Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Development of a Chrome Extension for Personalized Filter Suggestions

Final Thesis

submitted for the academic degree:

Bachelor of Science (B.Sc.)

at

Hochschule für Technik und Wirtschaft (HTW) Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang *Angewandte Informatik*

1. Supervisor: Prof. Dr. Elena Schüler
2. Supervisor: Birol Aksu

Submitted by Sunan Regi Maunakea [s0566144]

22. August 2022

Abstract

Filtering resources should be one of the primary focus for users when searching for a product or an article online, in order to reduce the number of options available to them and find the desired item in seconds. This thesis examines the possibility to create a personalized filter suggestions tool using a google chrome extension. These suggestions are based on the user's most entered URL. The more often user visits a web page with its filter within the URL parts, the more likely the filter is suggested.

To properly identify the requirements and features needed for the development of the extension, a requirement analysis process is conducted. The extension allows users to select their most used filters within certain websites without numerous steps. In addition, it enables users to navigate through the list of the visited URLs inside the extension in a manner similar to file directory system. While traversing through the list of the path names, the extension assembles a new URL which the user can then visit.

Zusammenfassung

Bei der Suche nach einem Produkt oder einem Artikel im Internet sollte das Filtern von Ressourcen im Vordergrund stehen, um die Anzahl der verfügbaren Optionen zu reduzieren und den gewünschten Artikel in Sekundenschnelle zu finden. In dieser Arbeit wird die Möglichkeit untersucht, mit Hilfe einer Google Chrome Extension ein Tool für personalisierte Filtervorschläge zu erstellen. Diese Vorschläge basieren auf der vom Benutzer am häufigsten eingegebenen URL. Je öfter der Benutzer eine Webseite mit einem Filter in der URL besucht, desto wahrscheinlicher wird der Filter vorgeschlagen.

Um die Anforderungen und Funktionen, die für die Entwicklung der Extension benötigt werden, richtig zu identifizieren, wird ein Anforderungsanalyseprozess durchgeführt. Die Extension ermöglicht es den Benutzern, ihre am häufigsten verwendeten Filter auf bestimmten Websites ohne zahlreiche Schritte auszuwählen. Außerdem können die Benutzer durch die Liste der besuchten URLs innerhalb der Extension navigieren, ähnlich wie in einem Dateiverzeichnis-System. Beim Durchlaufen der Liste mit den Pfadnamen stellt die Extension eine neue URL zusammen, die der Benutzer dann besuchen kann.

Contents

List of Figures

Listings

1 Introduction

The way we access to digital information directly has fundamentally altered how information is retrieved, allowing us to create search engines that can significantly speed up the search process by allowing users to jump straight to the content they are interested in without having to navigate through convoluted systems. This instant gratification far outperforms previous methods of flipping through physical pages. But as digital information access has grown, so has the amount of information that is available on any given subject. As a result, finding the desired product would be difficult even with instant access via search.

The idea that the more options available, the better, and that human desire for choice is limitless are both prevalent in modern society. One study showed that the existence of choice increases motivation and enhances performance on doing tasks [27]. However, another study has shown that although having more choices might appear desirable, it may sometimes have negative effects on human motivation [11]. To resolve this issue, filters are introduced. Filters help users find informations faster. Rich information systems have recently begun to provide faceted navigation, which basically extends the idea of filters into a complex structure that attempts to describe all the different aspects of an object in order to maximize flexibility in retrieving information [26]. Nevertheless, using this more flexible and more useful tool requires multiple steps, expecially if users repeatedly search for the same information.

1.1. Objectives

The goal of this thesis is to design and implement a filtering suggestions tool, consisting of a client component, to circumvent the numerous steps involved in searching for an article or a product on a website. Thereby, a clear presentation and its development will be discussed as well as the technical background of the extension. The client component is implemented as a Google Chrome extension. To achieve a user-friendly extension, it is critical to provide the user with access to useful and clear information, so the extension to be developed will be compactly customized to one view, allowing the user to see their frequently visited websites.

1.2. Structure of the Thesis

The thesis is divided into four chapters. First, the descriptions of filtering, exploratory search and faceted search are provided. In addition, the anatomy of a URL as well as search and filtering usage in a URL are defined. Finally, the fundamentals of the technologies used are then explained in chapter 2. Further on, in chapter 3 the experimental methodology used for studies conducted with the Chrome extension is described. The extension's implementation and use in a practical situation are discussed in some detail in chapter 4 along with some lessons learned. Subsequently, the extension is evaluated and analyzed. Finally, problems are identified and an outlook for possible improvements of the extension is given.

2 Theoretical Basis

In the first section, this chapter describes the anatomy of a URL and how resources are searched and filtered using URL parameters. The second section illustrates the concept of a site searching. Furthermore, filters and facets are depicted within the scope of this thesis. The fourth section covers the definition of a browser extension along with its architecture. Finally, the framework for the extension's UI implementation is outlined.

2.1. URL

Uniform Resource Locator, or URL, is a compact string representation for a resource available via the Internet [2]. By offering a general identification of the resource's location, URLs are used to locate resources. These resources could be an image, a CSS file, an HTML page, etc. In practice, there are a few exceptions, the most frequent of which is a URL leading to a resource that has either relocated or vanished. After locating a resource, a system may carry out a number of actions on it, which can be described by phrases like "access", "update", "replace", and "find attributes". For each URL scheme, only the "access" method needs to be supplied. The scheme of the URL will be discussed further. Here is an example of an HTTPS URL: `https://example.com/filtre/popup.html`.

Uniform Resource Identifier, or URI, is a string of characters that, like the URL, identifies a resource on the internet by location, name, or both. It makes it possible to identify resources consistently. Since URI is the superset of URL, every URL is also a URI, but some URIs are not URLs.

2.1.1. Anatomy of a URL

A URL is made up of various components, some of which are required and others which are not [17]. The most important parts are provided in the following sections:

Scheme

The scheme is the first component of the URL, and it specifies the protocol that the browser has to use to make a request for the resource. In a computer network, a protocol is a predetermined procedure for data exchange or transfer. The most common protocol is HTTP which stands for Hypertext Transfer Protocol. Nowadays most websites use HTTPS protocol which stands for Hypertext Transfer Protocol Secure.

Authority

Character pattern ("://") separates the authority from the scheme. If the authority is available, the host and port are also included, with a colon in between:

- The host holding the resource is identified by its host name. Either a domain name or an IP address can be used as the host name. Although hosts and servers do not correspond one to one, a server offers services in the host's name.
- The technical gateway used to access the resources of the web server is indicated by the port number. It is typically excluded if the web server allows users to access its resources using the standard ports for the HTTP protocol. If not, it is necessary.

Path

The web client's desired access point is identified by the path as a specific resource on the host. The path component contains data, usually organized in hierarchical form, that, along with data in the non-hierarchical query component, serves to identify a resource within the scope of the URI's scheme and naming authority [1]. The first question mark ("?") or number sign ("#"), or the last URI character, mark the end of the path. A path is made up of a series of path segments that are divided by the slash ("/") character. For every URI, a path is defined, even though the path may be empty. Only when a URI will serve as the context for relative references must the slash character be used to denote hierarchy. For example, `https://example.com/filtre/dist/popup.html` has a path of `/filtre/dist/popup.html`, whereas the path to this URI <`foo://example.com?popup`> is empty.

Query String

The query component contains non-hierarchical data that, along with data in the path component, serves to identify a resource within the scope of the URI's scheme and naming authority [1]. The first question mark ("?") character designates the start of the query component, and the number sign ("#") or the URI's final character designates its end.

A query is commonly found in the URL of dynamic pages. and is represented by a question mark followed by one or more parameters. The query directly follows the host name, path or port number. For example, this URL was generated by Google when searching for the word "query":

```
https://www.google.com/search?q=query&rlz=1C5GCEM_enDE993DE993&oq=query&
aqs=chrome..69i57j0i512l4j69i60l3.938j0j7&sourceid=chrome&ie=UTF-8
```

This is the query part:

```
?q=query&rlz=1C5GCEM_enDE993DE993&oq=query&aqs=chrome.
.69i57j0i512l4j69i60l3.938j0j7&sourceid=chrome&ie=UTF-8
```

It is sometimes preferable to forgo percent-encoding those characters for usability reasons because query components are regularly used to store identifying information as "key=value" pairs, with a reference to another URI being one commonly used value.

Anchor

An anchor is a type of "bookmark" within the resource that instructs the browser to display the content located at that "bookmarked" location. For instance, the browser will scroll to the point where the anchor is set in an HTML document; in a video or audio document, the browser will try to find the time the anchor denotes. It is important to note that the part following the ("#") character is never included in the request sent to the server.

2.1.2. Search and Filter URL Parameters

Search and filter URL parameters are parameters or query strings that add information to a specific URL. A search or filter parameter facilitates the search for a specific phrase or keyword within search engine results. They include what is requested while excluding irrelevant content. Aside from the functions mentioned above, the most common use cases for parameters are tracking, pagination, site search, sorting and filtering.

2.2. Site Search

Providing a search function that searches your Web pages is a design strategy that offers users a way to find content [25]. Without having to comprehend or navigate the site's structure, users can find content by searching for particular words or phrases. On large websites, this may be a quicker or simpler method of finding content. A great "site search" function is specific to the website and not only constantly indexes the site to make sure that access to the most recent content is simple, but it also directs users as they navigate a website's page, assisting them in discovering new and interesting content. The best site search tools draw users by making it easy for them to find the information they need immediately and by compiling helpful data on the site's engaging products and articles.

2.3. Filters and Facets

Filtering refers to the process of limiting a search based on predefined categories. These categories are frequently broad and based on a single dimension of the product. This allows user to quickly narrow down a great number of products to a more smaller group for further search.

Filters are broad categories set forth by the business that remain constant across searches and are frequently applied in the background. *Nielsen Norman Group*[1] defines

[1]*Nielsen Norman Group* is a pioneer in the field of user experience. NN/g carries out ground-breaking research, educates and accredits UX professionals, and offers clients UX consulting. More information on `https://www.nngroup.com/`

filters as one such tool, which analyze a given set of content to exclude items that don't meet certain criteria [26]. For instance, a book retailer online might use the genre filter, with the four categories of fiction, non-fiction, romance, and thriller as options. The genre filter is activated and only displays fiction books on the results page when a website visitor clicks on fiction in the top navigation.

Facets, also known as facet filters, enable users to filter results by selecting values along different dimensions or facets [20]. It is widely used in e-commerce search engines and digital libraries where documents have rich metadata. Broad, universal filters do not allow for finding products and results in a specific, targeted manner. Faceted search is a more granular method that does.

The distinction between filters and facets is essentially one of degree, but it is a significant one. The ideal faceted navigation should offer numerous filters, one for each distinctive aspect of the content. In contrast to systems that only offer one or two types of filters, faceted navigation is therefore more adaptable and practical, especially for huge content sets. In addition to giving users a structure to help them understand the content space and provide them with suggestions on what is available and how to look for it, faceted navigation also offers a description of various aspects of the content.

A well-designed user interface includes filters and facets. By guiding the user to the best result quickly, contextual factors that differ depending on the product or category encourage a user-friendly experience. Despite the fact that faceted navigation system has clear advantages for users, it is substantially more expensive to design and maintain because each facet requires different metadata to be applied to both current and future content. By giving users more options to comprehend and manipulate, the additional power of faceted navigation increases interaction costs. Often, a straightforward filter is simpler to understand and apply.

2.4. Browser Extension

Browser extensions or add-ons are third party programs, that can extend the functionality of browsers and improve users' browsing experience [23]. A browser extension, as opposed to a standard web page, is created specifically for a given browser and uses that browser's extension API. It was necessary to choose a browser as a result. There are frameworks that try to make it feasible to create an extension for several different browsers at once. Although the caliber of these frameworks was unclear, it was decided that the expense of potential problems and additional time spent debugging in many browsers outweighed the benefits.

