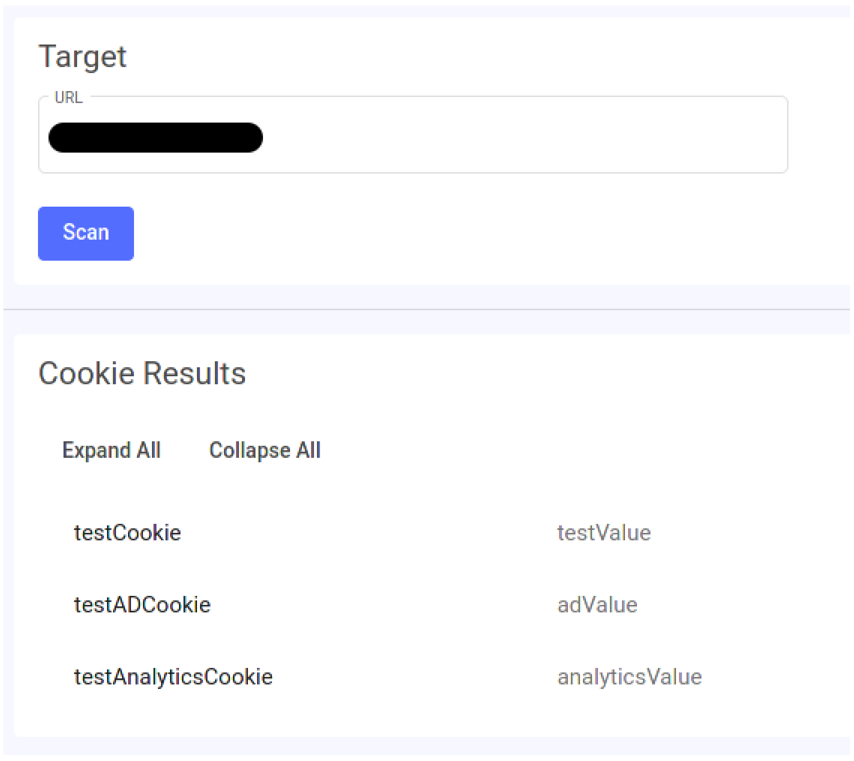
The application is used by a website owner with minimal knowledge of the website it is meant to manage. If that is not the case, the application supplies the tools for any website owner to know the necessary data and make accurate assessments on which cookies are used and what information they store on a visitor's browser. To demonstrate the application, we have a test website that sets 3 cookies on a visitor's browser. The website sets the cookies with 3 scripts as Listing 4.5 shows. Each script sets a cookie that can be classified as Functional, Marketing/Advertising, and Analytics/Performance respectively.

```
<!--(...)-->
<!--cookies-->
    <script type="application/javascript"
        src="/javascripts/functional_cookie.js"></script>

    <script type="application/javascript"
        src="/javascripts/AD_cookie.js"></script>

    <script type="application/javascript"
        src="/javascripts/analytics_cookie.js"></script>
<!--(...)-->
```

Listing 4.5: Script elements setting cookies.

A website owner can utilize the application's scanner feature to gather how many cookies are being set by these scripts. Such a process is as simple as Figure 4.14 exemplifies. It's worth noting that the scanner feature can't obtain all the cookie results from websites that necessitate prior consent to establish cookies.

With the gathered information, the website owner can formulate a configuration file that sets the needed properties for cookie management. In Figure 4.15, the website owner can choose to create or load its own configuration.



Target

URL

[redacted]

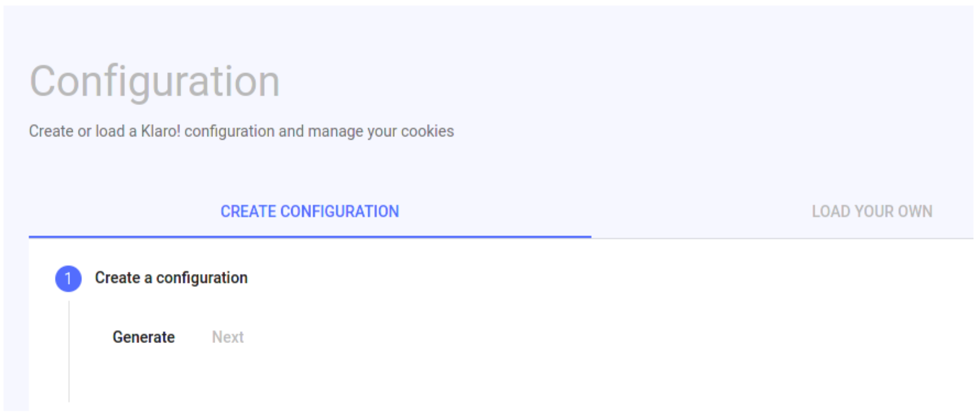
Scan

Cookie Results

Expand All Collapse All

testCookie	testValue
testADCookie	adValue
testAnalyticsCookie	analyticsValue

Figure 4.14: Scanner results on the example website.

