Development of a Chrome Extension for Personalized Filter Suggestions

Abschlussarbeit

zur Erlangung des akademischen Grades:

Bachelor of Science (B.Sc.)

an der

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

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

Eingereicht von Sunan Regi Maunakea 566144

22. August 2022

Danksagung

[Text der Danksagung]

Abstract

[Summary of the thesis]

Keywords

Google Chrome Extension, Search and Filter URL Parameters

Contents

List of Figures

Listings

1 Introduction

Direct access to digital information has completely changed how a single piece of information is retrieved, enabling us to be "better than reality" with search boxes, allowing users to jump directly to what they are interested in without using any complex systems. This instant gratification far outperforms previous methods of flipping through physical pages. However, the spread of digital information access has been accompanied by an explosion in the volume of information available about any given topic, to the point where even instant access via search does not necessarily make finding our needle in a haystack any easier.

It is a common belief in modern society that the more choices, the better–that the human ability to manage, and the human desire for, choice is unlimited. One study showed that the existence of choice increases motivation and enhances performance on doing tasks [23]. However, another study has shown that although having more choices might appear desirable, it may sometimes have negative effects on human motivation [9]. 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 [22]. 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 concept of URL and how resources are searched and filtered using URL parameters. The second section describes the concept of a browser extension. Furthermore, the choice of implementation browser is defined with which it is possible to develop an extension. The third chapter covers the architecture of a Chrome Extension. Finally, the framework for the extension's UI implementation is elucidated.

2.1. Uniform Resource Locator

Uniform Resource Locator, or URL, is a compact string representation for a resource available via the Internet [1]. URLs are used to "locate" resources, by providing an abstract identification of the resource location. 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. Here is an example of an HTTP URL: `http://www.example.com/software/index.html`

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 [15]. The most important parts are provided in the following sections:

Scheme

The scheme, which indicates the protocol that the browser must use to request the resource, is the first part of the URL. A protocol is a set method for exchanging or transferring data around a computer network. 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

The authority is then separated from the scheme by the character pattern `://`. If the authority is present, it includes both the host (e.g., `www.example.com`) and the port (80), separated by a colon:

- The host name identifies the host that holds the resource. A server provides services in the name of the host, but hosts and servers do not have a one-to-one mapping.
- The port number denotes the technical "gateway" used to access the web server's resources. It is typically omitted if the web server grants access to its resources via the HTTP protocol's standard ports. Otherwise, it is required.

Path

The path identifies the specific resource in the host that the web client wants to access. For example, `/software/htp/cics/index.html`.