2.4.1. Chrome Extension Architecture

Extensions are built on web technologies such as HTML, JavaScript, and CSS. They run in a separate, sandboxed execution environment and interact with the Chrome browser [9]. They also have access to the APIs that browsers provide for tasks like XMLHttpRequests and HTML5 features on web sites. The following files can be found in an extension:

- A `manifest.json` file

- One or more HTML files
- Any other files such as CSS or JavaScript needed by the extension to run

The majority of extensions have a background page that contains their primary logic and state. They frequently also contain content scripts that can communicate with websites. Asynchronous message passing is used to communicate between the background page and the content scripts. Additionally, extensions can save data via `localStorage` and other HTML5 storage APIs.

2.4.2. Manifest Files

A `manifest.json` file is required for each extension. It includes crucial information about the extension, such its name, version, scripts used for its content, minimum Chrome version, and permissions. The content-scripts field was the most crucial one for this expansion. Each study and content-related webpage need its own content script. Each one was defined in the scripts column, which also mapped each one to the appropriate URLs.

2.4.3. Content Scripts

Content scripts are JavaScript files that are used on websites to add new functionality. They have full control to modify the entire web page because they can directly access the DOM of these web sites. The content script is injected into a tab when the web page is loaded, and runs in the same process space of the renderer of the web tab and can thus access its DOM objects. Injected content scripts in a tab can only communicate with the extension core via Chrome's IPC [13].

2.4.4. Service Worker

In Manifest V3, the Chrome extension platform shifts from using background pages to using service workers. A service worker is a script that your browser runs in the background, distinct from a web page, opening the door to features that don't need a web page or user input [8]. With the aid of this technology, users can enjoy native-like features on the open web, such as push notifications, robust offline support, background synchronization, and "Add to Home Screen". Service providers drew some of their inspiration from the background pages in Chrome Extensions, but improvements were added for the web.

Service workers are specialized JavaScript assets that act as proxies between web browsers and web servers. They aim to improve reliability by providing offline access, as well as boost page performance [8]. Websites are gradually improved by service workers through a lifetime akin to that of platform-specific programs.

2.5. React

React is a product of Facebook's engineering team, which is a JavaScript framework for creating user interfaces [5]. Because of its simplicity and straightforward but efficient development process, React is quite well-liked in the developer communities. Interactive user interfaces are simpler to develop with React. It effectively updates by accurately drawing each state's view's constituent parts, and it updates the application's data [10].

React's primary goal is to deliver the best rendering performance. Its strength stems from the attention paid to individual elements. Rich UI designs are found to be simple for developers to create when using reusable components. React incorporates with View part from MVC model [15]. One Way dataflow is implemented in React to make it simpler than conventional data binding. React uses a virtual DOM to provide programming that is less complex and executes more quickly. It makes use of composition to create intricate user interfaces out of simple building blocks known as Components [3].

2.5.1. Component and Props

Components are self-contained and reusable pieces of code. They perform the same function as JavaScript functions, but they operate independently and return React elements that describe what ought to be displayed on the screen. They are classified as Class components or Function components. To describe what should be rendered, JSX, a combination of HTML and JavaScript, is used. See subsection 2.5.8 for a more in-depth explanation. These functions accept arbitrary inputs, called `props`, and store property values inside a `state` object.

Immutable data supplied into a component during development is known as `props`, or properties. Because a component can function and seem differently depending on the properties supplied into it, `props` enable React components to be flexible and reusable.

Using props, data moves down the component tree in React. Callback functions are supplied as `props` so that a child component can communicate with its parent. Callback methods and other data must be passed down numerous layers in large React apps due to the component tree's depth. *Prop-drilling*[2] is a technique that results in tightly connected components and a less maintainable program. This is one of the reasons why complex React apps require architectural patterns.

2.5.2. State and Lifecycle

State allows an application to manage changing data. It is defined as an object in which key-value pairs specify various data that should be tracked in the application. The `state` is optional; components without a state are referred to as presentational components. Components with a state are referred to as stateful components.

The built-in functions known as lifecycle methods are called whenever a state or prop changes, before rendering or after rendering a component, component is destroyed, or both.

[2] *Prop-drilling* is the unofficial term for the process of delivering data to a deeply nested component by passing it through a number of nested children components.

2.5.3. Class-based components

By extending the `React.Component` class, a class-based Component can be produced. Class-based component's state is modified via the `setState()` method and retrieved via the `this.state` property. Using `setState()` function causes the component to be re-rendered, which is not the case when directly modifying the state. Class-based components include lifecycle methods that can be used to create behavior with a higher degree of complexity. In addition, a render method is required to return JSX elements.

2.5.4. Function components

The term function component refers to a pure function that takes `props` as input and outputs a JSX element. React Hooks are used to give the function component the same access to state and lifecycle methods as class-based components. For more information on what React Hooks are and how they work, refer to subsection 2.5.5. In comparison to class-based components, using function components with hooks might make React applications smaller and more manageable. There are a couple reasons why function component is preverable [19]:

- Since function components lack state and lifecycle-hook, development time is reduced, readability is improved, testing and debugging are simplified, and components can be reused. Function component is a straightforward JavaScript function.
- As the compilation time for function components is shorter than that of class components, performance will increase.
- Utilizing a function component eliminates the need to plan for the component's separation into a container and a regular UI component.

2.5.5. React Hooks

React Hooks allow the same functionality found in Stateful Components to be used in Functional Components, making them ideal for more complex functional components. As opposed to using classes, React's state and lifecycle functions can be accessed through the use of "hooks", which are essentially functions that allow the management of the logic within a component into reusable, isolated units. There is no state in a function component by default, but it is possible to maintain it with the help of the `useState` React hook. There is no restriction on the number of times any hook, including `useState`, can be used within the same component. In most cases, the word "use" will be the first part of a hook's name. The importance of adhering to naming conventions remains even though React does not enforce them.

Though function components lack native support for lifecycle methods, these can be implemented with the `useEffect` hook. The `useEffect` hook will automatically trigger a function to be executed whenever the component is re-rendered; however, this behavior can be modified to execute only when certain conditions are met.

The `useContext` hook, which will be described in subsection 2.5.7, is another React hook. In addition to built-in hooks, custom hooks can be made, allowing functionality to be reused throughout components.

2.5.6. Virtual DOM

React creates a new View for each component in the application based on their immutable states and props, and any changes to the states or props trigger a re-rendering of the View. As a result, the Views can be anticipated and even tested. Re-rendering views by replacing the DOM with a new version is a time-consuming process that can negatively affect the user experience because it results in the loss of the user's scroll position and input data. Using a Virtual DOM, React provides a solution for this [3].

By creating a new virtual DOM subtree whenever data in the application changes and comparing it to the previous subtree, views that need updating are redisplayed efficiently without causing other DOM nodes to break. To bring the virtual and actual DOMs into sync, React calculates how few DOM modifications are required. Once queued, the JavaScript DOM manipulations can be executed in bulk rather than one at a time, saving valuable processing time. To achieve the various states, developers use JSX to specify how components should be rendered, and React takes care of the corresponding DOM manipulation.

2.5.7. Context API

By utilizing the Context API, data can be passed around between React components without resorting to the tedious practice of prop-drilling. Multiple components in the component tree can share the same `Context` object created with the `createContext` method of the React library. Every `Context` object comes with a `Provider` React component that allows consuming components to subscribe to context changes [21]. When React renders a component that subscribes to this `Context` object, it will use the closest matching `Provider` above it in the tree to read the current context value. Data is added to the `Context` object and made available to its dependent components using a `Provider`. The `Provider`, which has a single property named value and accepts variables, functions, or objects as values, defines the `Context`'s data.

By placing child components inside the `Provider` component, the `Context` data can be accessed from any of them via a `Consumer` that is also present on the `Context` instance. A `Consumer` is a React component that subscribes to context changes [21]. The application uses a variety of contexts for various purposes rather than being restricted to a single instance.

By inheriting its parent's callback function, a child component can modify the parent's state using the Context API. According to the official documentation, `Context` is used to exchange data that is considered global for the React component tree, such as the current user's authentication status, the current application's color scheme, or the user's preferred language. Using context can limit the reusability of components if they need data provided by another part of the system. Data for a functional component wrapped in a `Context.Provider` can be accessed via the `useContext` hook, which acts as the `Consumer`.

2.5.8. JSX

A React extension called JSX makes it simple for web designers to change the DOM using straightforward HTML-style code. Additionally, as all current web browsers are supported by React JS, JSX is interoperable with any browser platform.

Most people find it helpful as a visual aid when working with UI inside the JavaScript code. It also allows React to show more useful error and warning messages [22]. Among the benefits of JSX is the fact that it simplifies and accelerates the template-writing process for HTML-savvy users. Performance improves when code is compiled into JavaScript [19].

3 Methodology

The initial idea of the thesis is first defined in the following chapter. In the second section, we look at some of the comparable programs that are already on the market. The third section discusses the resulting design considerations necessary for implementing the extension. The fourth section covers the software's design in light of the requirements. The decision to implement the technologies is described in the final section.

3.1. Design Concept

This thesis' original idea is to add an additional feature to Marta's customer-facing platforms. *Marta*[1] is currently best described as a marketplace between families requiring 24-hour care and caregivers. 24-hour care can be defined as a period of time spent living in the same household as the individual requiring care. This means that the primary responsibilities of caregivers are basic care and household chores. In addition, they provide assistance to the care recipient's relatives so that they can participate in activities of their choosing. Marta, a marketplace that connects families and caregivers, competes with more traditional agencies, where it can take days or even weeks to find a family for a newly registered caregiver or the other way around.

To compete in this rapidly growing industry, Marta would need to improve their product as an expanding startup that has gained a large number of users in recent months. One way in which Marta can provide a superior experience for both caregivers and families is by accelerating the matching process. The caregiver and family inquiry forms are intended to collect as much relevant information as possible for use during the matching phase. The matches are then generated by the Berlin and Romanian teams. To streamline the process, the Marta technical team devised a "matching score" by which potential matches are ranked. The family is then presented with a number of caregiver profiles with high matching scores. Filtering capabilities are provided to expedite the search process. This includes the caregiver's earliest start date, German proficiency, disease experience, etc.

To meet the requirements of its users, Marta must implement a continually enhanced, user-friendly filtering capability. An example of a user-friendly filter would be to provide quick filter suggestions that the user can click once to display the desired results, as opposed to allowing users to repeatedly select the same filter manually. To determine which filter suggestions are the most beneficial, it is necessary to identify frequently used filters.

Within this realm of thought, the question of how to enhance the existing filtering functionality for a better user experience arose. An early idea was to create quick

[1]*Marta* can be accessed through `https://www.marta.de/`

filters within the application that would suggest to users which filters they used most frequently. For instance, if a family is seeking a caregiver with level three German language proficiency, the German filter would be activated. Each filter application will be recorded in a relational database. The three most frequently used filters will be displayed as quick filters for all users to activate with a single click. In addition, the user can combine these quick filters to further narrow the search results.

The first issue with this concept was that it collected and stored filter usage data from every user without providing specific information about who used the filter and when. Each family has distinct search criteria. Therefore, the first concept would not be as advantageous to users. Additionally, we must remember that this strategy will only benefit those who actively use the platform. The second concern was that if filter recommendations were personalized for each user, the amount of data maintained would quickly grow to be quite substantial. It would be an inefficient use of storage space, especially as the number of filters and users increases. Not only would we develop quick filters for a single page, but also for multiple. And each would necessitate the same quantity of storage space.

The idea was then elaborated upon, and a decision was made. Instead of integrating these quick filters directly into the platform, it would be preferable to create an extension that allows for even more personalized filter suggestions. The aforementioned problems can be resolved by creating an extension. The information that will be saved is user-specific and is stored in the browser of the end user, so the filter suggestions are not affected by the actions of other users. Moreover, search and filter is a common web design technique. It would benefit not only Marta's platform, but also Lacoste, Nike, Adidas, and other e-commerce websites. Moving it to an extension would provide users with the option to install it. And because this is a distinct capability with a different software architecture than Marta's primary platform, a new repository or codebase is created to make it easier to differentiate and manage.