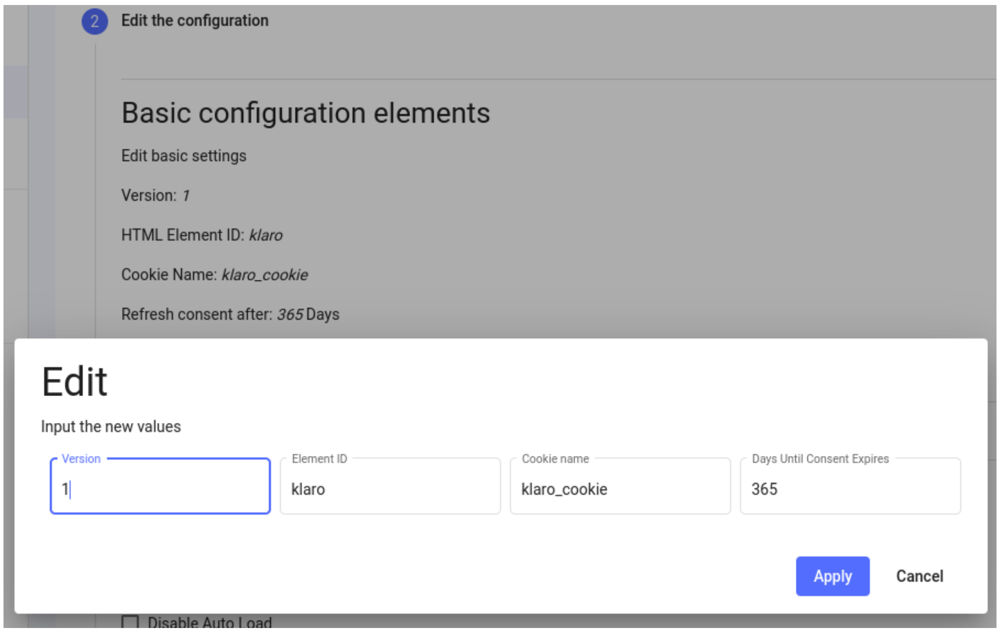


Configuration

Create or load a Klaro! configuration and manage your cookies

Figure 4.15: Generate or Load a configuration file.

Whichever path is taken, the same properties and fields are made available for customization. The website owner should pick according to the behavior it wishes to see reproduced later in the banner. Figure 4.16 and Figure 4.17 picture the elements and properties available.


2 Edit the configuration

Basic configuration elements

Edit basic settings

Version: *1*

HTML Element ID: *klaro*

Cookie Name: *klaro_cookie*

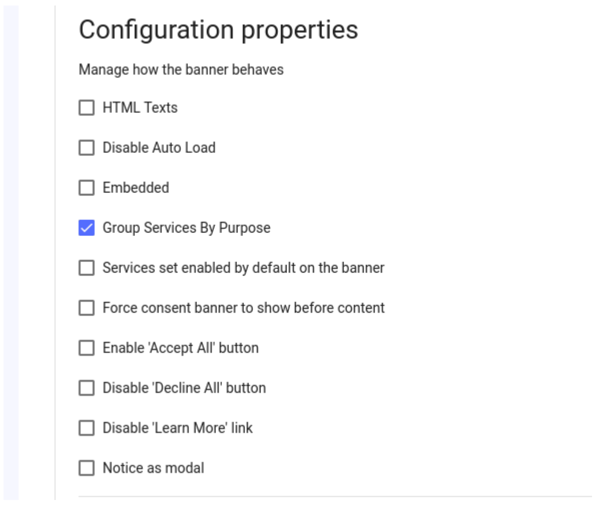
Refresh consent after: *365* Days

Edit

Input the new values

Version	Element ID	Cookie name	Days Until Consent Expires
1	klaro	klaro_cookie	365

Figure 4.16: Edit banner configuration elements.