Query String

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 doing a search 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
```

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 example, in an HTML document, the browser will scroll to the point where the anchor is defined; in a video or audio document, the browser will attempt to navigate to the time the anchor represents. It is important to note that the part following the `#`, also known as the fragment identifier, is never sent to the server with the request.

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 [21]. Users can find content by searching for specific words or phrases without having to understand or navigate the site's structure. This can be a faster or easier way to find content on large sites. A great site search function is specific to the website and not only constantly indexes the site to ensure the most recent content is easily accessible, but it also guides users as they explore a website's content, assisting them in discovering content they may not have known they were interested in. The best site search products delight users by allowing them to quickly connect with the content they require while also collecting valuable data about the content and products that visitors are most interested in.

2.3. Filters

The process of narrowing down a search based on predefined categories is known as filtering. These categories are frequently broad and based on a single dimension of the product. This allows user to quickly narrow down a large number of products to a more manageable set for further investigation.

Filters are broad categories defined by the business that do not change between searches, and they are frequently used behind the scenes. For example, an online clothing store might use "clothing type" as a filter, with four possible categories: shirts, pants, shoes, and accessories. When a website visitor clicks on "shirts" in the top navigation, the clothing type filter is applied, and the visitor sees only shirts on the results page.

2.4. Facets and faceted search

Facets, also known as facet filters, enable users to filter results by selecting values along different dimensions or facets [17]. It is widely used in e-commerce search engines and digital libraries where documents have rich metadata. Faceted search is a more granular method of finding products and results in a specific, targeted way that broad, one-size-fits-all filters do not allow.

Facets and faceted search are features of a well-designed user interface. Contextual facets that change depending on the item or category drive a user-friendly experience by guiding the user down the quickest path to the best result.

2.5. Browser Extension

Browser extensions or addons are third party programs, that can extend the functionality of browsers and improve users' browsing experience [19]. 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.

2.5.1. 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 [7]. 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.

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.

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 [11].

Service Worker

The Chrome extension platform switches from background pages to service workers in Manifest V3. 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 [6]. This technology allows native-like experiences over the open web, including 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 [6]. Websites are gradually improved by service workers through a lifetime akin to that of platform-specific programs.

2.6. React

React is a product of Facebook's engineering team, which is a JavaScript framework for creating user interfaces [4]. 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 [8]. The core objective of React is to provide the best possible rendering performance. Its strength comes from the focus on individual components. Using reusable components, it is found to be easy development for developers to design rich UI's. React incorporates with View part from MVC model [13]. React implements One Way dataflow so that it gets easier than traditional data binding. React uses virtual DOM it offers not so complex programming with faster execution. It makes use of composition to create intricate user interfaces out of simple building blocks known as Components [2].

Component

Each component has a render method, which can either return HTML or another React component, and returns a description of what to render. A combination of HTML and Javascript known as JSX is used to describe what should be rendered. A more detailed explanation of JSX is included in the JSX section. It is possible to specify a component as a class or a function. Props and state are crucial components when creating React applications.

States and props

The state of a component allows it to "remember" things, and it can change in response to user interaction or other application-wide actions. The State is optional; components without a state are referred to as presentational components. Components with a state are referred to as stateful components.

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. Props-drilling 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.

Class-based components