3.2. Existing Softwares

This section covers the currently available applications or extensions that are similar but do not completely solve the aforementioned problem. There exists a considerable amount of softwares on manipulating URL parameters but none that provide quick filter suggestions.

Query params

Query params[2] is an extension for Chrome that gives users the ability to read and write URL query parameters for the tab that is currently active through a graphical user interface. The size of this extension is approximately 85 KB, which is considered to be quite small for an extension of this kind. However, this extension has a purpose that is quite distinct from ours; rather than storing URL parameters, its primary objective is to manage them.

[2]*Query params* extension is accessible on: `https://chrome.google.com/webstore/detail/query-params/jgacgeahnbmkhdhldifidddbkneahmal`

Easy URL Editor

Another Chrome extension for modifying URL parameters is *Easy URL Editor*[3]. The goal of this extension is to make it easier to edit URL parameters for long URLs. This extension allows users to view each URL parameter, add new ones, and delete existing ones. Although they both serve the same purpose, this extension is comparatively larger in size (207 KB) than the previous one.

store-params

This software, unlike the previous two, is not a browser extension, but rather a vanilla JavaScript library. *Store-params*[4] is lightweight and used for storing URL parameters in cookies, `localStorage` or `sessionStorage` with flexible configuration options. Although there are advantages on using this library, the functionality remains limited. Because this is only a library, it must be integrated with an application to function. It serves the purpose of storing the URL parameters, but not modifying them.

ItemsJS

ItemsJS[5] is another JavaScript library that aims to perform fast searches on JSON datasets. This library differs from the other three mentioned above in that it does not use URLs and instead relies solely on JSON datasets. It is primarily used for data classification of companies, products, publications, documents, and so on. This dependency-free library offers faceted navigation as well as custom full-text search. Since the goal of the thesis is to create a universal solution for every website, this library remains briefly addressed.

Although softwares have been released by many developers, this problem is still insufficiently explored. A new approach is therefore needed for the goal of this thesis.

3.3. Requirements Elicitation and Analysis

The extension is used to enhance the user's experience when searching for a desired product or service on an e-commerce website. The plan is to integrate a browser extension into the user's browser that will record the host name or domain of the visited website's URL along with the corresponding parameters. When sufficient records have been collected, the extension will display a list of domain parameters. These parameters will be separated from path and query parameters. The user is then able to navigate through the path parameters in the same manner as they would a file system. When a path containing parameters is reached, the query parameters and the number of times they are referenced in the URL are displayed. Below the list is an input field and a button; the input field is populated based on the selected parameters

[3]*Easy URL Editor* extension is accessible on: `https://chrome.google.com/webstore/detail/easy-url-editor/kojpdadnbbicdfgfadonheclfpcjpiah`

[4]*store-params*'s GitHub repo is accessible on: `https://github.com/livechat/store-params`

[5]*ItemsJS*'s GitHub repo is accessible on: `https://github.com/itemsapi/itemsjs`

and as the user navigates through the list. When the "Navigate" button is clicked, the full-path URL is loaded in a new tab.

The third usability heuristic for user interface design by Jakob Nielsen is user control and freedom. The following statement is made by virtue of this principle:

> Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo [18].

In light of this, the user should be able to return after proceeding. A close button should be included to make it obvious to users how to close the extension. To expand the definition of "control," users should also be able to customize; they should be able to select which query parameters are saved and which are not.

These prerequisites and considerations result in the following requirements for the extension, which is to be developed within the scope of this thesis. These requirements are classified into functional and non-functional requirements.

3.3.1. Functional Requirements

A requirement is called functional if its underlying need is functional, i.e., it relates to information processing objects (data, operations, behavior). In other words, functional requirements are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations [24]. The following functional requirements are defined for the extension:

- FR-01: User can see how many times each query parameters for each host name and path name are used
- FR-02: User can see a list of paths and parameters based on their history
- FR-03: User can assemble a URL query string by selecting a path or a parameter
- FR-04: User can navigate to the URL with the assembled query string
- FR-05: The extension saves the host name, pathname and query parameters of the URL
- FR-06: The extension stores the URL attributes each time a page is loaded
- FR-07: User can define which query parameters should be excluded in the options page
- FR-08: User can remove a query parameter count by clicking on remove icon
- FR-09: User can delete all the filter from the options page
- FR-10: User can reset the configuration from the options page

In addition to functional requirements, non-functional requirements are also defined as follows.

3.3.2. Non-Functional Requirements

A requirement is called non-functional if its underlying need is a non-objective property. In other words non-functional requirementes are constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc [24]. Typically, it applies to the system as a whole, rather than to specific features or services. The following non-functional requirements arise for the extension to be developed:

- NFR-01: UI components must be implemented using React.
- NFR-02: TypeScript must be used to implement dynamic functions.
- NFR-03: Data must be saved on the end-user's Chrome storage.
- NFR-04: User's configuration from the options page must be saved on the end-user's Chrome storage.
- NFR-05: Both the external components used and the developed component itself must be well documented.
- NFR-06: The individual external libraries must be easy to update.
- NFR-07: Modularization must be taken into consideration throughout development so that individual modules and/or components can be easily reused, expanded or changed.

While functional requirements are perceived as such by the user, non-functional requirements are implementation details that remain largely hidden from the user.

3.4. Software Design

As the requirements are analyzed, the software's design can now be determined. This section discusses the structure and design patterns of the extension in order to meet the listed requirements. Software development requires a software architecture. It provides a framework for how the software should be constructed, and the decisions made at this stage are essential for the development process moving forward. In light of this, a software architecture diagram is provided.

Figure 3.1 depicts the components of the software and their interactions with one another. From the software architecture diagram, it is conceivable that there are two major components: Extension core and Chrome APIs. An extension core consists of a user interface and service workers. Content scripts are not displayed, because they are not required for the development of the extension. The extension core would interact with Chrome APIs to persist as well as fetch information on `chrome.storage` needed for concluding the most used parameters.

The extension's user interface consists of the pop-up and the options page. The pop-up communicates with `chrome.storage` API to obtain the available filters for the URL's host name, which is retrieved using `chrome.tabs` API. Similarly, the options page uses `chrome.storage` API to get the configuration object rather than the filters.

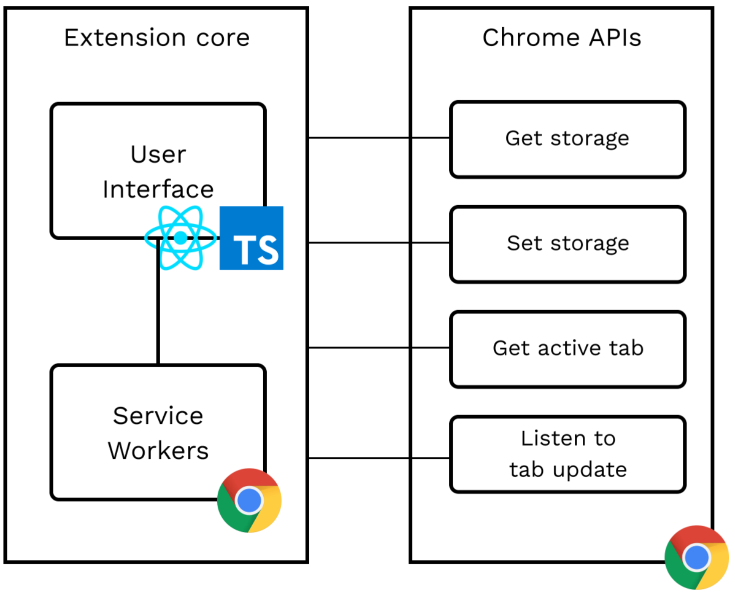


Figure 3.1: *Software Architecture Diagram*

As mentioned in subsection 2.4.4, a chrome extension has a background script called service worker. The service workers interact with `chrome.tabs` API to update the filter object on `chrome.storage` whenever a tab is updated.

3.5. Technologies

Software architecture does more than just define the software's structure; it can also provide insight into what technologies may be required to build the software. The implementation browser decision will be presented in the next section. The implications of this change on various data storage mechanisms are discussed in the second section.

3.5.1. Choice of Implementation Browser

The selection of a browser was based on a number of factors. The first was the browser's usage rate. This is significant since anyone who doesn't already have the necessary browser will need to download and install it. Drop-outs due to the installation process or being unfamiliar with a new browser can significantly raise the cost of conducting the survey. The ease of use of the API and simplicity of implementation were additional crucial criteria. The extension programming process should ideally only need a basic understanding of the extension API. The capabilities of the API offered by the browser was another factor considered. The extension needs to:

1. Store a big amount of data on the client side
2. Read URL - so path and query string can be passed to the extension
3. Modify URL - so frequently used path and query string can be utilized

4. Open a full-path URL in a new tab
5. Access opened tabs

Internet Explorer[6] is officially retired and no longer supported as of June 15, 2022, after more than 25 years of facilitating web usage and experience. Consequently, Internet Explorer was removed from the list, leaving Firefox and Chromium-based browsers such as Google Chrome, Microsoft Edge, Opera, and Vivaldi as the only viable options.

Both the Firefox and Chrome extension APIs provided comparable functionality. Firefox's extension technology is, to a large extent, compatible with the extension API utilized by Chromium-based browsers. Most extensions designed for Chromium-based browsers operate in Firefox with very minor modifications. One disadvantage was that Mozilla only support Manifest V2 and not V3. Since announcing Manifest V3 in 2018, Google has launched Manifest V3 in Chrome, started accepting Manifest V3 extensions in the Chrome Web Store, co-announced joining the W3C WebExtensions Community Group (formed in collaboration with Apple, Microsoft and Mozilla), and, most recently, laid out a timeline for Manifest V2 deprecation [16]. Beginning in January 2022, no new Manifest V2 extensions will be accepted, and Manifest V2 will cease to function in January 2023. Manifest V3 mandates a change to the ecosystem that restricts Manifest V2 extensions and will likely require Manifest V2-based extensions to conform to Manifest V3 in the near future.

[6]*Internet Explorer* is a World Wide Web browser included with Microsoft Windows. Windows 10 deprecated the browser in favor of Microsoft's Edge Browser.

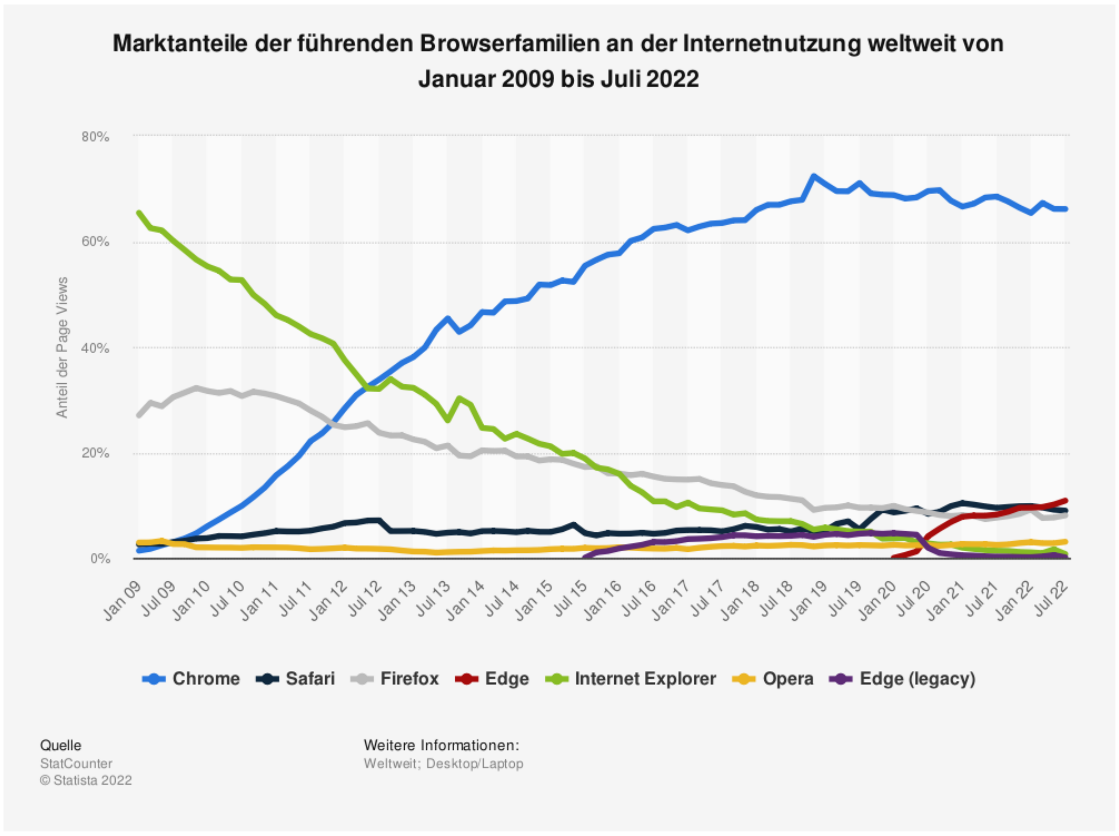


Figure 3.2: *Market shares of leading browsers worldwide by July 2022*

Additionally, it was discovered that the Chrome documentation was easy to grasp and was divided up into sections. Over two-thirds of all Internet users worldwide use Chrome as their primary browser (See Figure 3.2). As a result, it was decided to develop a Chrome-based browser extension.