Configuration properties

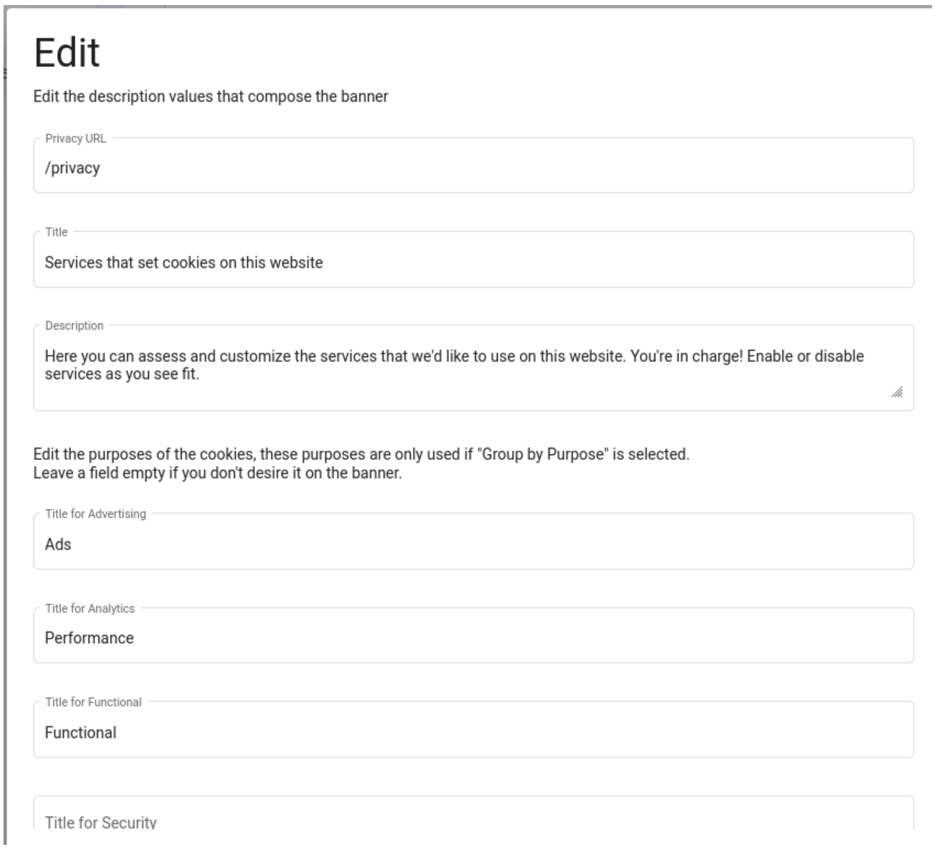
Manage how the banner behaves

- [ ] HTML Texts
- [ ] Disable Auto Load
- [ ] Embedded
- [x] Group Services By Purpose
- [ ] Services set enabled by default on the banner
- [ ] Force consent banner to show before content
- [ ] Enable 'Accept All' button
- [ ] Disable 'Decline All' button
- [ ] Disable 'Learn More' link
- [ ] Notice as modal

Figure 4.17: Edit banner configuration properties and behavior.

The next step is the customization of the text displayed on the banner. The available fields are shown in Figure 4.18. This step also allows the customization of the names of the categories for the cookies (this assumes that the option ”Group By Purpose” was selected previously, otherwise all the cookies will be listed without grouping).

Figure 4.19 shows an example of two services that define cookie-setting scripts. These are defined by the website owner according to the data known or gathered at the step shown in Figure 4.14. This entails the declaration of title of the service, name, description, purpose, and cookie identification (either the name of the cookie or a regular expression for similarly named cookies) and other requirements.



Edit

Edit the description values that compose the banner

Privacy URL: /privacy

Title: Services that set cookies on this website

Description: Here you can assess and customize the services that we'd like to use on this website. You're in charge! Enable or disable services as you see fit.

Edit the purposes of the cookies, these purposes are only used if "Group by Purpose" is selected.
Leave a field empty if you don't desire it on the banner.

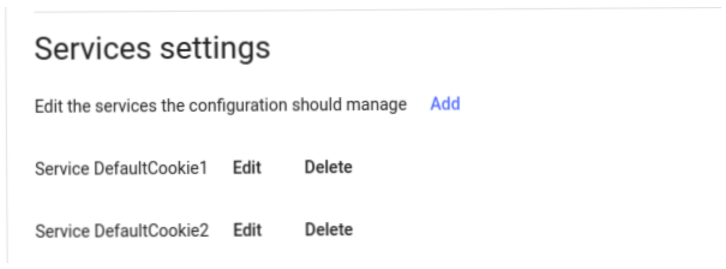
Title for Advertising: Ads

Title for Analytics: Performance

Title for Functional: Functional

Title for Security

Figure 4.18: Edit banner text and cookie category group names.



