UNIVERSITÄT
DUISBURG
ESSEN

Offen im Denken

EI-Logger Plug-In

Explicit and implicit Logger for elicitation of SE user's data

Hossam Al Mustafa

Matriculation number: 3053109

Bachelor Thesis with the Department of Computer Science and Applied Cognitive Science

Applied Computer Science (B.Sc.)

supervisor:
Prof. Dr. Dirk Lewandowski
second supervisor:
Prof. Dr. German Neubaum

July 10, 2023

Contents

1 Introduction

We are living through the information revolution, where tremendous amounts of useful data are stored in servers all around the world. To utilize the stored data, we need to know how to extract relevant information from it. To do that we consider two paradigms. The first is a simple question-answer paradigm, where the searcher is looking for a well-defined information need and has a well-defined question, which is called information retrieving (Marchionini 2006). The other paradigm starts with a general or poorly defined information need that is oftentimes accompanied by an intent of exploration. This is called information seeking (R. White and Roth 2009).

To improve the extracting procedure, it is necessary to have an understanding of people's search behaviour and expectations (Singer et al. 2011; Marchionini 2006; R. W. White, Muresan, and Marchionini 2006). To gather this kind of data, we propose an easy-to-use tool that can log users' behaviour while engaging in a search procedure. reference

Although such tools exist, they are either discontinued, old, or do not meet the researcher's needs. There for, in this thesis, I will discuss evaluation tools of information retrieval and seeking systems and introduce a new tool for evaluating information-seeking systems, that addresses some of the unmet needs of researchers, like the ability to provide a precompiled set of search tasks and collect user feedback on search processes.

In Chapter 2 *(Theoretical Basis)* we will first discuss the information retrieval system model, motivate the needs for other models that led to the information-seeking systems models, and discuss various aspects of those models. Then continue in section 2.4 *(Evaluation)* to explain the evaluation of both models and their differences.

In Chapter 3 *(Existing Solutions)* we will proceed to talk about existing tools for evaluating information-seeking systems and list their shortcomings, which will pave the way for introducing the EI-Logger that addresses those shortcomings and offers flexibility that enables it to be deployed in and out of the context of information-seeking systems.

After that in the Chapter 4 *(Explicit Implicit Logger)*, the new tool will be introduced, where system architecture, data structure, and explanation diagrams will be discussed. In addition to that, a list of clear requirements will be provided and design decisions will be discussed.

For evaluating the new tool an evaluation method will be provided in Chapter 5

(Evaluation).

And at last in Chapter 6 *(Future Work)* I will list suggestions for improving the EI-Logger. Some of those suggestions were planned to be implemented in this thesis but were omitted due to prioritizing other features due to time constraints and some go beyond the scope of the EI-Logger as a browser extension and address the environment it works within, which will benefit the functionality and user experience provided by the EI-Logger.

1.1 Acronyms used:

- ESS: exploratory search system
- IR: information retrieval
- SERP: search engine result page
- GUI: graphical user interface
- EI-Logger: explicit implicit logger
- API: Application Programming Interface
- MDN: Mozilla Developer Network web documentation
- IndexedDB: NoSQL database API provided by web browsers to manage internal storage
- UUID: Universally Unique ID
- HTML: Hyper Text Markup Language
- VCS: Version Control System

2 Theoretical Basis

In this chapter, I will deal with some of the existing methods of gaining information. Having a large amount of stored data poses a challenge to categorising and extracting useful information. Ideally, we would have a tool that understands our question, intent and context and then delivers an answer that meets our information needs. Ideally, it would look through all relevant documents and create a correct, personalized, and optimized answer. It would provide us with additional relevant

information to improve our understanding or give more context to our search to expand our horizons. As difficult as this concept might sound, there have been some admirable advancements lately using artificial intelligence large language models like ChatGPT developed by OpenAi (*OpenAI* n.d.) and Bard developed by Google (*Bard* n.d.). The form of information gain offered by large language models like ChatGPT and Bard is still not widely used as traditional search engines and will be outside this thesis's scope.

The current dominant used tools on the internet are search engines, the most commonly known being Google, Bing and Yahoo (R. White and Roth 2009). From our different environments at work, university, school or family we can notice the wide use and importance of search engines, as they are the first contact point for finding information about a question in mind, which implies its usefulness in finding satisfying answers to those questions.