3.5.2. Data storage

Data can be stored either on the web server or on the web client (the user's computer). Certain data types are better suited for one, while others perform more efficiently on the other. Sensitive and fragile data, for example, should always be stored on the server, whereas static data and user preferences can be stored on the client [14]. Currently, most organizations manage various data aspects through various record systems. They have difficulty locating appropriate data stores because they report on each data source separately, resulting in complex data analytics processing. For this project, the types of data that must be stored are specified in section 3.3. Client-side storage is used to store and manipulate data in the browser, as no servers are required for this project. The following are instances of data storage and manipulation in the browser:

- Maintaining the state of a client-side application, including the current screen, entered data, user preferences, etc.
- Utilities with strict privacy requirements that can access local data or files.

- Offline progressive web apps (PWAs)

Cookies

HTTP cookies, or Cookies were created to keep track of session data on the client side. According to the specification, a Set-Cookie HTTP header with session information was to be sent by the server as part of any response to an HTTP request. Here is an example of the headers in a server's response:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: cookie=any
Other-header: other-header-value
```

Listing 3.1: *Cookie server response's headers*

The cookie "cookie" with the value "any" is created by this HTTP response. The name and value are both URL-encoded before being sent. Session data is stored in the browser and retransmitted to the server with each request using the Cookie HTTP header, as shown in Listing 3.2.

```
GET /index.jsl HTTP/1.1
Cookie: cookie=any
Other-header: other-header-value
```

Listing 3.2: *Cookie HTTP header*

This additional data returned to the server can be used to determine the origin of the request. Although cookies provide a simple and well-supported storage mechanism, they have a number of drawbacks. To begin, each cookie is sent back and forth with each HTTP request (via HTTP headers), adding a significant amount of unnecessary overhead. Second, their storage capacity is far insufficient for modern web applications. Third, the same web application that uses cookies cannot run in multiple web browser tabs at the same time. Lastly, the API is somewhat cumbersome [12, 14].

Web Storage

Web storage was first addressed in the Web Applications 1.0 specification of the Web Hypertext Application Technical Working Group (WHAT-WG). Early development of this specification was incorporated into HTML5 before it was separated into its own specification. Its purpose is to circumvent some of the limitations imposed by cookies when data is required only on the client side, without the need to constantly send data back to the server. The most recent revision of the Web Storage specification is the second edition. The Web Storage specification's two key goals are to provide a system for storing large amounts of data that survives between sessions and to provide a way to store session data outside of cookies [4].

The second edition of the Web Storage specification defines two types of storage: `localStorage`, for persistent data, and `sessionStorage`, for temporary data that is only kept for the duration of a single user session. Until the `localStorage` object is deleted either by JavaScript or the user by clearing the browser's cache, it will remain accessible. All information stored in `localStorage` will survive browser restarts, page refreshes, and the closing of all open windows and tabs. However, the `sessionStorage` object only stores information temporarily, until the browser is closed. This works in a similar way to a session cookie, except that it disappears when the browser is closed. Information saved to `sessionStorage` is preserved even after the page is refreshed, and depending on the browser's provider, may be accessible even if the browser crashes and is restarted. There are two different ways to store data in the browser that will persist through a page refresh, and both of these APIs are available in modern browsers. LocalStorage and SessionStorage are two types of window properties that have been widely supported by all major browser vendors since 2009.

There are restrictions with Web Storage, just like there are with other client-side data storage options. There are limitations that can only be experienced in a certain browser. The client-side data size limit is typically determined per origin (protocol, domain, and port), giving each origin a predetermined amount of storage space. This prohibition is enforced by looking into where the storing page originated from. Aside from that, Web Storage uses synchronous operations, which will cause the main thread to become blocked. In addition, web workers and service workers are unable to access it, and the only type of data that it can store is strings.

IndexedDB

The Indexed Database API, abbreviated IndexedDB, is a structured data store in the browser. IndexedDB was created as a replacement for the now-deprecated Web SQL Database API. The goal of IndexedDB was to create an API that allowed for the simple storage and retrieval of JavaScript objects while also allowing for querying and searching. IndexedDB is intended to be almost entirely asynchronous. As a result, the majority of operations are performed as requests that will be executed later and produce either a successful or an error result. To determine the outcome of nearly every IndexedDB operation, you must attach `onerror` and `onsuccess` event handlers.

It is a key-value browser-based NoSQL (Not Only SQL) data store that operates asynchronously. The NoSQL method of database management is an alternative to the more traditional relational and object-oriented schemas. Instead, NoSQL uses a key/value pair to store information. More than that, the database can store a lot of information, and its API allows for instant access to a practically infinite amount of organized data. But security was not a priority when designing IndexedDB, so it could be seen as insecure.

IndexedDB is a low-level API that necessitates extensive configuration before use, which can be a burden even when storing relatively straightforward information. Instead of relying on promises like most modern APIs do, it uses events instead. Wrappers for the IndexedDB library that provide promises, such as *idb*[7], hide both the powerful features and the complex machinery (such as transactions and schema versioning) that come with using the library.

[7] *idb* is a small library that closely resembles the IndexedDB API, albeit with significant usability enhancements. GitHub repository: `https://github.com/jakearchibald/idb`

Chrome Storage API

Google Chrome provides an API called `chrome.storage`. This API has been optimized to meet the specific storage needs of extensions [6]. It allows you to directly synchronize data and persist data across tabs. Asynchronous storage ensures that scripts are not interrupted. As a result, asynchronous saving outperforms synchronous saving in terms of performance. The asynchronous behavior must be considered during implementation. Web Storage or `localStorage` has a maximum memory limit of 5 MB and does not support tab-spanning storage because the data is only available in the current context. The cross tab storage and readout is required for the extension's development. Storage-wise, it's very similar to the `localStorage` API, but there are a few key distinctions:

- User data can be synchronized in an automated fashion using `storage.sync`.
- Without requiring a background page, the content scripts of the extension can directly access user data.
- Even when using split incognito behavior, a user's extension settings can be saved.
- It is faster than the blocking and serial `localStorage` APIs because it is asynchronous with bulk read and write operations.
- Objects can be used to store user data (the `localStorage` API stores data in strings).
- It is possible to read the administrator's enterprise policies for the extension (using `storage.managed` with a schema).

Using `chrome.storage` can alleviate the aforementioned limitations with other client-side data storage solutions. Chrome Storage API is well-documented, simple, and takes very little time to set up. Data stored in `chrome.storage` can be accessed not only from tabs and windows of the same origin. As determined by the JSON stringification of each value and the length of each key, the maximum amount of data that can be stored in local storage is 5MB. However, this value can be ignored if `unlimitedStorage`[8] permission is used. Due to the limitations of HTML5 storage in the context of developing a Chrome Extension, the decision is made to use `chrome.storage`.

[8]This permission is only applicable to Web SQL Database and application cache (See issue `https://bugs.chromium.org/p/chromium/issues/detail?id=58985`). Additionally, wildcard subdomains such as `http://*.example.com` are not supported at this time.

4 Implementation

After the design has been specified, the extension's implementation may begin. This section describes further explanation from section 3.5 for the technology used. Because no backend implementation is required for the project, this section will only cover the front-end implementation.

4.1. User Interface

This section goes through some of the technical aspects of the user interface of the extension. The user interface was built with React. As described in section 2.5, React makes it easy to create interactive UIs. Using JSX makes it even simpler for web designers to change the browser's DOM using HTML. Furthermore, the user interface is developed in TypeScript rather than regular JavaScript to ensure type safety.

4.1.1. Build Tool

Nowadays, most front end projects use a build tool to assist in the development of web applications. The build tool for the user interface is *Webpack 5*[1].

4.1.2. Component Library

A component library is used in this project to speed up UI development. The rebass component library is used in this project. *Rebass*[2] was chosen because it is lightweight and an excellent choice for prototype and UI development without the need to invest time in establishing a custom design system from the start.

4.1.3. Code Style

In this project, a code formatter (Prettier) and linter (ESLint) are used to ensure a uniform code style and conform with TypeScript best practices.

4.1.4. Utility Libraries

The large JavaScript bundle is one of the most prevalent performance problems in front-end development. Watching the loading spinner and using complicated, slow applications are situations that should be avoided. The utility libraries, which are sets

[1] *Webpack* is a bundler for static modules that is used in modern applications written in JavaScript. Webpack version 5 includes faster builds with persistent caching, smaller bundler size, Module Federation, etc. More information on `https://webpack.js.org/blog/2020-10-10-webpack-5-release/`

[2] *Rebass* is a simple React UI component library that allows you to create primitive UI components using the Styled System library. More information on `https://rebassjs.org/`

of functions for typical tasks like formatting dates and looking for unique items in arrays, have a big impact on the bundle size. However, the extension's development can go much more quickly with the aid of utility libraries. The following utility libraries were used for the extension:

- Lodash[3]: A modern utility library that performs typical functional programming tasks.
- date-fns[4]: A modern date utility library for manipulating dates and time, which are typically difficult for developers to handle.
- React Feather[5]: A set of icons for React that are used in the scope of this project.

4.2. File structure

This section walks through the project's file structure, as shown in Figure 4.1 `.husky` is used to format code with prettier on git commit. Webpack generates the `dist` folder. This folder contains the JavaScript files generated by TypeScript (such as `background.js`, `contentScript.js`, `popup.html`, etc.). From the folder structure it is conceivable that `yarn` is the package manager and `jest` is the testing library. The extension's essential files are inside `src` and `static`. The `options` directory is in charge for the View of the options page, whereas `popup` is responsible for the extension's pop-up. All of the React components used in this project, like all other React projects, are saved in the `components` directory. All necessary components are imported from the `components` directory into the extension's pop-up and options page. Additionally, all of the React Hooks used in this project are kept in the `hooks` directory. The `hoc` folder, houses the files that make up the Higher-Order Components (HOC), which are an advanced component pattern in React. This pattern is employed for this project's component styling and conditional rendering.

[3]*Lodash*. GitHub repository `https://github.com/lodash/lodash`
[4]*date-fns*. GitHub repository `https://github.com/date-fns/date-fns`
[5]*React Feather*. GitHub repository `https://github.com/feathericons/react-feather`