Services settings

Edit the services the configuration should manage Add

Service DefaultCookie1 Edit Delete

Service DefaultCookie2 Edit Delete

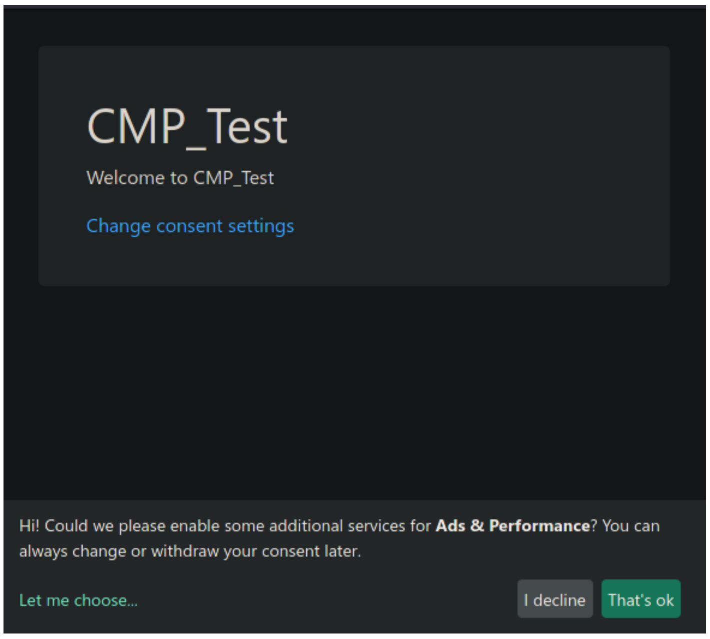
Figure 4.19: Edit script definitions.

Once finished, the configuration file can be exported and saved on disk. With the configuration creation process done, it is necessary to proceed with the required changes to the website. Firstly, its necessary to establish a HTML element with the same id as specified in Figure 4.16. Secondly, Klaro must also be imported alongside the configuration file we created in this process. Lastly, all of the identified scripts must change their HTML elements to both impede their automatic loading and to allow Klaro to recognized them according to the configuration file. All the changes are displayed in Listing 4.5.

```html
<!--(...)-->

<!--klaro configuration file-->
    <script defer type="application/javascript"
    src="/javascripts/klaroConfig.js"></script>

<!--klaro CMP-->
    <script defer data-config="klaroConfig"
    type="application/javascript" src="/javascripts/klaro.js">
    </script>

<!--cookies-->
    <script type="text/plain" data-type="text/javascript"
    data-src="/javascripts/functional_cookie.js"
    data-name="testCookies"></script>

    <script type="text/plain" data-type="text/javascript"
    data-src="/javascripts/AD_cookie.js"
    data-name="testADCookies"></script>

    <script type="text/plain" data-type="text/javascript"
    data-src="/javascripts/analytics_cookie.js"
    data-name="testAnalyticsCookies"></script>

<!--(...)-->
```

Listing 4.6: HTML after the necessary changes.

Lastly, the website should display a consent banner with all the intended preferences we set in this whole process. Any cookies that would have been set automatically are no longer loaded alongside the page, and a banner is shown first, as seen in Figure 4.20. The Figure 4.21 displays all the services managed by the Klaro CMP and, assuming we allow all to be used, we can see an example of these service-setting cookies on the browser in Figure 4.22 including the cookie Klaro sets to record consent.