2.1 Introduction to information retrieval

Traditionally those search engines have been used as information retrieval tools that work with the information look-up paradigm (R. White and Roth 2009). This paradigm is useful to retrieve information in response to "when", "where" and "who" queries and here we can notice some similarities between search engines and other information retrieval systems like database management systems that support fast and accurate responses to look-up tasks. Searchers generally think about look-up search, according to Marchionini 2006, as "fact-finding" or "question-answering". Those systems that work with the look-up paradigms can be described according toR. White and Roth 2009 as a model comprising 4 components that are visualised in Figure 1.

1. the information-need or the intent of the searcher,
2. the expression of this need as a question or query,
3. the representation of the document that stores collections of data,
4. the collection that's being searched.

Interaction with search engines that use the look-up paradigm is familiar to our everyday life and can be described as the following according to R. White and Roth 2009. First, a searcher formulates the information need as text and enters it in the search field on the search engine's website. The search engine will give the searcher a list that's ranked based on various metrics including relevance for the query, with

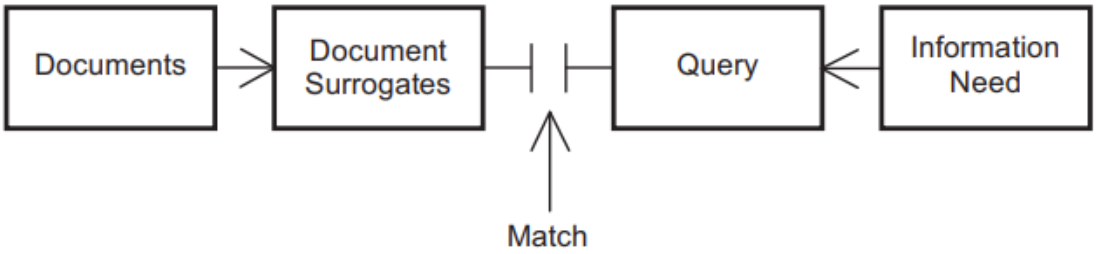

Figure 1: Lookup-based IR model (based on Bates, 1989)

each list item containing a title, snippets from the item source and a URL in addition to some metadata like page size and last modification date.

2.2 The need beyond information retrieval

Although the look-up search model has been one of the most successful applications of computers and it is still a topic of research and development (Marchionini 2006), if we start analyzing the information we got so far, we discover some shortcomings in the represented model.

Taking a closer look at the four components of this model, we find the first two steps depend solely on the searchers themselves. The first component is "The information need or intent of the searcher", so if the searcher does not have a clear well-defined need, they won't be able to give the search engine a good quality input. And if we assume that the searcher's information need is clear, then according to Blair 1992 the description of intellectual content can fail in multiple ways. For example, a searcher could describe their information need imprecisely, loosely, or even incorrectly rendering the textual formulation useless (Blair 1992), which means that the result of the second step is as good as the searcher is at textually formulating their information need, which in some cases will also lead to low-quality input for the search engine. This will limit the search engine's ability to satisfy the searcher's need by delivering a result that's less likely relevant to the searcher's intent, since according to Marchionini 2006 the look-up model functions best in analytical search strategies that begin with a well-defined query and result in a precise answer.

Furthermore, when comparing the suggested model to the real-world application of search engines, we find that in a search session, there are multiple iterations of the query until the best result is met and then the searcher might do some post-query browsing and examination of the search result (R. White and Roth 2009), therefore

this model has been criticized for its inability to capture the reality of human-search-engine interaction and its ignorance of vital factors like task context and information use (R. White and Roth 2009).

After we discussed the shortcomings of the model's architecture let us now discuss the model's usability shortcomings. Having previously mentioned that we can notice in our daily lives the wide usage of search engines, it's apparent that search engines and the internet, in general, have become the first destination for people when having a question and that's what Singer et al. 2011 and Marchionini 2006 confirm. This made people expect the internet and especially search engines to be the provider of other kinds of information needs, and therefore the search engines must strive to meet this expectation (Marchionini 2006). People now want to learn, discover and intellectually develop while engaging with search engines (Marchionini 2006). The information retrieval model is, as previously mentioned, answering the searcher's questions that begin with "when", "where", and "who", but satisfying other sorts of information needs would require a broader spectrum of questions to be answered like questions that start with "what", "how" and "why" (Marchionini 2006).

Developing intellectually and discovering new information transcends fetching specific pieces of information and goes into the realm of delving into new topics without having preexisting knowledge, which is motivated by curiosity (R. White and Roth 2009). This presents the need to distinguish between information retrieval and information seeking. As defined by R. White and Roth 2009, we find that with information seeking, there is ambiguity about the existence of the sought information as the keyword here is seeking and not retrieving. When retrieving information we are sure of its existence and the searcher's task is to formulate the query to find it. This behavioural change from information retrieving to information seeking, combined with all the shortcomings in the architecture and usability, triggered the emergence of the exploratory search model that I'm going to discuss in the next segment.