A class-based Component is created by extending the React.Component class. The state of a class-based component is updated using the method setState() and read by using this.state within the class. Using the setState() method makes the component re-render which is not the case when mutating the state directly. Class-based components include life-cycle methods that can be used to create more complex behavior. It also requires a render method to return JSX elements.

Life-cycle methods

Life-cycle methods are built-in functions that are called whenever a state or prop updates, a component renders, is destroyed, or both.

Function components

A pure function that takes props as input and outputs a JSX element is referred to as a function component. React Hooks are used to give the function component the same access to state and life-cycle methods as class-based components. In compared 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 [16]:

- Faster development, easier to read and test, debug and reusable; because function components do not have state and life-cycle-hook. Function component is a straightforward JavaScript function.
- Performance will be improved since function components are smaller and compiles more quickly than class components.
- There is no need to consider how to divide the component into a container and a standard UI component when utilizing a function component.

React Hooks

For more complex function components, React Hooks are utilized; they "hook onto" React capabilities. React hook names begin with the word "use," per tradition. By default, state is absent from function component; however, the useState React hook can be used to keep state for the duration of the component. All hooks, including the useState one, are repeatable inside a single component.

Function components do not come with built-in life-cycle methods, but they can be added using the useEffect hook. By default, the useEffect hook will run a function for each time the component is re-rendered, but it may be modified to just run for specific modifications.

The useContext hook, which will be described in the Context API section, is another React hook. In addition to built-in hooks, custom hooks can be made, allowing functionality to be reused throughout components.

Virtual DOM

For all components in the application, React generates a new View based on immutable states and props; a change to either a state or a prop causes the View to be re-rendered. The Views are now predictable and testable as a result. The user experience is negatively impacted by the time-consuming process of re-rendering views by swapping out the DOM for a new version, which also causes scrolling position and input information to be lost. Using a Virtual DOM, React provides a solution for this [2].

By establishing a new Virtual DOM subtree when data in the application changes and comparing it to the previous one, the Virtual DOM re-renders Views that need to update in an affordable manner without disrupting other DOM nodes. React determines the minimal number of DOM alterations required to match the virtual DOM with the real DOM. Javascript is used to do DOM manipulations, which are then queued up and executed in batches to save time. React takes care of manipulating the DOM to get to the various states, while developers utilize JSX to define how components should render in various states.

Context API

The Context API is a built-in tool that allows data to be shared between React components without the need for props-drilling. The createContext method in the React library is used to initialize a Context, which can be used by several components in the component tree. A Provider present on the Context instance is used to add data to the Context and make it accessible to children components. The data for the Context is specified in the Provider, a component that has a single property named value and accepts variables, functions, or objects as values. The Context data can be retrieved from any child component using a Consumer that is also available on the Context instance by enclosing children components inside the Provider component. The application uses a variety of contexts for various purposes rather than being restricted to a single instance.

Context API's callback function inheritance allows a child component to change the state of a parent component. Context is used, according to the official documentation, to exchange information that is regarded as global for the tree of React components, such as whether a user is logged in, the application's theme, or the language preferences. Because they depend on information provided by context from another component, using context may reduce the reusability of components. UseContext functions as the Consumer and grants access to the data for a functional component encased in a Context Provider.

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 [18]. The advantages of JSX include the fact that it makes writing templates for users who are familiar with HTML simpler and faster. Performance improves when code is compiled into JavaScript [16].

3 Methodology

In the following chapter, the component analysis is defined first. This results in the requirements that must be considered in the design when implementing the extension, which is then discussed in the third section. In the final section, the tests, to ensure that the extension matches the expected requirements and it is defect free, are described.

3.1. Design Concept