```
/
├── .husky
├── dist
├── src
│   ├── background
│   ├── components
│   ├── constants
│   ├── content-script
│   ├── contexts
│   │   ├── parameter.tsx
│   │   ├── pathname.tsx
│   │   └── url.tsx
│   ├── hoc
│   ├── hooks
│   ├── options
│   │   ├── index.css
│   │   ├── index.tsx
│   │   └── options.tsx
│   ├── popup
│   │   ├── index.css
│   │   ├── index.tsx
│   │   ├── popup-with-router.tsx
│   │   └── popup.tsx
│   ├── spec
│   ├── types
│   └── utils
│       ├── storage.ts
│       ├── tabs.ts
│       └── url.ts
├── static
│   ├── icons
│   └── manifest.json
├── webpack
├── .gitignore
├── jest.config.js
├── package.json
├── README.md
├── tsconfig.json
└── yarn.lock
```

Figure 4.1: *Directory Tree*

4.3. Installation

When a Chrome extension is installed, a manifest file is read, which acts as a contract between the extension and the browser. In addition to trivial data such as the extension's name, description and version, the runtime permissions are defined (`permissions`).

Furthermore, the functions that are to be triggered after events occur are registered with the help of so-called service workers (`background`). Furthermore, an options page or configuration page (`options_page`) can be defined. The version of the manifest file is determined by `manifest_version`, which is currently the most recent and recommended version. The field `action` allows the user to customize the appearance and behaviour of the buttons that appear on the Chrome toolbar. In this case when the button with the icon `icons/icon-filtre-16.png` is clicked Chrome will load the `popup.html` file. The `content_scripts` field in the manifest is used to register scripts that are statically declared and contains the following variables:

- `matches`: Specifies which pages will receive the script injection. The special pattern that is used `<all_urls>` matches any URL that starts with a permitted scheme[6].
- `js`: An array of JavaScript files to be embedded in relevant web pages. The elements of this array are injected in the order in which they appear here.
- `run_at`: Controls at which point in time the JavaScript files are injected into the web page.

By setting the `run_at` field to be `document_idle`, the browser chooses a time to inject scripts between `document_end` and immediately after the *window.onload*[7] event fires. The exact moment of injection depends on how complex the document is and how long it is taking to load, and is optimized for page load speed [7].

```
{
  "name": "Filtre",
  "description": "A filtering assistant for Chrome",
  "version": "0.0.1",
  "manifest_version": 3,
  "icons": {
    "16": "icons/icon-filtre-16.png",
    "48": "icons/icon-filtre-48.png",
    "128": "icons/icon-filtre-128.png"
  },
  "action": {
    "default_popup": "popup.html",
    "default_title": "Filtre",
    "default_icon": "icons/icon-filtre-16.png"
  },
  "permissions": ["storage", "tabs", "unlimitedStorage"],
  "options_page": "options.html",
  "background": {
    "service_worker": "background.js"
  },
  "content_scripts": [
    {
```

[6] `http`, `https`, `urn`, `file`, or `ftp`, and that can contain "*" characters are all permitted schemes of a URL

[7] *window.onload* event is triggered when the entire page, including all dependent resources such as stylesheets and images, has loaded. More information on `https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event`

```
      "matches": ["<all_urls>"],
      "js": ["contentScript.js"],
      "run_at": "document_idle"
    }
  ]
}
```

Listing 4.1: *Manifest File (JSON)*

This function (Listing 4.2) is invoked upon extension installation, extension update, and Chrome update. An empty object is set as the initial filters and a default config object is set as the initial configuration.

```
import { setStoredConfig, setStoredFilters } from '../utils/storage'

chrome.runtime.onInstalled.addListener(() => {
  setStoredFilters({})
  setStoredConfig({
    excludedParameters: [],
    topFiltersCount: DEFAULT_TOP_FILTERS_COUNT // 3
  })
})
```

Listing 4.2: *On install functions (TypeScript)*

```
export const setStoredKey = (
  key: LocalStorageKeys,
  data: Record<string, any>
): Promise<void> => {
  const vals: LocalStorage = { [key]: data }
  return new Promise((resolve) => {
    chrome.storage.local.set(vals, resolve)
  })
}

export const setStoredFilters = (
  filters: Record<string, any>
): Promise<void> => {
  return setStoredKey('filters', filters)
}
```

Listing 4.3: *Helper functions for Chrome Storage API (TypeScript)*

```
export interface LocalStorage {
  filters?: Record<string, any>
  config?: Record<string, any>
}
```

```

export type LocalStorageKeys = keyof LocalStorage
```

Listing 4.4: *TypeScript interface of a LocalStorage object (TypeScript)*

4.4. Realization

While the implementation's primary focus was on storing records, a number of issues arose during the process of integrating the source code into a Google Chrome extension that required attention. When and how the URL should be stored were the main concerns. This section explores the solutions to these problems.

Storing Point in Time

The first question can be addressed by Listing 4.1, on the `content_scripts` field. Since the `run_at` field is set to `document_idle`, the URL is stored immediately after the whole page has loaded. That is, everytime a website is viewed or refreshed, the record is always saved, even if the user navigates away from the tab while the page loads.

This solution works, however, only if the user do a full page refresh. Client-side routing is used by the majority of front-end applications today. Due to the rise in popularity of single-page applications, a greater number of developers now consider client-side routing when building single-page applications (SPAs). When a user clicks on an internal link within the SPA, the URL bar should change to indicate that the page is being updated without requiring a complete page refresh.

Therefore, we moved the upserting filter function from the content scripts to the service workers. Whenever a tab is updated, the upsert filter function is invoked by the services workers.

TypeScript type of data

To answer the second question, the record that we want to store needs to be uniform. To provide a clear record structure, a TypeScript type is created.

```
export type ParameterType = {
  uuid: string
  createdAt: number
  version: string
  paramKey: string
  paramValue: string
  count: number
  lastUpdatedAt: number
}

export type Parameters = ParameterType[]

export type PathType = {
  name: string
  subpaths: PathType[]
  parameters: Parameters
}

export type Paths = PathType[]

export type Entry = {
  [host: string]: Paths
}

export type GeneralState = 'loading' | 'ready' | 'error'

export type ITopFilter = Pick<
  IParameter,
  'count' | 'uuid' | 'paramKey' | 'paramValue'
> & { path: string }

export type ITopFilters = ITopFilter[]
```

Listing 4.5: *TypeScript type of a record entry (TypeScript)*

Filters in URLs

As mentioned in subsection 2.1.1, a URL consists of various components, some of which are host name, path and query string. The topics, filters and facets, were also discussed in section 2.3. When adding persistent filters to a website, these definitions play a significant role.

Due to the hierarchical nature of a URL's path, traditional filters are typically positioned there. Facets typically utilize the query portion of a URL, as they are not hierarchical. Hierarchical classification refers to the process of classifying objects using a single hierarchical taxonomy. Faceted classification may employ hierarchy in one or more of its facets, but also permits the use of multiple taxonomies to classify objects. As seen in the preceding examples, the "multiple taxonomies" provided by facets are unsuitable for inclusion in the URL's path segment. Returning to the use of non-standard URL encoding, as stated in subsection 2.1.1 of the Path definition:

> The path component contains data, usually organized in hierarchical form, that, along with data in the non-hierarchical query component, serves to identify a resource ...

and subsection 2.1.1 under Query String continues:

> The query component contains non-hierarchical data

This indicates that facets should not appear in path segments, but rather as query parameters.

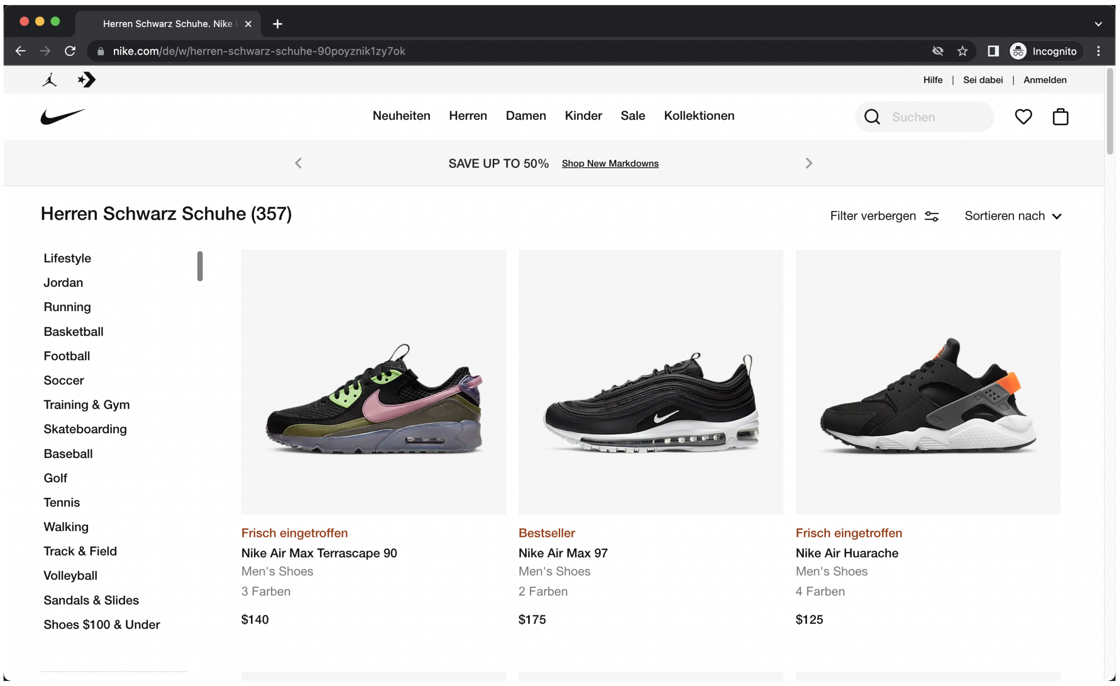
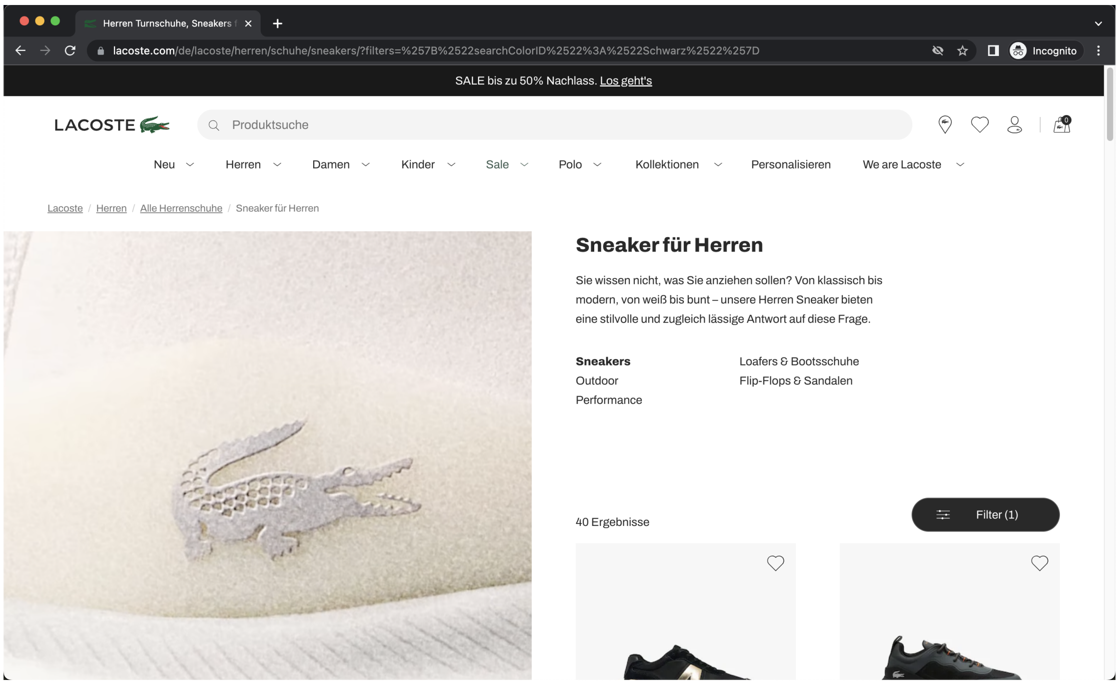

nike.com/de/w/herren-schwarz-schuhe-90poyznik1zy7ok

Figure 4.2: *Nike's Mens Black Sneakers URL full-path*

In practice, however, not all websites adhere to these guidelines. A good comparison would be how Nike and Lacoste filter their products on their respective websites (See Figure 4.2 and Figure 4.3). In both of these examples, the user is looking for black sneakers on their respective websites. The figures show that both have different ways of displaying the URL. Nike includes the filters and facets in the URL path, as well as a hash string at the end. Lacoste, on the other hand, stores the classic filter in the URL path and the facets as URL queries, which is the preferred method of utilizing the URL. Lacoste stores the query parameter's value in a JSON format. If the URL is decoded twice, the output would be: `?filters={"searchColorID":"Schwarz"}'`.


lacoste.com/de/lacoste/herren/schuhe/sneakers/?filters=%257B%2522searchColorID%2522%3A%2522Schwarz%2522%257D

SALE bis zu 50% Nachlass. Los geht's

Sneaker für Herren

Sie wissen nicht, was Sie anziehen sollen? Von klassisch bis modern, von weiß bis bunt – unsere Herren Sneaker bieten eine stilvolle und zugleich lässige Antwort auf diese Frage.

40 Ergebnisse

Figure 4.3: *Lacoste's Mens Black Sneakers URL full-path*

The solution to this issue would be to store both the query parameters and the path component of the URL. These properties ought to be accessible through the same host name. However, this solution raises an additional problem: how to transform the URL so that it can be stored as JSON.

Converting URL to JSON Object

The URL must be saved as a JSON object in order to count the number of times each query parameter is entered for each host. Here is an example of storing a simple URL with a single path and a query parameter:

`https://app.marta.de/caregivers?caregiver.germanVerbalProficiency=one,two`