2.3 Exploratory search

All kinds of searches are in some way exploratory, therefore it is somewhat difficult to enclose the concept of exploratory search into a definition (R. White and Roth 2009). However, when using the term exploratory search we can think about two aspects that represent it, the information-seeking problem and the information-seeking process. As we will see, those two aspects are closely connected since the information-seeking problem motivates the searcher to start the information-seeking process.

2.3.1 Information-seeking problem

The information-seeking problem identifies a problematic situation, where the state of the searcher's knowledge is gaped and incomplete and this gap between what is known and what is wished to be known can be closed, for reasons like curiosity and aspiration to personal and intellectual development

The knowledge gap or wish for intellectual development is not limited to a certain domain, it could manifest as problems of scientists, designers or politicians, and it could last for minutes or years to come. The context of the information-seeking problem can be characterised by ambiguity, as people engaged in exploratory search generally have no background in the search domain and are unsure about their goals and how to achieve them (R. White and Roth 2009). Additionally, the context could be so complex, that it can be conceptually split into multiple abstraction levels, which deconstructs the search tasks into multiple smaller tasks that incrementally drive the learning and development process. However, if in this case, the context was ambiguous, it will be harder to split the task into smaller ones. Exploratory search systems should be able to solve those problems. In the case of ambiguity, people usually start with a provisional query to gain primary insight, further their understanding and refine their search methodology, which can be improved with exploratory search systems by providing users with annotations and external resources to facilitate understanding and decision-making, so they can continue their search purposefully (R. White and Roth 2009).

2.3.2 Information-seeking process

Information-seeking processes or learning searches are the actions taken in pursuit of solving information-seeking problems. In those searches, a searcher spends their time reiterating queries and investigating and comparing returned results. Another field of applying learning searches is social searching, where the aim is building friendships and finding people of interest (R. White and Roth 2009).

As previously stated, analytical search strategies begin with a well-defined query and yield precise answers and according to Marchionini 2006 learning search tasks are functional at best with a combination of those analytical strategies and browsing strategies, that depend on trial-and-error, selection, and investigation tactics. R. White and Roth 2009 explains more by saying that lookup searches are a part of learning and learning is a crucial component of the investigation. In Figure 2 we see an illustration of the relationship between lookup and exploratory searches, where the lookup processes interconnect with the exploratory search processes, acting as an integral part of learning and investigating.