The original concept of this thesis is to create an additional feature for marta's customer-facing platforms. Marta as a business is currently best described as a marketplace between caregivers and families requiring 24-hour care. 24-hour care can be defined as living in a household with the person in need of care for a certain period of time. This means that caregivers are primarily responsible for basic care and household chores. In addition, they support the person in care's relatives in need of assistance in carrying out the activities they wish to do. Marta as a marketplace connecting families with caregivers is competing against more traditional agencies, where it can take several days or even weeks to find a family for a newly signed up caregiver or the other way around.

As a growing start-up that gained a lot of users in the past few months, marta would need to enhance their product, to compete in this expeditiously developing business. One way marta can provide a superior experience for both caregivers and families is to speed up the matching process between both parties. The caregiver and family inquiry forms are designed to record as much information as possible which can be used during the matching phase. The matches are then created by the teams in Berlin and Romania. To make the job more seamless, the technical team in marta introduced a "matching score" by which possible matches are sorted. As a result, a number of caregiver profiles with high matching scores are shown to the family. Filter functionality is provided to speed up the search process. This includes caregiver's earliest starting date, German skills, experience with diseases, etc.

Marta needs to introduce a continually improved, user-friendly filtering functionality to adjust to the needs of their users. An example of a user-friendly filter would be to provide quick filter suggestions which the user can click once and the desired results will be shown, instead of letting users select the same filter manually over and over again. In order to determine which filter suggestions are the most beneficial, frequently used filters need to be identified.

A question arose within this world of thought about how to improve the existing filtering functionality for a better user experience. An early concept was to create quick filters within the application that suggested to users which filters they frequently used. For example, if a family is looking for a caregiver with German speaking capability level 3, the German filter would be activated. Each filter usage will be counted and

saved to a relational database. The three most frequently used filters will be displayed as quick filters for all users, and the user will only need to click on those quick filters to activate them. Not only that, the user can also combine those quick filters to narrow the search even further.

The first problem with this idea was that it collected filter usage data from every user and stored it without providing any detailed information about who used the filter and when it was used. Each family's search criteria are unique. Hence the first idea would not be as beneficial to the users. We must also keep in mind that this approach will only benefit those who actively use the platform. The second concern was that if the filter recommendations were tailored to each individual user, the amount of data maintained would rapidly grow to be rather large. It would be a waste of storage space and would not be sustainable, especially if the number of filters and users grows. We would develop quick filters not only on one page, but on several. And each would require the same amount of storage space.

The concept was then expanded upon and a decision was reached. Instead of adding those quick filters directly into the platform, an extension that allows for even more personalized filter suggestions should be developed. The aforementioned issues can be addressed by developing an extension. The information that will be saved varies for each user and is saved in the end user's browser, which means that other users' activity will not impact the filter suggestions. Furthermore, search and filter is a popular website design method. It would benefit not just Marta's platform, but also other e-commerce websites like Lacoste, Nike, Adidas, and others. Moving it to an extension would allow people to choose whether or not to install it. And, because this is a separate capability with a different software architecture than Marta's primary platform, a new repository or codebase is developed to make it easier to distinguish and manage.

3.2. Requirements Elicitation and Analysis

The extension is used to improve the user's experience when navigating through an e-commerce website to search for a wanted product or service. The idea is to integrate the end user's browser with an extension, which will record the host name or domain of the visited website's URL along with the respective parameters. When enough records have been collected, the extension will display a list of parameters for the visited domain. Path parameters and query parameters will be separated from these parameters. The user can then navigate through the path parameters in the same way that they would navigate through a file system. When the last path parameter is reached, the query parameters and the number of times these parameters are called in the URL are displayed. Below the list is an input field and a button; these input fields will be filled based on and while navigating through the selected parameters. When the "Navigate" button is clicked, a new tab will open with the full-path built URL.