```json
{
  "app.marta.de": [
    {
      "name": "/caregivers",
      "parameters": [
        {
          "count": 1,
          "createdAt": 1660224894790,
          "lastUpdatedAt": 1660224894790,
          "paramKey": "caregiver.germanVerbalProficiency",
          "paramValue": "one,two",
          "uuid": "8006a4a5-482e-4282-940f-569a54fd29ce",
          "version": "0.1.0"
        }
      ],
      "subpaths": []
    }
  ]
}
```

Listing 4.6: *A record example from app.marta.de (JSON)*

Converting such a simple URL to a JSON object is a straightforward process. A problem arises when converting more complex URLs on a faceted navigation system, as depicted in Figure 4.3 To store nested URL paths in our data structure, URLs are traversed in a manner similar to directories. In order to convert URL pathname to a JSON object, a recursive helper function has been implemented, as shown in Listing 4.9. Therefore, the previously mentioned URL can be saved as follows:

```
{
  "www.lacoste.com": [
    {
      "name": "/de",
      "parameters": [],
      "subpaths": [
        {
          "name": "/lacoste",
          "parameters": [],
          "subpaths": [
            {
              "name": "/herren",
              "parameters": [],
              "subpaths": [
                {
                  "name": "/schuhe",
                  "parameters": [],
                  "subpaths": [
                    {
                      "name": "/sneakers",
                      "parameters": [],
                      "subpaths": [
                        {
                          "name": "/",
                          "parameters": [
                            {
                              "count": 1,
                              "createdAt": 1660552042754,
                              "lastUpdatedAt": 1660552042754,
                              "paramKey": "filters",
                              "paramValue": "%7B%22searchColorID%22:%22
    Schwarz%22%7D",
                              "uuid": "8a737280-8feb-4968-a97f-222424
    eb6c95",
                              "version": "0.1.0"
                            }
                          ],
                          "subpaths": []
                        }
                      ]
                    }
                    ]
                  }
                ]
              }
            ]
          }
        ]
      }
    ],
}
```

Listing 4.7: *Record with long pathname (JSON)*

Due to the scalability of the data structure depicted in Listing 4.7, visiting another pathname from the same host is not a problem, as shown in Listing 4.8. This structure permits subpaths to contain additional subpaths and queries. Each subpath can contain additional subpaths and queries with this structure. If a user visits the URL `https://www.lacoste.com/de/lacoste/herren/schuhe?filters=...`, the parameter values are added to the array of parameters under the subpath `/schuhe`.

```
...
  {
    "name": "/schuhe",
    "parameters": [],
    "subpaths": [
      {
        "name": "/sneakers",
        "parameters": [],
        "subpaths": [
          {
            "name": "/",
            "parameters": [
              {
                "count": 2,
                "createdAt": 1660552137705,
                "lastUpdatedAt": 1660552137705,
                "paramKey": "filters",
                "paramValue": "%7B%22searchColorID%22:%22Schwarz%22%7D",
                "uuid": "5882c1bf-92e8-4e3f-9454-1b6f1acd6dc7",
                "version": "0.1.0"
              }
            ],
            "subpaths": []
          }
        ]
      },
      {
        "name": "/outdoor",
        "parameters": [],
        "subpaths": [
          {
            "name": "/",
            "parameters": [
              {
                "count": 1,
                "createdAt": 1660552464135,
                "lastUpdatedAt": 1660552464135,
                "paramKey": "filters",
                "paramValue": "%7B%22searchColorID%22:%22Schwarz%22%7D",
                "uuid": "31b25e02-9631-4bc1-acda-740c7fb58343",
                "version": "0.1.0"
              }
            ],
            "subpaths": []
          }
        ]
      }
    ]
  }
...
```

Listing 4.8: *Record with different pathnames (JSON)*

```
const recursiveUpsertFunc = (
  filters: PathType[],
  paths: string[],
  parameters: URLSearchParams
) => {
  const element = paths.shift()
  if (typeof element === 'undefined' || element === null) return
  const pathIndex = filters.findIndex((f) => f.name === element)

  if (pathIndex > -1) {
    if (paths.length > 0) {
      recursiveUpsertFunc(
        filters[pathIndex].subpaths,
        paths,
        parameters
      )
    } else {
      filters[pathIndex].parameters = upsertParams(
        filters[pathIndex].parameters,
        parameters
      )
    }
  } else {
    if (paths.length > 0) {
      const newLength = filters.push({
        name: element,
        parameters: [],
        subpaths: []
      })
      recursiveUpsertFunc(
        filters[newLength - 1].subpaths,
        paths,
        parameters
      )
    } else {
      const newParameters = upsertParams([], parameters)
      filters.push({
        name: element,
        parameters: newParameters,
        subpaths: []
      })
    }
  }
}
```

Listing 4.9: *Resursive pathname to JSON function (TypeScript)*

Retrieving the Top Filters

While converting the URL into a JSON object is a formidable task, obtaining the top most used filters is more challenging. It is necessary to use yet another recursive helper function because it is unclear how deeply the JSON object is nested. This task cannot be resolved by using a regular loop function. Before starting to write the function, as depicted in Listing 4.10, the procedures needed to solve the issue are defined.

An array must first be initialized, and it will contain the top most used parameters. The options page of the extension allows user to change the length of this array. The second step entails iterating through each of the root paths that are accessible. Third, the parameters array for each path must be looped through and then the number of times each parameter is entered needs to be compared against existing values inside the top parameters array. The process is then repeated until every subpaths and parameters array is iterated.

```
const recursiveTopFilterFunc = (
  topFilters: ITopFilters,
  paths: IPaths,
  pathname?: string
) => {
  for (const { name, parameters, subpaths } of paths) {
    let subpathname = pathname ? `${pathname}${name}` : name

    // Find out the maximum count of the parameters
    for (const param of parameters) {
      const { uuid, count, paramKey, paramValue } = param
      if (
        // Continue if current `count` is less than
        // the minimum value of the `topFilters` array
        (topFilters.length === topFiltersCount &&
          count < topFilters[topFiltersCount - 1].count) ||
        // Continue if user excludes current `paramKey`
        // in the options page
        excludedParameters.includes(paramKey)
      ) {
        continue
      }
      topFilters.push({
        path: subpathname,
        uuid,
        count,
        paramKey,
        paramValue
      })

      // Always sort the parameter count in a descending
      // order
      topFilters.sort((a, b) => b.count - a.count)