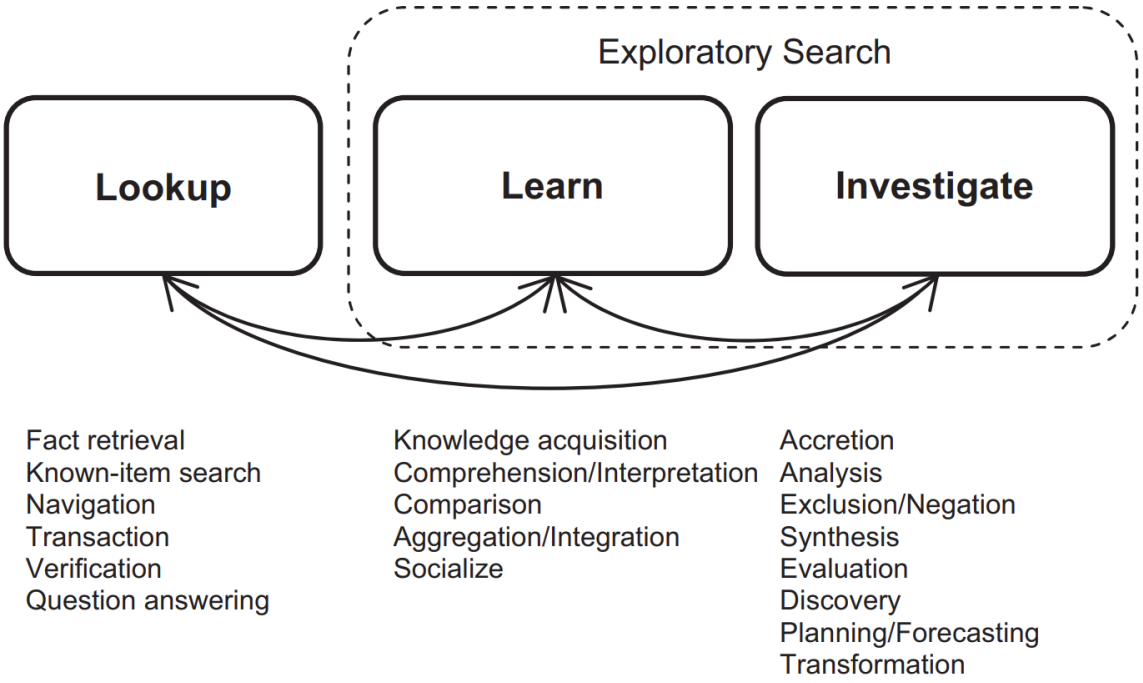

Figure 2: Exploratory search activities taken from R. White and Roth 2009. Arrows represent the interaction between activities

One important feature that distinguishes exploratory search processes from look-up searches, is that a look-up search ends when the sought fact is found, whereas an exploratory search might span minutes, days or months since, the information-seeking problem is open-ended, multifaceted a might change and develop during the search process (R. White and Roth 2009).

Since the lookup search ends by finding the sought-after fact, the search engines that are optimized for lookup searches are tuned to have high precision, which means that it minimizes the number of possibly irrelevant objects to precisely find the answer. This approach is counterproductive to supporting the searcher's exploration (Manning, Raghavan, and Schütze 2008). Ideally, we would want to maximize the number of possibly relevant objects that are retrieved, which is called recall. Moreover, recall and precision trade-off against one another (Manning, Raghavan, and Schütze 2008) as the next example should elaborate. According to Manning, Raghavan, and Schütze 2008 precision is defined as

$$Precision = \frac{\#(relevantitemsretrieved)}{\#(retrieveditems)}$$

and recall is defined as

$$Recall = \frac{\#(relevantitemsretrieved)}{\#(relevantitems)}$$

According to the previous definitions, we can always get a recall of value one if we retrieve all documents for every query, but that will cause very poor precision.

2.4 Evaluation

With all of the differences discussed so far between the information retrieval and exploratory search models, one would also expect there to be differences between their evaluation methods. In this section I will shed some light on the information retrieval model's evaluation and its shortcoming in evaluating exploratory search models and what are the needed features in the evaluating framework of an exploratory search model.

2.4.1 Information retrieval metrics

Since the ultimate goal is satisfying the searcher's information needs, a happy searcher implies a successful search. Of course, there are multiple relevant and irrelevant factors for searchers' happiness. An example of relevant factors is the speed of response and information relevance. A result should be fast but relevant to the searcher's query, as a lightning-fast non-relevant answer won't make a user happy. An example of non-relevant factors in searchers' happiness is the design and layout of the graphical user interface, which are independent of the quality of the result.

Since relevance is a key factor in result quality, researchers leaned toward objective testing for evaluating information retrieval systems. Such tests would consist of three components:

1. A document collection.
2. A collection of information needs, expressable as queries.
3. Binary relevant judgments, (relevant, non-relevant), which is called the golden standard.

Relevance judgment is done based on the information need and not on the query, which means that an answer is relevant when it addresses the information need and not only contains the words comprising the query. For example, by giving a search engine the word Java, the user could hereby mean the programming language Java

or the Indonesian island of Java. For sure it is very hard to infer the searcher's intent from one word without a context, but the user certainly has one. Therefore the queries used for evaluation must clearly express the information-need to be able to evaluate the returned documents by relevance. Tests of the discussed structure are objective, standardized and can be easily done by researchers in their research facilities. Although they are a simplification of reality since they do not include the searcher as a factor, they have proven to be good enough for Evaluating Information Retrieval Systems. Information in this section is all taken from (Manning, Raghavan, and Schütze 2008)

2.4.2 Exploratory search evaluation metrics

As mentioned IR system evaluation can be done objectively but since the user experience plays a big part in the exploratory search paradigm a more user-centric approach is needed (Singer et al. 2011; Marchionini 2006; R. W. White, Muresan, and Marchionini 2006; R. White and Roth 2009). A complete model would even integrate the searcher, search task, information base, search system, and effects and it would look like Figure 3. The information seeker has an information need (Task) that encourages him/her to interact with a search system. This interaction is heavily affected by the information base of the searcher hence the multiple arrows between the information base and the search system. As the search session proceeds, the task, the information seeker, and thus the information-seeking process will get influenced by the newly gained information, which represents the multiple iterations in the process. The double-sided arrows suggest two-way interactions as the process goes on (R. W. White, Muresan, and Marchionini 2006). This model captures all relevant elements of an exploratory search process, that the evaluation framework needs to address.