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.

3.2.1. 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 [20]. The following functional requirements are defined for the extension:

- User can see how many times each query parameters for each hostname are used
- User can build a URL query string based on their history
- User can navigate to the URL with the assembled query string
- The extension saves the hostname, pathname and query parameters of the URL
- The extension stores the URL attributes each time a page is loaded

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

3.2.2. 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 [20]. It often apply to the system as a whole rather than individual features or services. The following non-functional requirements arise for the extension to be developed:

- UI components must be implemented using React.
- TypeScript must be used to implement dynamic functions.
- Data must be saved on the end-user's Chrome storage.
- Both the external components used and the developed component itself must be well documented.
- The individual external libraries must be easy to update.
- 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.

3.3. Technologies

3.3.1. 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

After 25+ years of helping people use and experience the web, Internet Explorer (IE) is officially retired and out of support as of June 15, 2022. Thus, IE was eliminated from the list, leaving Firefox and Chromium-based browsers (such as Google Chrome, Microsoft Edge, Opera, Vivaldi) as the options.

Both the Firefox and Chrome extension APIs provided comparable functionality. To a considerable degree, Firefox's extension technology is compatible with the extension API used 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. New Manifest V2 extensions will no longer be accepted as of January 2022, and Manifest V2 will no longer function as of January 2023 [14]. Manifest V3 dictates an ecosystem change that limits Manifest V2 extensions and would likely force Manifest V2 based extensions to conform to Manifest V3 in the near future.

Additionally, it was discovered that the Chrome documentation was easy to grasp and was divided up into sections. Chrome also has an overwhelming market share: over two-thirds of all users globally use Chrome as their browser (See Figure 3.1). As a result, it was decided to develop a Chrome-based browser extension.

3.3.2. Data storage

Extensions can store data in two places: (1) on the web server or (2) on the web client (the end-user's-computer). Certain data types belong on one, while others perform

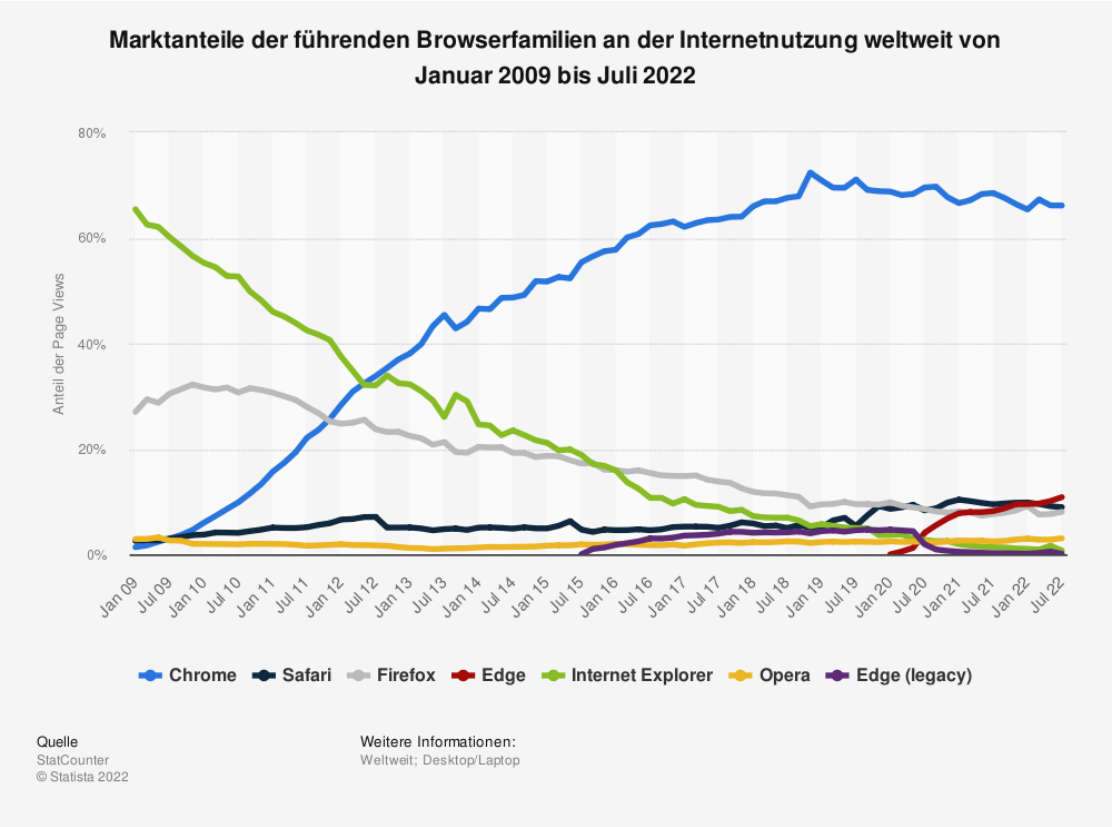

Figure 3.1: *Market shares of leading browsers worldwide by July 2022*

better 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 [12]. 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 3.2. Since no servers are needed for this project, data is stored and manipulated in the browser — also known as client-side-storage. Examples of data storage and manipulation scenarios in the browser include:

- Preserving the state of a client-side application, such as the current screen, entered data, user preferences, and so on.
- Utilities that access local data or files and have strict privacy requirements.
- Offline progressive web apps (PWAs)

Cookies

HTTP cookies, or simply cookies, were designed to store session information on the client. As part of any response to an HTTP request, the server was required to send a Set-Cookie HTTP header containing session information, according to the specification. A server response's headers, for example, might look like this:

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

Listing 3.1: *Cookie server response's headers*

This HTTP response creates a cookie called "name" with the value "value." When sent, both the name and the value are URL-encoded. Browsers save such session information and send it back to the server via the Cookie HTTP header for each subsequent request, as shown below:

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

Listing 3.2: *Cookie HTTP header*

This additional data returned to the server can be used to uniquely identify the client from which the request was sent. 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. Finally, the API is somewhat clumsy [10, 12].

Web Storage

The Web Applications 1.0 specification of the Web Hypertext Application Technical Working Group (WHAT-WG) was the first to address web storage. This specification's early development became part of HTML5 before being broken off into its own specification. Its goal is to circumvent some of the constraints imposed by cookies when data is required solely on the client side, without the need to send data back to the server on a constant basis. The Web Storage specification's most recent revision 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 [3].