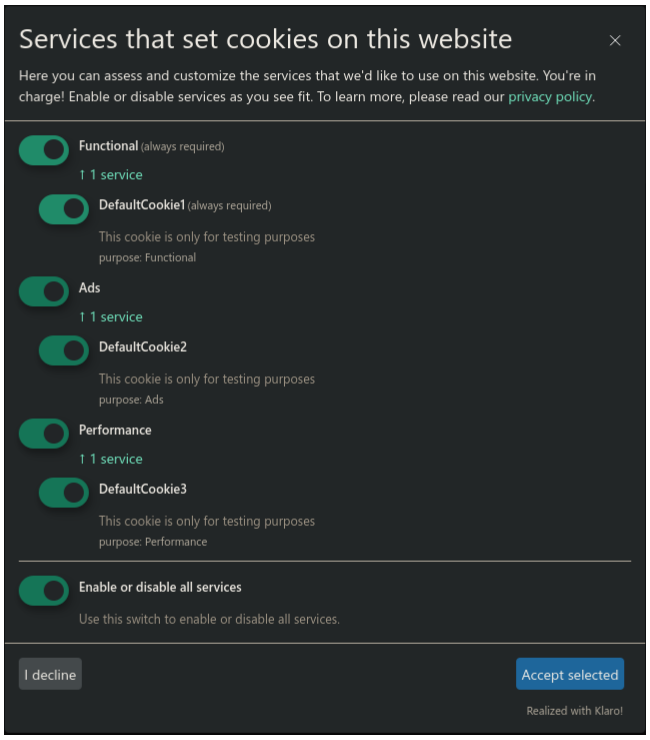


Hi! Could we please enable some additional services for **Ads & Performance**? You can always change or withdraw your consent later.

Figure 4.20: Banner popup.

4.6 Summary

The development of the solution had the emphasis on the informed creation of a configuration file that properly reflects the used scripts on a website. A major concern was bridging the difficulty of context acquisition and the setup of a valid configuration. The application provides a step-by-step process to set up key properties for presenting cookie usage information to website visitors. Furthermore, through the use of Angular as a frontend framework, the application has a consistent user experience that aggregates property configuration and website scanning. The latter, from a backend perspective, was separated due to security concerns and to allow later changes to its functions. The developed solution receives data collected from WEC and displays it on the Angular developed frontend. As such, an easy configuration file can be crafted and used in Klaro to display what cookies are present and how they are used in the website.



Services that set cookies on this website

Here you can assess and customize the services that we'd like to use on this website. You're in charge! Enable or disable services as you see fit. To learn more, please read our privacy policy.

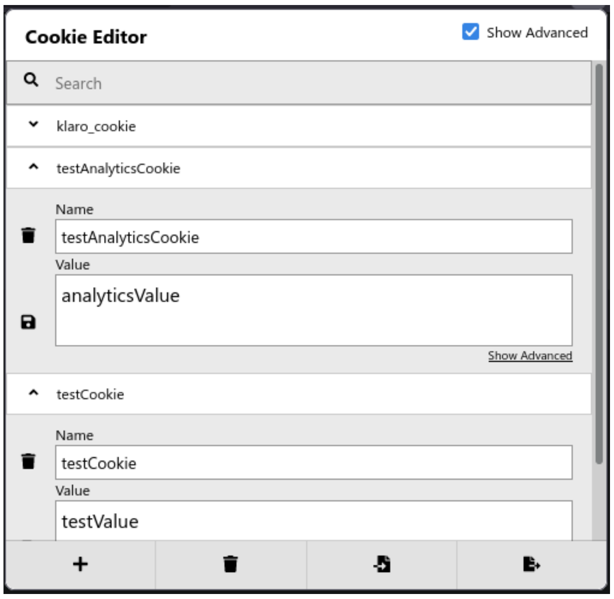
Functional (always required)
↑ 1 service
DefaultCookie1 (always required)
This cookie is only for testing purposes
purpose: Functional

Ads
↑ 1 service
DefaultCookie2
This cookie is only for testing purposes
purpose: Ads

Performance
↑ 1 service
DefaultCookie3
This cookie is only for testing purposes
purpose: Performance

Enable or disable all services
Use this switch to enable or disable all services.

Figure 4.21: Maximized banner with services grouped per category.



Cookie Editor

klaro_cookie

testAnalyticsCookie
Name: testAnalyticsCookie
Value: analyticsValue

testCookie
Name: testCookie
Value: testValue

Figure 4.22: Current cookies displayed with CookieEditor [7]

Chapter 5

Conclusions

This chapter is dedicated to the presentation of conclusions that can be formed from the fulfilment of this work. Finally, we conclude the thesis with the expected future work directions that can bring this project to its maturity.

5.1 Present Work

The goal of this thesis was the development of an application that accelerates the process of identifying and informing visitors of the data collected in a website through cookies. To properly understand this endeavor, we analyzed the problem from the very definition of cookies and their role in information transactions while a user visits a website. Firstly, a contextualization of where cookies are used and their usual function was made. This was necessary to understand why cookies are important and why they can become a chore to manage. To understand when cookies can become a problem, a proper description of each type was given, concluding with an analysis of how tracking could be nefarious and which cookies are responsible for said tracking. With the proper perception of the role of, most specifically, third-party cookies, we can infer that these require the most concern from an information security and privacy perspective, and should, therefore, be a major focus for any website owner, data controller, or data protection officer.