From Figure 3 we conclude that the golden standard used in IR evaluation that is based on precision-recall metrics is ineffective. Still, metrics are needed nonetheless, even though it is hard to assess the success of an exploratory search session. In the workshop of (R. W. White, Muresan, and Marchionini 2006) the following metrics were suggested:

1. engagement and enjoyment: The level of searcher engagement and their positive emotional experience can serve as a reliable indicator of system performance. Considering the interactive nature of exploratory search we can take the searcher's focus on the task as a determining factor of how the system effectively supports the search activity. Tracking the number of actions like

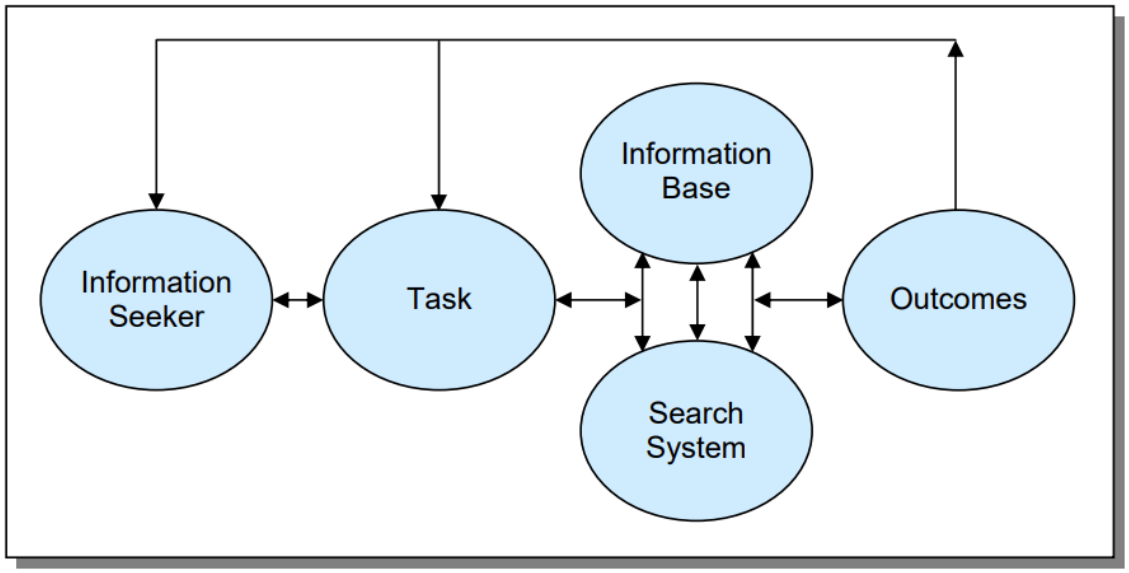

Figure 3: Model of exploratory search process taken from R. W. White, Muresan, and Marchionini 2006 Workshop on evaluating exploratory search systems

purchases and bookmarking can be used as a metric to estimate user engagement and enjoyment levels.

2. Information novelty: As mentioned in the information-seeking problem, one of the intentions behind exploration is to discover previously unseen elements, so it appears reasonable to incorporate the quantity of new information encountered as a measurement of the effectiveness of an exploratory search System. Additionally, we can include the rate at which users come across new information as a significant aspect of novelty that could offer further insights into interaction efficiency.

3. Task success: Unlike look-up tasks, exploratory search tasks are not completed when finding a particular document, and thus its success should not be measured based on that. In the workshop, researchers agreed that task success should be measured on whether the searcher has received enough information and adequate detail.

4. Task time: Once a task is completed, measuring the time it took to finish was suggested to indicate exploration activity effectiveness. In the workshop, multiple time measurements were suggested like, total time spent, time spent

looking at irrelevant documents, and the proportion of time spent engaged in direct search versus time spent exploring.

5. Learning and cognition: Learning plays a crucial part in the exploratory search. By measuring cognitive and mental loads, the achievement of learning objectives, the comprehensiveness of a user's perspective after exploration, the extent of topic coverage, and the number of insights generated, there's a possibility of comparing exploratory systems based on their learning and cognitive aspects.

2.4.3 Exploratory search evaluation methods

After discussing the search models, their evaluation metrics, and their differences, we now focus on the exploratory search model. After learning what we want to evaluate and according to which metrics, we now discuss how we are going to evaluate it.