The Web Storage specification's second edition comprises definitions for two objects: `localStorage`, the permanent storage mechanism, and `sessionStorage`, the session-scoped storage mechanism. The `localStorage` object is kept until it is explicitly erased via JavaScript or until the user clears the browser's cache. Data from `localStorage` will be retained even after page reloads, closing windows and tabs, and restarting the browser. On the other hand, the `sessionStorage` object only keeps data for a single session, which means the data is kept until the browser is closed. This is similar to a session cookie, which is deleted when the browser is closed. Data stored on `sessionStorage` endures through page refreshes and, depending on the browser vendor, may also be available if the browser crashes and is restarted. Both of these browser storage APIs provide two distinct methods for saving data in the browser that

can withstand a page reload. Both `localStorage` and `sessionStorage` are available as window properties in all major vendor browser releases since 2009.

Web Storage, like other client-side data storage solutions, has limitations. These are browser-specific restrictions. In general, the client-side data size limit is set per origin (protocol, domain, and port), so each origin has a fixed amount of space to store its data. This restriction is enforced by analyzing the origin of the page that is storing the data. Aside from that, Web Storage operates synchronously and will block the main thread. Furthermore, it is inaccessible to web workers or service workers and can only contain 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 an asynchronous, key-value browser-based NoSQL (Not Only SQL) data store. NoSQL is a database approach that is not relational or object oriented. Rather, NoSQL stores data in key/value format. Furthermore, the database can handle a large amount of data and has an API that provides quick access to an unlimited amount of structured data. However, IndexedDB may be considered insecure because security was not a consideration in its design.

IndexedDB has been eliminated from the list for the same reason that Web Storage has been removed. First, IndexedDB databases are tied to the page's origin (protocol, domain, and port), so information cannot be shared across domains. This means that `www.wrox.com` and `p2p.wrox.com` have their own data stores. Second, IndexedDB is extremely slow, even slower than a database on a low-cost server. Inserting hundreds of documents can take several seconds. Time is important for a quick page load. Even sending data to the backend over the internet can be faster than storing it in an IndexedDB database. Third, a workaround is required for IndexedDB to function alongside the Chrome extension. IndexedDB is a low level API that requires significant setup before use, which can be particularly painful for storing simple data. Unlike most modern promise-based APIs, it is event based. Promise wrappers like *idb*[1] for IndexedDB hide some of the powerful features but more importantly, hide the complex machinery (e.g. transactions, schema versioning) that comes with the IndexedDB library.

[1] *idb* is a tiny library that mostly mirrors the IndexedDB API, but with small improvements that make a big difference to usability. 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 [5]. 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. It has more or less the same storage capabilities as the `localStorage` API, with the following important differences:

- Using `storage.sync` user data can be automatically synced.
- Without the need for a background page, the extension's content scripts 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).
- The administrator's enterprise policies for the extension can be read (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. The maximum amount of data that can be stored in local storage is 5MB, as measured by the JSON stringification of each value plus the length of each key. However, this value can be ignored if `unlimitedStorage`[2] 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`.

[2]This permission applies only to Web SQL Database and application cache (see issue `https://bugs.chromium.org/p/chromium/issues/detail?id=58985`). Also, it doesn't currently work with wildcard subdomains such as `http://*.example.com`.

4 Implementation

After the design has been specified, the extension's implementation may begin. This section describes further explanation from section 3.3 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 2.6, 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.

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 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 an uniform code style and conform with TypeScript best practices.

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.). It is clear from the folder structure that `yarn` is the package manager and `jest` is the testing library. The extension's essential files are inside `src` and `static`.

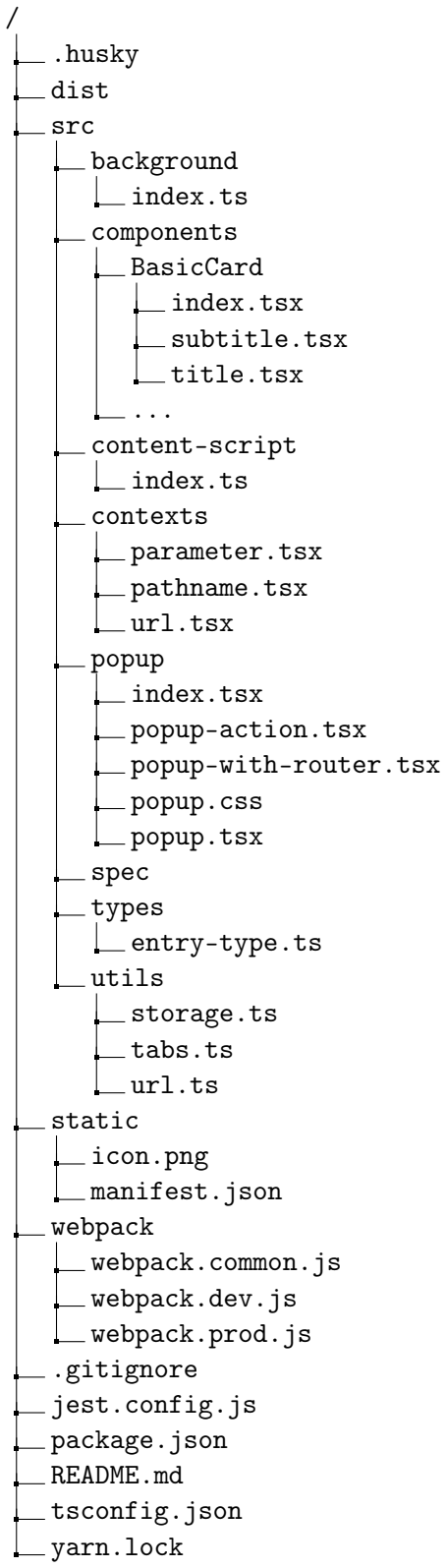

Figure 4.1: *Directory Tree*

4.3. Data storage

This section goes over the data storage used for the project.

- Implementation decisions, why use chrome.storage, why use react
- Introduce the simple software architecture
- Introduce the frontend react architecture

4.4. 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 and is widely used, Jest will be used as the test framework.

- What kind of test? Unit testing, integration testing or functional testing?
- Short explanation on types of test
- Implementation of Test
- Which test libraries are being used?
- What are the assertions?

4.5. 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, 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.