      // Remove excess `count` if `topFilters` length
      // exceeds the `topFiltersCount`
      if (topFilters.length > topFiltersCount) {
        topFilters.pop()
      }
    }
    recursiveTopFilterFunc(topFilters, subpaths, subpathname)
  }
}
```

Listing 4.10: *Recursive function to get the top filters (TypeScript)*

5 Test

Testing is essential to the software development process. The code that is written today should not be difficult to maintain later in the application's life cycle. To avoid creating these burdens, tests are written. The benefit of tests is that they enable developers to provide documentation for future contributors in addition to merely validating function behavior. There are numerous types of software testing techniques, including Unit tests, Integration tests, and Functional tests, among others. For the extension's development, unit tests are written to ensure that each system component performs flawlessly and fulfills its function in isolation.

5.1. Unit Test

To ensure that the `chrome.storage` is tested, the unit tests must run within the context of a Chrome Extension. Furthermore, this allows for more thorough testing of the extension's individual components. Due to the increasing complexity, the use of a mocking framework for data simulation was omitted. For the tests, a configuration object is used instead. Because it allows for modular development, relatively faster than other testing frameworks thanks to parallel testing and is widely used, *Jest*[1] will be used as the testing framework.

The unit tests have three main goals: to test the parse function from a path name to an array, to ensure that the parameter count is correctly updated and to test the transformation of the full-path URL to a JSON object. Table 5.1 shows, sorted by test identifier, how the path name parsing tests are defined:

Table 5.1: *Test definitions for parsing path name to an array of string*

ID	Req. Scenario	Test data	Expected result
T-01	Convert simple single path name	`/path`	`['/path']`
T-02	Convert simple path name with one subpath	`/path/a`	`['/path', '/a']`
T-03	Convert simple pathname with multiple subpath	`/path/a/b/c/d/e/f/g/h`	`['/path', '/a', '/b', '/c', '/d', '/e', '/f', '/g', '/h']`

[1]*Jest* is a JavaScript testing framework that ensures the integrity of any JavaScript codebase. More information on `https://jestjs.io/`

T-04	Convert path name with only "/" character	`/`	`['/']`
T-05	Convert path name with "/" character before parameters	`/path/`	`['/path', '/']`

After testing the URL path name parsing to an array of strings, multiple scenarios when updating and inserting parameters are tested as shown in Table 5.2.

Table 5.2: *Test definitions for upserting parameters*

ID	Req. Scenario	Test data	Expected result
T-06	Insert new parameter	`?a=b`	The parameter key `a` is saved along with its value `b` one time
T-07	Update parameter count	`?a=b`	The parameter key `a` is saved along with its value `b` mutliple times
T-08	Insert multiple parameters	`?a=b&c=d&e=f&g=h&i=j`	The parameter keys `a, c, e, g, i` are saved along with its values `b, d, f, h, j` one time
T-09	Update multiple parameter counts	`?a=b&c=d&e=f&g=h&i=j`	The parameter keys `a, c, e, g, i` are saved along with its values `b, d, f, h, j` multiple times
T-10	Insert parameters with same values but different keys	`?a=b&c=b`	The parameter keys `a, c` are saved separately even though they have the same value `b`
T-11	Insert parameters with same keys but different values	`?a=b&a=c`	The parameter values `b, c` are combined and saved together due to matching parameter key `a`
T-12	Insert parameter with only key, without value	`?a&c`	The parameter keys `a, c` are saved with an empty string as its parameter value

Finally, as shown in Table 5.3, the scenarios of converting a full-path URL to a JSON object are tested.

Table 5.3: *Test definitions for URL*

ID	Req. Scenario	Test data	Expected result
T-13	Store URL with only path name	`/path`	The subpath `/path` is saved without another subpaths or parameters
T-14	Store URL with path name and a parameter	`/path?p=o`	The subpath `/path` is saved with `p` as key and `o` as value
T-15	Store URL with a long path name and a parameter	`/path/a/b/c/d/e/f/g/h?p=o`	Nested subpaths are saved and the parameter `p=o` is appended on the last subpath `/h`
T-16	Store URL with a path name and multiple parameters	`/path?f=o&s=t`	The subpath `/path` is saved with two different key parameters and their respective values
T-17	Store URL with only parameter	`/?p=o`	The subpath `/` is saved with `o` as key and `o` as value
T-18	Store URLs with parameters on two path names	`/path?p=o` and `/path/a?p=o`	The subpath `/path` is saved with parameter `p=o` along with a subpath `/a` also with the parameter `p=o`

6 Demonstration and Evaluation

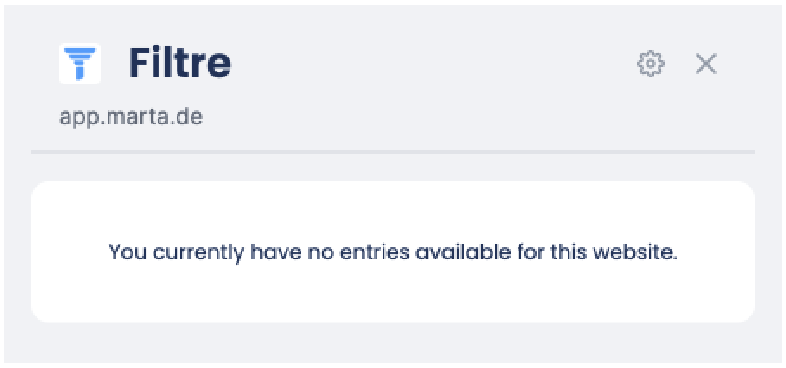
After the system's features have been implemented, the system can be demonstrated and evaluated. This chapter provides a demonstration of the use cases derived from the objective of the thesis, as well as an evaluation of the use cases based on the requirements analysis conducted in section 3.3.

6.1. Demonstration

Due to the fact that the extension was originally created for Marta's website, it will be used as an example in the demonstration. The following procedure is illustrated in this chapter as a result of this thesis:

1. With the extension installed, visit any webpage with URL path or query
2. Store the URL path and query in a JSON format
3. See the available top filters for the specified host name
4. View the number of times URL parameters have been entered by opening the extension's pop-up.

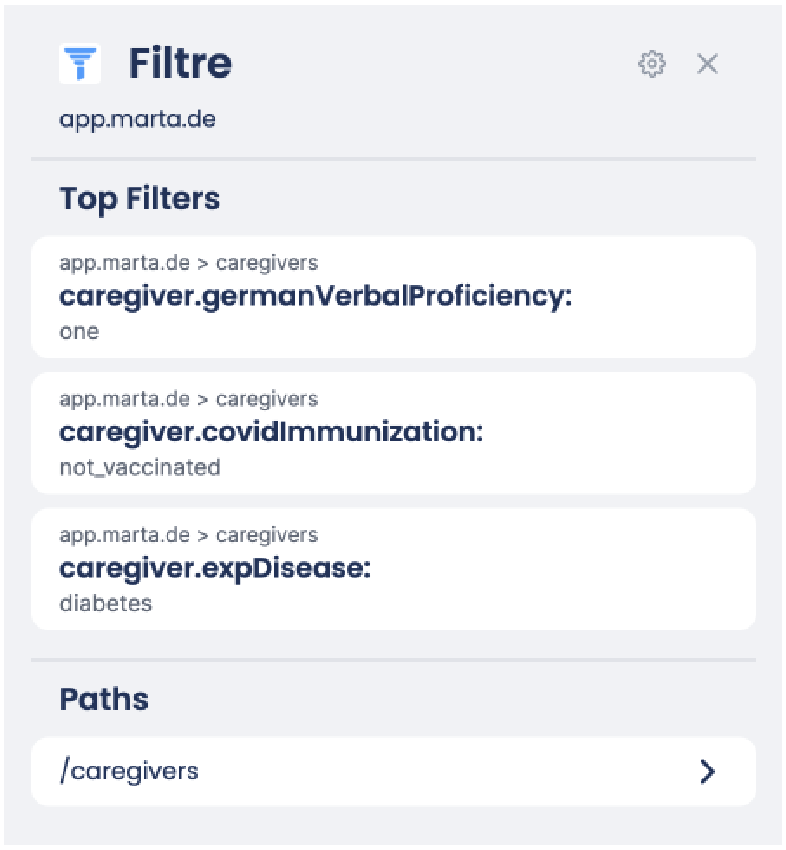
The `dist` directory is loaded manually to `chrome://extensions` because the extension has not yet been published in the Chrome Web Store. Once the extension has been loaded, it appears in the Chrome Extensions Toolbar. When the extension is opened for the first time, users will be notified that the entries for the specified host are empty (See Figure 6.1).



You currently have no entries available for this website.

Figure 6.1: *Screenshot of the no entries available page*

Families can find suitable caregivers through the Marta webapp at `app.marta.de/caregivers`. The page limits search results to 30 caregivers. In addition to the restriction, therefore, a filter based on URL query is implemented, allowing families to filter more caregivers. A modal dialog[1] is displayed for filter configuration. After modifying the filter settings, the URL becomes `app.marta.de/caregivers?caregiver.germanVerbalProficiency=one`. This URL is now stored in the extension's `chrome.storage`. Figure 6.2 is a listing of the most frequently entered query parameters after configuring the filter settings multiple times, as well as the top-level paths stored from the website's hostname at the bottom of the page. Since the user has only visited the `/caregivers` page, only one item is displayed under the Paths section.


Filtre
app.marta.de

Top Filters

app.marta.de > caregivers
caregiver.germanVerbalProficiency:
one

app.marta.de > caregivers
caregiver.covidImmunization:
not_vaccinated

app.marta.de > caregivers
caregiver.expDisease:
diabetes

Paths

/caregivers

Figure 6.2: *Screenshot of the top filters page*

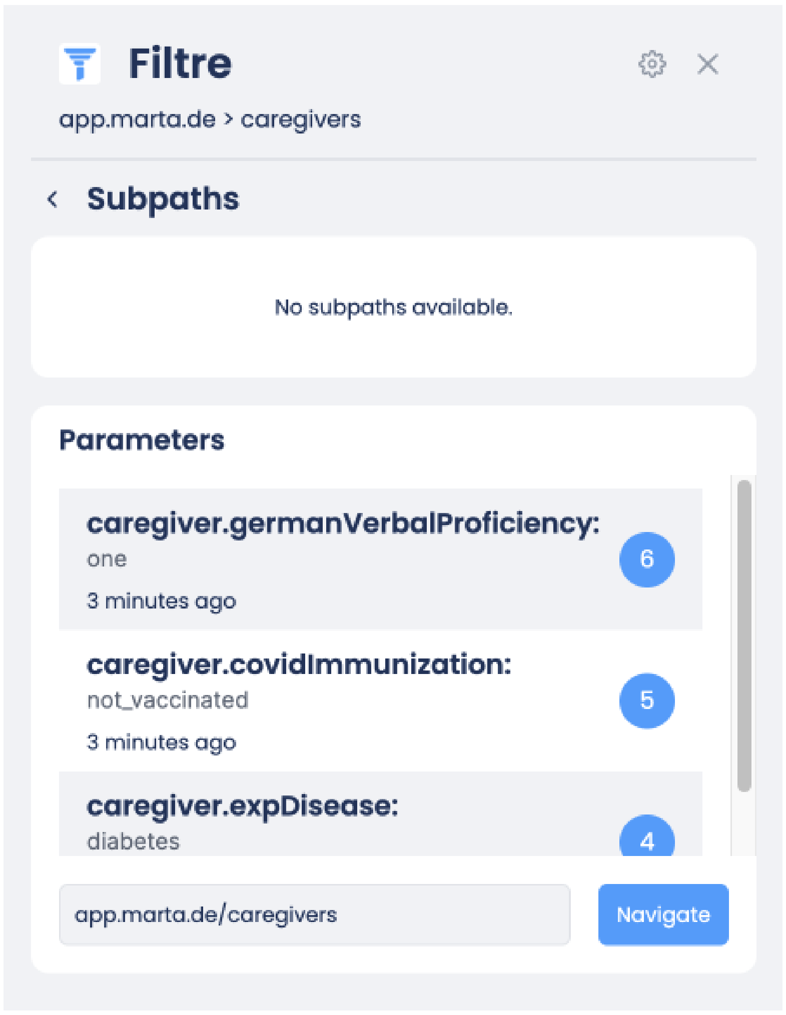
When a user clicks on one of the top filters within the list, the URL of the active tab is modified to the corresponding host name, path and query parameter of the chosen top filter. As long as the top filters share the same path name, they can be combined. If not, the user is redirected to another path name of the top filter selected. The number of top filters displayed on the list is configurable from the extension's options page (See Figure 6.4). The user also has the possibility to go through their URL history.

Clicking on one of the top-level paths below the top filters section navigates user to a more refined page inside the extension. Figure 6.3 displays a list of saved URL subpaths and parameters. The user can see the number of times each parameter is

[1]*Modal dialog* is a dialog that appears on top of the primary content and initiates a special mode that requires user interaction. More information on `https://www.nngroup.com/articles/modal-nonmodal-dialog/`

called on the right side of the box, as well as when the parameter was last called below the parameter value, on this page. Reloading the page updates the `lastUpdatedAt` key and the parameter count

Another thing to note is the input field and the "Navigate" button at the bottom of the page. The value in the input field is updated whenever the user navigates through the extension's "directory path" and selects the parameters. By clicking the "Navigate" button, the user can then navigate to the updated URL in a new tab.


Filtre

app.marta.de > caregivers

Subpaths

No subpaths available.

Parameters

caregiver.germanVerbalProficiency:
one
3 minutes ago
6

caregiver.covidImmunization:
not_vaccinated
3 minutes ago
5

caregiver.expDisease:
diabetes
4

app.marta.de/caregivers

Figure 6.3: *Screenshot of the known subpaths and parameters page*

The options page of the extension can be accessed by the user through the settings icon on the upper right corner. Excluded parameters can be defined on the options page, according to the functional requirements of the extension mentioned in subsection 3.3.1. Every string in the "Exclude Parameters" field is not displayed anywhere in the extension, including the top filters and the query parameters section. Furthermore, the user can specify how many top filters are displayed in the extension pop-up. The value is set to 3 by default.

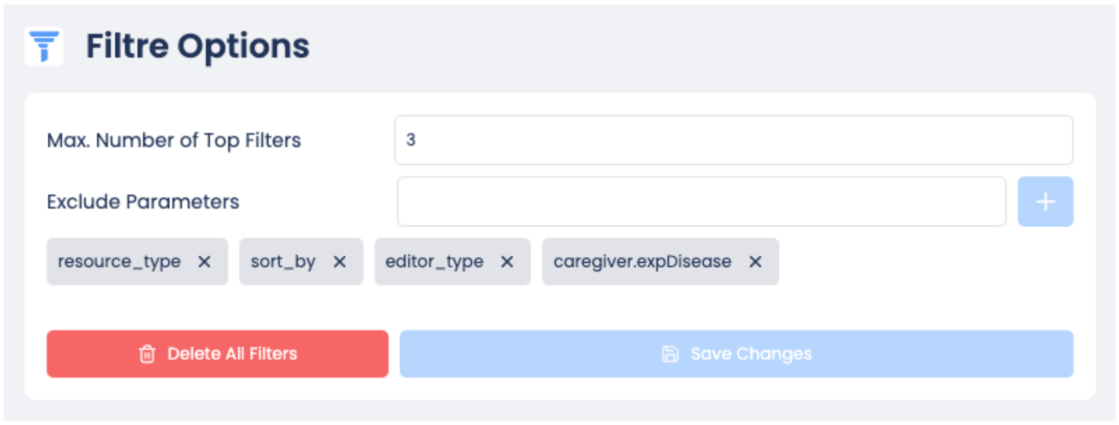
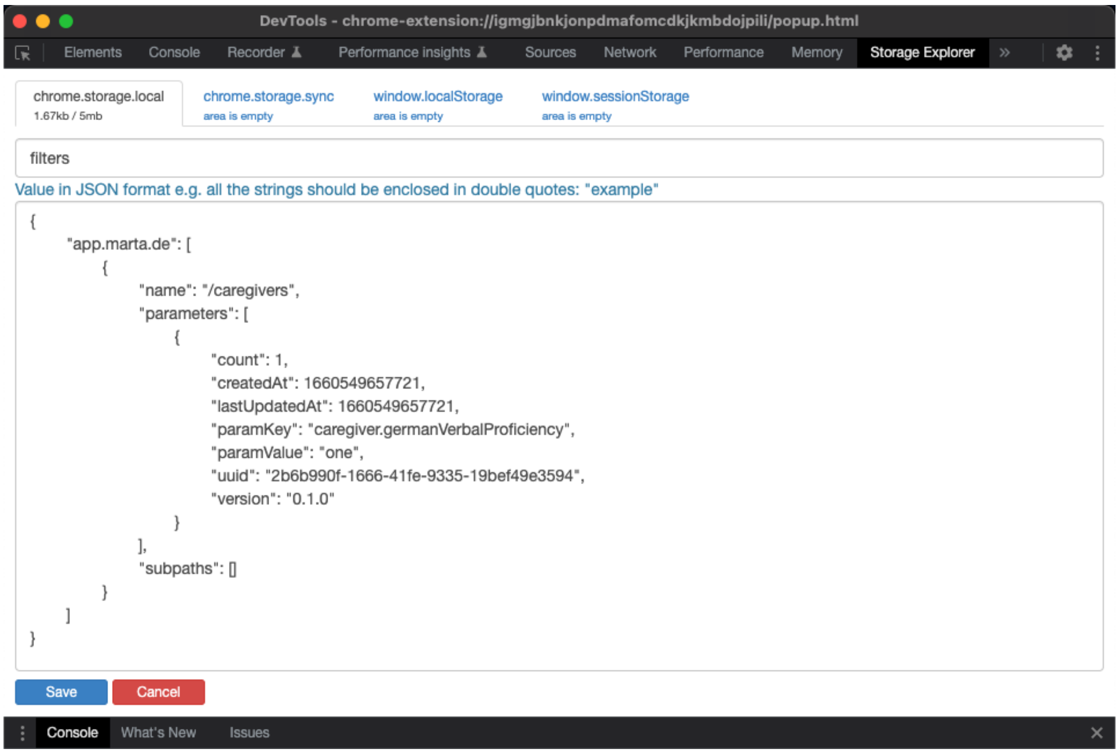


Figure 6.4: *Screenshot of the extension's options page*

With the help of *Storage Area Explorer*[2], it is possible edit or view the `chrome.storage` using a user interface without the need for console logging the `chrome.storage` object in the code (See Figure 6.5). The current size and size limit are also displayed within the Storage Explorer.