A cookie management system/platform automates the process of declaring which cookies were used and presenting the consent option to a visitor. We analyzed some of the available solutions and gathered which features are expected to be present.

CMPs often separate the feature of website scanning to specialized cookies scanners. This can hinder the ability to quickly perform a self-scan and immediately proceed with the declaration of cookies used. Although some of the options seemingly possess overwhelmingly more features than others, these are often gated behind paywalls with the free option offering very little in the way of providing a website owner with the basic tools it needs to properly manage cookies. The developed application seeks to provide the insight gathering functionality of a cookie scanner to the CMP proper. The latter was powered by Klaro but requires some advanced knowledge to use. The application creates the relevant configuration file with options easily accessible. In order for the application to make the process as quick as possible, the scanner is powered by WEC and provided a click away from the configuration editing page.

The goal was to bring the whole procedure, from beginning to end, close together as much as possible and not let the website owner lost amidst several different tools. The application sits neatly in the middle of the process, mostly between the insight gathering phase and consent gathering. It uses the knowledge a website owner had and gathered on his own website, creates the necessary components, and presents them on a banner.

The choice of framework for the development was made mostly on the exploration of web application functionalities such as two-way data-binding. Secondly, the framework also possessed interesting security features, namely, escaping any user input data that may be shown on the application protecting against some XSS attacks. The application fulfills the following, as proposed:

- Displays a consent warning message with granular choice through Klaro;
- Website scanning for cookies through WEC;
- Allow/Deny cookies based on user choice in the consent banner;
- Prevent the loading of cookies as per user consent.

The application, however, has its own limitations. Such limitations can be mitigated by fine-tuning features as it will be explained in the **Future Work** section. Limitations can be also from a commodity perspective, such as a contextualization feature to help categorize and identify the purpose of scanned cookies. A final summary and comparison of the developed application and studied solutions is presented in Table 5.1.

	Installation	Features	Cost	Open-Source
Osano CC Free	**	*	Limited Free Version	Yes (Free Version)
CookieBot	***	**	Limited Free Version	No
CookieYes	***	***	Limited Free Version	No
CookieFirst	***	**	Limited Free Version	No
OneTrust PC	***	***	Limited Free Version	No
CookieConsent	***	**	Free	Yes
Klaro	*	**	Free	Yes
Dev. Solution	**	***	Free	Yes

Table 5.1: CMP comparison table with the developed solution.

5.2 Future Work

To reach maturity, the application should add further functionalities and improvements to existing ones. The improvements should mitigate the flaws the current features have. After these are mitigated, new features can be properly added. Given the "toolbox" like nature of the application, a user access control functionality, although important, would serve mostly for features to be added after other key features are implemented.

Despite the main objectives of the thesis being met, some features require improvement:

- Configuration creation to Klaro integration should be expanded upon to further speed the process and with less manual interaction;
- Connect the Scanner page with the Configuration page by functionality as to allow scanned cookies to appear in the Configuration page and provide insight in the creation of a configuration file.

Furthermore, some features are to be added to give more context to results and analysis, and provide control over different configurations:

- Integration with an open-source database of cookies names and purposes to add context [31];

- Automate the detection of newly added scripts to a webpage and notify the website owner of the obsolescence of the current configuration
- Add user control to allow users to save desired configurations with different cookie definitions.

Lastly, the final work is to transition the application to a production environment and to be made available for perfect use.

References

[1] Adam Barth. HTTP State Management Mechanism. RFC 6265, April 2011. URL https://rfc-editor.org/rfc/rfc6265.txt.

[2] Lucas Birdeau, Clifford Yeh, Michael Peachey, and Kevin Hakman. Delivery of data and formatting information to allow client-side manipulation, 2002.

[3] Rob Bonta. California Consumer Privacy Act (CCPA). https://www.oag.ca.gov/privacy/ccpa, 2020. [Accessed February-2021].

[4] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259, December 2017. URL https://rfc-editor.org/rfc/rfc8259.txt.

[5] Aaron Cahn, Scott Alfeld, Paul Barford, and S. Muthukrishnan. An empirical study of web cookies. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee. doi: 10.1145/2872427.2882991.

[6] Cathy McKnight. Single page applications – why they make sense. https://www.digitalclaritygroup.com/single-page-application-make-sense/, 2014. [Accessed March-2021].