```
{
  "name": "Filtre Extension",
  "description": "Filtre Chrome Extension",
  "version": "1.0.0",
  "manifest_version": 3,
  "icons": {
    "16": "icon.png",
    "48": "icon.png",
    "128": "icon.png"
  },
  "action": {
    "default_popup": "popup.html",
    "default_title": "Filtre Extension",
    "default_icon": "icon.png"
```

```
  },
  "permissions": ["storage", "tabs", "unlimitedStorage"],
  "options_page": "options.html",
  "background": {
    "service_worker": "background.js"
  },
  "content_scripts": [
    {
      "matches": ["<all_urls>"],
      "js": ["contentScript.js"],
      "run_at": "document_idle"
    }
  ]
}
```

Listing 4.1: *manifest.json*

4.6. Application

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

chrome.runtime.onInstalled.addListener(() => {
  setStoredFilters({})
})
```

Listing 4.2: *background/index.ts*

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

export type LocalStorageKeys = keyof LocalStorage
```

Listing 4.3: *TypeScript interface of a LocalStorage object in utils/storage.ts*

```
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> => {
```

```
14    return setStoredKey('filters', filters)
15  }
```

Listing 4.4: *utils/storage.ts*

5 Results and Discussion

6 Conclusion

References

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

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

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

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

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

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

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

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

[9] 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.

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

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

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

[13] 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.

[14] 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`.

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

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

[17] 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.

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

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

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

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

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

[23] 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.

7 List of Abbreviations

API Application Program Interface

COM Component Object Model

CSS Cascading Style Sheets

DOM Document Object Model

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IPC Inter-process Communication

JS JavaScript

JSX JavaScript XML

MVC Model-View-Controller

Props Properties

UI User Interface

URL Uniform Resource Locator

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.

_____

Datum, Ort, Unterschrift