```
{
    "app.marta.de": [
        {
            "name": "/caregivers",
            "parameters": [
                {
                    "count": 1,
                    "createdAt": 1660549657721,
                    "lastUpdatedAt": 1660549657721,
                    "paramKey": "caregiver.germanVerbalProficiency",
                    "paramValue": "one",
                    "uuid": "2b6b990f-1666-41fe-9335-19bef49e3594",
                    "version": "0.1.0"
                }
            ],
            "subpaths": []
        }
    ]
}
```

Figure 6.5: *Screenshot of the Storage Area Explorer*

[2]*Storage Area Explorer* is a an easy-to-use editor for Chrome Packaged Apps & Extensions' Storage Area. GitHub repository: `https://github.com/jusio/storage-area-explorer`

6.2. Evaluation

The achievement of the goals and criteria outlined in the requirement analysis should serve as a metric for determining whether the system and its functionalities have achieved their primary objective in accordance with the thesis objective. Table 6.1 displays, by requirement identifier, the degree to which the defined requirements have been met:

Table 6.1: *Evaluation table of requirement analysis*

ID	Implemented	Remark
FR-01	☑	
FR-02	☑	Based on the user's history after installing the extension.
FR-03	☑	It is possible to select multiple parameters. When two identical keys with different values are selected, the second value overwrites the first and no duplicates are produced.
FR-04	☑	
FR-05	☑	
FR-06	(☑)	The URL is saved but not on each page load because most front-end frameworks nowadays avoid reloading the entire page when navigating through the website. As a result, the URL is now saved whenever the URL value is changed.
FR-07	☑	
FR-08	☐	Removing a query parameter count is unnecessary because it contradicts the extension's purpose, which is to suggest the most frequently used filters. The reason this requirement was written in the first place was to allow for customization for a better user experience. Data storage optimization is an option for improvement as data grows and storage capacity is limited.
FR-09	☑	The "Delete All Filter" button will prompt the user to type "yes" to ensure that the button is not accidentally clicked. To avoid confusion, the button should be differentiated more.

FR-10	☐	Reset configuration button is for now not necessary because currently there are only two available configuration options, which are "Max. Number of Top Filters" and "Excluded Parameters".
NFR-01	☑	
NFR-02	☑	
NFR-03	☑	
NFR-04	☑	
NFR-05	☑	
NFR-06	☑	
NFR-07	☑	

Minor exceptions notwithstanding, the extension was therefore fully implemented. No usability tests were conducted.

6.3. Publish in the Chrome Web Store

The release of the extension in the Chrome Web Store would allow Google to at least examine the implementation and identify potential vulnerabilities. Nonetheless, the extension must be internationalized beforehand.

7 Conclusion

This chapter provides a concise summary of the results and discoveries made in the preceding chapters, as well as an outlook on potential improvements to the system in the near future.

7.1. Summary

A large website with filter options or even a faceted navigation system can be difficult to navigate, especially if the search must be repeated. Toggling on a filter is a simple task, but performing it multiple times can be tedious, especially if the filter is identical. The thesis's primary objective is to enhance the user experience in this particular scenario.

Amazon and Netflix, among other leaders in e-commerce and media, recognize the importance of a good search and navigation experience on their websites. They have invested heavily in making their vast catalogs easily navigable, thereby exposing users to new, valuable products and content. However, there is one aspect on which they place little emphasis: the frequency of visits to an item or product. Amazon and Netflix recommend the most popular global searches to users. These recommendations are not customized to the user and are based on data collected from users around the world. They provide related products, but not the most frequently visited item.

The objective of this thesis is to address the aforementioned problem for websites that support search and filtering. With the extension, users can return to a previously visited page in a matter of clicks. The extension is exclusive to Chrome users and stores its data in the user's `chrome.storage` directory. Each time a user modifies the URL of a browser tab, the extension saves the new information. More frequently a user visits a page or uses a filter on the page, the higher the URL suggestion will appear in the extension.

7.2. Outlook

The extension proposed in this thesis is far from perfect. Extensions are products that can and should be improved over time. One significant improvement of the extension is the internationalization of the application, which allows users to select their preferred language when using the extension. Due to the fact that the extension's initial focus was on Marta's customer-facing platform, some obstacles were overlooked.

Some websites use hash values in URL path names, which reduces the readability of the extension. Even though users can exclude URL parameters from the options page, there are still plenty of marketing or UTM parameters that vary by website (UTM parameters do not vary). Another useful improvement would be to suggest users directly the top most used filters in the first view, instead of guiding them to traverse through the "URL directories". Furthermore, the extension currently works for every website. Another good improvement would be to disable the extension if the website does not allow filtering.

Last but not least, after adding the internationalization support, the extension could be published to the Chrome Web Store.

References

[1] Tim Berners-Lee, Roy Fielding, and Larry Masinter. *Uniform resource identifier (URI): Generic syntax*. Tech. rep. 2005.

[2] Tim Berners-Lee, Larry Masinter, and Mark McCahill. *Uniform resource locators (URL)*. Tech. rep. 1994.

[3] Johansson David. *Building maintainable web applications using React: An evaluation of architectural patterns conducted on Canvas LMS*. 2020.

[4] Matt Frisbie. *Professional JavaScript for Web Developers*. John Wiley & Sons, 2019.

[5] Cory Gackenheimer. "Introducing flux: An application architecture for react". In: *Introduction to React*. Springer, 2015, pp. 87–106.

[6] Google. *chrome.storage*. Online. 2022. URL: `https://developer.chrome.com/docs/extensions/reference/storage/`.

[7] Google. *Content Scripts Run time*. Online. 2021. URL: `https://developer.chrome.com/docs/extensions/mv3/content_scripts/#run_time`.

[8] Google. *Service worker overview*. Online. 2022. URL: `https://developer.chrome.com/docs/workbox/service-worker-overview/`.

[9] Google. *What are extensions?* URL: `https://developer.chrome.com/docs/extensions/mv3/overview/`.

[10] Naimul Islam Naim. "ReactJS: An Open Source JavaScript Library for Front-end Development". In: (2017).

[11] Sheena S Iyengar and Mark R Lepper. "When choice is demotivating: Can one desire too much of a good thing?" In: *Journal of personality and social psychology* 79.6 (2000), p. 995.

[12] Zachary Kessin. *Programming HTML5 applications: building powerful cross-platform environments in JavaScript*. " O'Reilly Media, Inc.", 2011.

[13] Lei Liu et al. "Chrome Extensions: Threat Analysis and Countermeasures." In: *NDSS*. 2012.

[14] Matthew MacDonald. *HTML5: The Missing Manual*. 1st ed. " O'Reilly Media, Inc.", 2013.

[15] Pratik Sharad Maratkar and Pratibha Adkar. "React JS - An Emerging Frontend JavaScript Library". In: *Iconic Research And Engineering Journal s* 4.12 (2021), pp. 99–102.

[16] Alexei Miagkov and Bennett Cyphers. *Google's Manifest V3 Still Hurts Privacy, Security, and Innovation*. Online. 2021. URL: `https://www.eff.org/deeplinks/2021/12/googles-manifest-v3-still-hurts-privacy-security-innovation`.

[17] Mozilla. *What is a URL?* Online. 2022. URL: `https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL`.

[18] Jakob Nielsen. *10 Usability Heuristics for User Interface Design*. Online. 1994. URL: `https://www.nngroup.com/articles/ten-usability-heuristics/`.

[19] Hong Duc Phan. "React framework: concept and implementation". In: (2020).

[20] Jiaming Qu, Jaime Arguello, and Yue Wang. "A Study of Explainability Features to Scrutinize Faceted Filtering Results". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 1498–1507.

[21] React. *Context*. Online. 2021. URL: `https://reactjs.org/docs/context.html`.

[22] React. *Introducing JSX*. Online. 2020. URL: `https://reactjs.org/docs/introducing-jsx.html`.

[23] Dolière Francis Somé. "Empoweb: empowering web applications with browser extensions". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 227–245.

[24] Ian Sommerville. "Software engineering 9th Edition". In: *ISBN-10* 137035152 (2011), p. 18.

[25] W3C. *G161: Providing a search function to help users find content*. Online. 2016. URL: `https://www.w3.org/TR/WCAG20-TECHS/G161.html`.

[26] Kathryn Whitenton. *Filters vs. Facets: Definitions*. Online. 2014. URL: `https://www.nngroup.com/articles/filters-vs-facets/`.

[27] Miron Zuckerman et al. "On the importance of self-determination for intrinsically-motivated behavior". In: *Personality and social psychology bulletin* 4.3 (1978), pp. 443–446.

8 List of Abbreviations

API	Application Program Interface
COM	Component Object Model
CSS	Cascading Style Sheets
DOM	Document Object Model
HOC	Higher-Order Components
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IPC	Inter-process Communication
JS	JavaScript
JSON	JavaScript Object Notation
JSX	JavaScript XML
MB	Megabyte
MVC	Model-View-Controller
Props	Properties
PWA	Progressive Web Apps
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WHAT-WG	Web Hypertext Application Technical Working Group
XML	Extensible Markup Language

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

21.08.2022, Berlin