The evaluation will primarily help to assess the success of the information-seeking process so that future designs of retrieval systems would be backed up by data (R. W. White, Muresan, and Marchionini 2006). Even though search engines now support exploratory search tasks, their evaluation is still limited to systems that only minimally consider human-machine interaction (Singer et al. 2011). And thus in this thesis, we propose software that bridges this gap and offers helpful functionalities for researchers evaluating exploratory search systems.

We discussed the system's model and its success metrics. To evaluate our system, we need to come up with a methodology that considers our model and its metrics (R. W. White, Muresan, and Marchionini 2006).

Existing methods for quality measurements focus predominantly on the search system and not on the search process. Therefore they only gather implicit searcher's actions, which are the taken actions by a searcher while searching like opening a tab or putting a search query. On the other hand, explicit user feedback, e.g. opinion on the success of a search session, is rarely collected Singer et al. 2011. Since the model and metrics of the look-up systems are different, their evaluation method won't be suitable for exploratory search systems especially since they do not include the searcher as an actor, which plays a big part in ESS and needs to be integrated into the evaluation method Singer et al. 2011. Still, the ESS evaluation method needs to include implicit information gatherings, like page visits, queries made, and time spent on pages to get a complete view of user's behaviour, especially since look-up searches are embedded into exploratory searches as we mentioned in 2.3.2

We shed light before on the contradictory nature of look-up searches and exploratory searches, which also (Singer et al. 2011) indicates, when quoting B.Jansen, by saying that there is a tension between information searching and information

retrieval but also a convergence towards each other can be noticed. Lewandowski and Höchstötter 2008 introduced a framework for measuring search engine quality that encompasses both a system-centric and a user-centric approach, arguing that for measuring user experience empirical studies are essential. The framework considers nonetheless index quality, result quality, search feature quality, and search engine usability as key factors in assessing search engine performance.

Doing experiments in a laboratory environment allows for a controlled environment for all participants but it has its limitation, in particular when it comes to sample size. Lab experiments allow only small sample sizes, which does not suit user-based experiments that require large sample sizes to have a representative sample. Furthermore, exploratory search tasks could span minutes, hours, or even days (Singer et al. 2011), which makes a lab experiment less suitable. One way to avoid these problems would be to use log file analysis, which unfortunately comes with problems in that the log files are anonymous and lack user demographic information. Log files have another disadvantage: they gather data from a specific website, for example, a single specific search engine, where we would like to investigate users' actions outside of the borders of one website (Singer et al. 2011). Gathering logs in multiple websites would also allow users to use the search engine they prefer.

Another solution would be to integrate software in the form of a browser extension that can be used on searchers' computers on their own time. In addition to solving the previously mentioned problems, this approach will allow researchers to capture the natural changes in searchers' information needs and search strategies (R. W. White, Muresan, and Marchionini 2006). This will provide researchers with information taken from real-world scenarios in natural conditions. A browser extension can log implicit data, e.g., used search engines, browser interactions, etc., and it allows the collection of explicit data through questionnaires pre- and post-task. This is a crucial feature because it considers the exploratory search success metrics, suggested in R. W. White, Muresan, and Marchionini 2006 and mentioned earlier in 2.4.2. Through implicit data gathering, we can evaluate the *engagement and enjoyment, task success* and *task time* metrics and through explicit data gathering, we can evaluate the *information novelty* and *learning and cognition.*

Furthermore, the extension can offer a precompiled set of search tasks that gives the searchers a clear goal to achieve and thus serve as a basis of comparison between the different users. After deducing the testing tool's general features, we now search for existing solutions and study if they fit our needs.

3 Existing Solutions

This chapter will discuss the existing software tools that support quality measurements of exploratory searches. We will discuss their advantages and disadvantages shortly and if they satisfy our needs. A comprehensive but not exhaustive list was created by Singer et al. 2011. I will depend on it in this chapter and expand it where ever needed.

There have been some tools that are said to support exploratory search evaluation. A powerful tool is called "Wrapper" by Jansen et al. 2006. The Wrapper is a desktop app created to execute studies for evaluating exploratory search systems. Since it's a desktop application, it can log user-system interactions, like interacting with the system clipboard, browsers, and other applications including Microsoft Office or email clients. Browser interactions include bookmarking, copying, saving, and scrolling (Jansen et al. 2006). With all of the wrapper's powerful features, it does not satisfy our needs, since it does not collect the searcher's explicit data like user feedback and cannot provide the user with precompiled search tasks (Singer et al. 2011).