[7] Christophe Gagnier. Cookie-editor. https://cookie-editor.cgagnier.ca/, 2021. [Accessed July-2021].

[8] Comissão Nacional de Proteção de Dados. Nota informativa da comissão nacional de proteção de dados relativa à utilização de cookies. https://www.cnpd.pt/media/x2zdus50/nota-informativa-cnpd_cookies_20210625.pdf, 2021. [Accessed July-2021].

[9] European Commission. Regulation on Privacy and Electronic Communications. ec.europa.eu/newsroom/dae/document.cfm?doc_id=41241, 2017. [Accessed February-2021].

[10] Global Privacy Control. Global Privacy Control. https://globalprivacycontrol.org, 2021. [Accessed February-2021].

[11] CookieConsent. Cookieconsent. https://www.cookieconsent.com/, 2021. [Accessed February-2021].

[12] CookieFirst. Cookie consent management platform cmp. https://cookiefirst.com/, 2021. [Accessed February-2021].

[13] CookieYes. Free cookie checker. https://www.cookieserve.com/, 2021. [Accessed February-2021].

[14] CovenantSQL. Cookie scanner. https://github.com/CovenantSQL/CookieScanner, 2021. [Accessed February-2021].

[15] CyBot. Gdpr, eprivacy and ccpa compliant cookies — cookiebot cmp. https://www.cookiebot.com/en/, 2021. [Accessed February-2021].

[16] CyBot. Cookiebot functions. https://www.cookiebot.com/en/functions/, 2021. [Accessed February-2021].

[17] CyBot. Cybot - eprivacy - today. https://www.cybot.com/, 2021. [Accessed February-2021].

[18] David Temkin, Google. Charting a course towards a more privacy-first web. https://www.blog.google/products/chrome/building-a-more-private-web/, 2021. [Accessed July-2021].

[19] Ian Davis. What are the benefits of mvc? https://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/, 2008. [Accessed July-2021].

[20] Disconnect. Disconnect. https://github.com/disconnectme/disconnect, 2021. [Accessed February-2021].

[21] EDPS Information Technology. Website evidence collector. https://github.com/EU-EDPS/website-evidence-collector, 2021. [Accessed February-2021].

[22] EDPS Information Technology. Website evidence collector. https://joinup.ec.europa.eu/collection/free-and-open-source-software/solution/website-evidence-collector, 2021. [Accessed February-2021].

[23] Executive Vice-President Vestager. E-000274/2021. https://www.europarl.europa.eu/doceo/document/E-9-2021-000274-ASW_EN.pdf, 2021. [Accessed July-2021].

[24] Federative Republic of Brazil. General personal data protection law. https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm, 2021. [Accessed February-2021].

[25] Jon Fielding. Four differences between the GDPR and the CCPA. https://www.helpnetsecurity.com/2019/02/04/gdpr-ccpa-differences/, 2029. [Accessed February-2021].

[26] Ewa Gasperowicz. Send beacon data in Chrome 39. https://developers.google.com/web/updates/2014/10/Send-beacon-data-in-Chrome-39, 2019. [Accessed February-2021].

[27] GitHub contributors. Puppeteer. https://github.com/puppeteer/puppeteer/, 2021. [Accessed February-2021].

[28] Google. Puppeteer. https://developers.google.com/web/tools/puppeteer/, 2021. [Accessed February-2021].

[29] Sandi Hardmeier. The history of internet explorer. https://web.archive.org/web/20051001113951/http://www.microsoft.com/windows/IE/community/columns/historyofie.mspx, 2005. [Accessed February-2021].

[30] Ian Hickson, Google Inc., David Hyatt, Apple Inc. Html 5. https://www.w3.org/TR/2008/WD-html5-20080122/, 2008. [Accessed February-2021].

[31] jkwakman. Open-cookie-database. https://github.com/jkwakman/Open-Cookie-Database, 2021. [Accessed July-2021].

[32] Justin Schuh, Google. Building a more private web. https://www.blog.google/products/chrome/building-a-more-private-web/, 2019. [Accessed April-2021].

[33] KirstenS, OWASP. Cross site request forgery (csrf). https://owasp.org/www-community/attacks/csrf, 2021. [Accessed February-2021].

[34] KirstenS, OWASP. Cross site scripting (xss). https://owasp.org/www-community/attacks/xss/, 2021. [Accessed February-2021].

[35] KitProtect. Klaro! Documentation. https://heyklaro.com/docs/, 2021. [Accessed February-2021].