Some other tools like HCI browser, Curious Browser, and Webtracker have been either discontinued or do not satisfy our needs of logging explicit actions and providing precompiled search tasks. Another powerful tool is LogUI by Maxwell and Hauff 2021. It offers advanced functionality for tracking events related to nodes on a webpage, such as a link or a search engine result page (SERP) element, and tracking events related to the webpage as a whole like resizing the browser's viewport or taking the browser out of focus. LogUI needs to be configured with a configuration file containing CSS standard selectors to tell the tool which elements to track in the DOM i.e. on the webpage (Maxwell and Hauff 2021). As flexible and powerful as this is, the researchers must have prior knowledge of the DOM and CSS selectors. In addition to this caveat, it does not collect explicit searcher's data.

YASBIL by Bhattacharya and Gwizdka 2021 is an open-source browser extension based on the MDN documentation for cross-browser compatibility. YASBIL supports multi-device search since a user can log in and requires no prior knowledge of the DOM and CSS selectors, but also does not support explicit collection of data and does not provide searchers with a precompiled set of search tasks.

The closest to satisfying our needs are the following two applications. The first software is a browser plug-in by Fox et al. 2005. This plug-in is the first tool to collect implicit and explicit information during searches. It evaluates the query level by gathering explicit searcher's feedback after each query and it also comes with a sophisticated analysis environment for the logged data. The disadvantage of it is

that it's only compatible with internet explorer, which is almost out of service now and it is not publicly available.

Search logger by Singer et al. 2011 is the closest to our needs. It is a browser plug-in built around the definition of an exploratory search task being an open-ended, abstract, poorly defined information need with a multifaceted character. It logs information on the query level and provides the searcher with precompiled search tasks. Furthermore, it collects demographical information from the user and takes explicit feedback using pre-, and post-task questionnaires. In addition to that, it also collects users' interactions with the browser and the plug-in. The search logger is also compatible with the RAT (Relevance assessment Tool) by Sünkler and Lewandowski 2017 which helps researchers to create a study with tasks and questionnaires and analyses the collected data from the search logger. However, the search logger was created in 2011 and uses the old Manifest v2, which is no longer supported by Chromium browsers *Manifest V2 [Deprecated]* 2020. The search logger's functionality heavily influences this thesis's suggested solution and introduces a new GUI and flexibility in some areas of the application that will be discussed in the next chapter. Furthermore, it will consider cross-browser compatibility and work with the new version Manifest v3, that's supported by Firefox browsers (*Manifest v3 in Firefox* 2023) and is the only version supported by Chromium browsers (*Chrome Manifest V3* 2022).

4 Explicit Implicit Logger

This chapter presents the proposed solution of this thesis, which will be referred to as EI-Logger. We first go into an abstract overview of the solution, then we delve into design decisions, data structures, and features. The Explicit Implicit Logger is a browser extension designed as a helping tool to collect logs for log-analysis studies and execute user-centric studies that focus on browser usage. The EI-Logger is intended to help researchers evaluate exploratory search systems and has the flexibility to adjust according to their needs inside and outside the realm of exploratory search tasks. Due to the context in which this software was developed and the time constraints of the bachelor thesis, EI-Logger is still a young software with a lot of room for optimisation and extension.

4.1 Predevelopment phase:

This phase was full of literature search, reading, and summarizing. This allowed me to be mindful of my exploratory search to collect the needed knowledge. Besides literature about exploratory search I also had to read through the documentation of

the Mozilla MDN web docs to gather information about extension development and cross-browser compatibility. After collecting enough theoretical examples, I decided on programming languages, frameworks, and libraries.

Since the extension is going to work on the browser, it has to be programmed with JavaScript language. JavaScript is a dynamically typed language (MDN-contributers 2023b), which means that the variables' types get decided at run-time instead of at compile-time or develop-time based on the variable's value at that time (MDN-contributers 2023a). This caveat and a decent number of other disadvantages make JavaScript a poor language for developing and maintaining large applications. This is why I chose to work with TypeScript instead. Not only does it get compiled into JavaScript so every browser can run it, but it also offers a more stable and pleasant development experience and solves a lot of JavaScript's problems, in particular dynamic typing (Bierman, Abadi, and Torgersen 2014). The static type system of TypeScript helps catch and mitigate mistakes statically while in development time and helps to give IntelliSense suggestions for programmers for object methods and variables, or while filling the parameters of a function (Bierman, Abadi, and Torgersen 2014).