[36] KitProtect. Klaro! Homepage. https://heyklaro.com, 2021. [Accessed February-2021].

[37] Richie Koch. Cookies, the GDPR, and the ePrivacy Directive. `https://gdpr.eu/cookies/`, 2021. [Accessed February-2021].

[38] Glenn Krasner and Stephen Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *Journal of Object-oriented Programming - JOOP*, 1, 01 1988.

[39] David M. Kristol. Http cookies: Standards, privacy, and politics. *ACM Trans. Internet Technol.*, 1(2):151–198, November 2001. doi: 10.1145/502152.502153.

[40] Lou Montulli. Persistent client state in a hypertext transfer protocol based client-server system, Jun 1998.

[41] Kirsten E. Martin. Data aggregators, consumer data, and responsibility online: Who is tracking consumers online and should they stop? *The Information Society*, 32:51 – 63, 2016.

[42] Microsoft. Choose between traditional web apps and single page apps (spas). `https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps`, 2020. [Accessed March-2021].

[43] Lou Montulli and David M. Kristol. HTTP State Management Mechanism. RFC 2109, February 1997. URL `https://rfc-editor.org/rfc/rfc2109.txt`.

[44] Lou Montulli and David M. Kristol. HTTP State Management Mechanism. RFC 2965, October 2000. URL `https://rfc-editor.org/rfc/rfc2965.txt`.

[45] Mozilla. Using HTTP cookies. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies`, 2021. [Accessed February-2021].

[46] Mozilla and individual contributors. Navigator.sendBeacon(). `https://developer.mozilla.org/en-US/docs/Web/API/Navigator/sendBeacon`, 2021. [Accessed February-2021].

[47] Mozilor. Cookieyes. `https://www.cookieyes.com/`, 2021. [Accessed February-2021].

[48] State of California. California Consumer Privacy Act of 2018 (CCPA). `https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?lawCode=CIV&division=3.&title=1.81.5.&part=4.&chapter=&article`, 2021. [Accessed February-2021].

[49] Stefanie Olsen. Nearly undetectable tracking device raises concern. `https://www.cnet.com/news/nearly-undetectable-tracking-device-raises-concern/`, 2002. [Accessed February-2021].

[50] LLC OneTrust. All You Need to Know About Cookies — Cookiepedia. `https://cookiepedia.co.uk/`, 2021. [Accessed February-2021].

[51] LLC OneTrust. OneTrust PreferenceChoice. `https://www.onetrust.com/products/cookie-consent`, 2021. [Accessed February-2021].

[52] Osano. Cookieconsent by osano. `https://github.com/osano/cookieconsent`, 2021. [Accessed February-2021].

[53] European Parliament and the Council of the European Union. Directive on privacy and electronic communications (DIRECTIVE 2002/58/EC). `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32002L0058&from=EN`, 2002. [Accessed February-2021].

[54] European Parliament and the Council of the European Union. DIRECTIVE 2009/136/EC. `https://edps.europa.eu/sites/default/files/publication/dir_2009_136_en.pdf`, 2009. [Accessed February-2021].

[55] European Parliament and the Council of the European Union. General Data Protection Regulation (GDPR). `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679`, 2021. [Accessed January-2021].

[56] SqueezeMind. Determines if your website is not comply with eu cookie law. `https://www.cookiemetrix.com/`, 2021. [Accessed February-2021].

[57] Steven Clarke. Measuring api usability. `https://www.drdobbs.com/windows/measuring-api-usability/184405654`, 2004. [Accessed May-2021].

[58] Richard Stim. Connecting to Other Websites. `https://fairuse.stanford.edu/overview/website-permissions/linking/`, 2019. [Accessed February-2021].

[59] Andrew Stuart. Where cookie comes from. `http://dominopower.com/article/where-cookie-comes-from/`, 2002. [Accessed February-2021].

[60] Vinay Goel, Google. An updated timeline for privacy sandbox milestones. `https://blog.google/products/chrome/updated-timeline-privacy-sandbox-milestones/`, 2021. [Accessed July-2021].

[61] W3C. Html standard. `https://html.spec.whatwg.org/multipage/webstorage.html#the-localstorage-attribute`, 2021. [Accessed February-2021].

[62] Mike West and Mark Goodwin. Same-Site Cookies. Internet-Draft draft-ietf-httpbis-cookie-same-site-00, Internet Engineering Task Force, June 2016. URL `https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cookie-same-site-00`.