For development, TypeScript can be used alone or with a framework. When developing with Vanilla TypeScript the DOM has to be directly manipulated, whereas by libraries such as ReactJs, the React Virtual DOM gets manipulated through React components (Persson 2020), which, in my opinion, makes the development experience more pleasant. According to the Stack Overflow Developer Survey for 2022, ReactJs enjoyed fifth place in the most loved web frameworks and technology (Stack-Overflow 2022). A boilerplate project for developing browser extensions was used to facilitate the setup of the TypeScript/React project (JongHak-Seo 2023). This boilerplate had all the troublesome configurations already created, from the compiler and bundler settings to the manifest and linting settings, which played a crucial part in saving a lot of time and allowed me to focus on the extension architecture and development.

To address the development in a systematic and organized manner, I also used a version control system. The VCS Git in combination with GitHub allowed me to separate the functionalities into branches, which kept the project organized and protected against faulty changes (Clone EI-Logger repository Al-Mustafa 2023). Furthermore, when saving changes, commit messages were written with explanations, so that the next programmers who will continue the work, can go back and see the reasoning behind some changes. In addition to that, Jira was used as an application lifecycle management service. This facilitates the conceptual design in breaking down the design into stories and tasks. The following programmers can see the finished stories and tasks, especially if there's a need for more explanation about some design

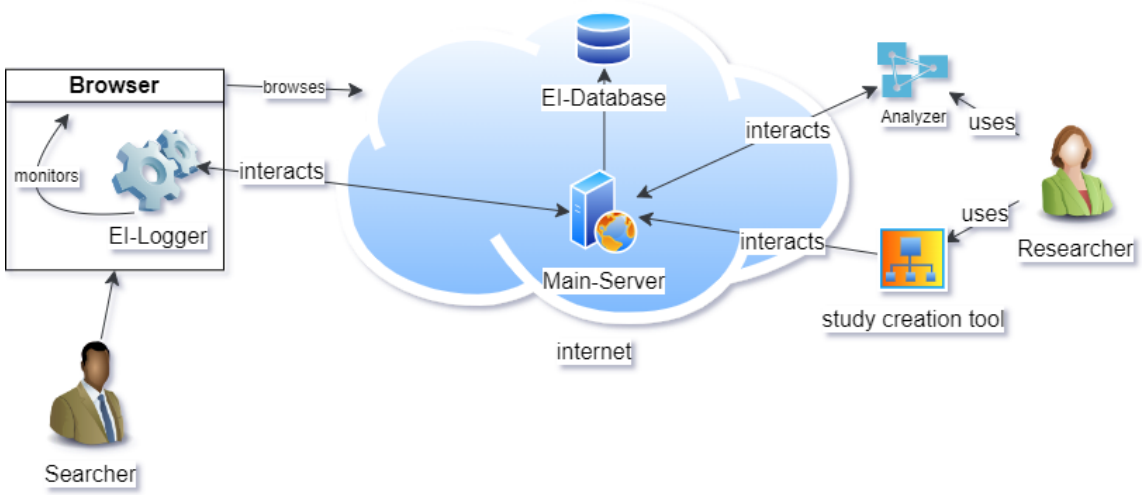

Figure 4: Evaluation system architecture

decisions. They will also see some undone stories that serve as suggestions for future functionalities. Those functionalities will be discussed in chapter 6 (*Future Work*).

4.2 System architecture:

The EI-Logger is thought to be a component in a software package. This thesis only handles the EI-Logger and sheds light on the environment it's supposed to work in. As previously mentioned, exploratory search tasks could span minutes to days and therefore an alternative or a supporting method to lab studies is needed. The logger is a browser extension based on the MDN cross-browser documentation. It can be installed on Firefox and Chromium-based browsers. More on browsers' compatibility can be found in section number 4.4.4.

The EI-Logger connects with the "Main Backend" server to register and authenticate users, fetch study data, and deliver the logged information. The searcher can interact with the EI-Logger through its GUI which can be activated through an icon in the browser's toolbar. When the logging function is started, the user can execute their search uninterrupted from the extension as it disappears at the searcher's will and continues logging in the background.

The server interacts with the "EI-Database" to store and call logs, user data, and study configuration. So far, the discussed parts are implemented in this thesis, and we'll go into their details in the coming sections. The other Elements of the environment are the analysis tool and the study creation tool, which are not part of

this thesis. The analysis tool reads the logged data and extracts information from them. It is abstracted as a component in this figure, still, it can be a complex system with different entities, for example, the client that allows the researchers to adjust the analysis parameters and then see the results. The last element is the study creation tool, which represents a client that enables researchers to create studies and create anonymous and non-anonymous user IDs.

Although the last two discussed elements are not implemented, the "Main Backend" is implemented for testing purposes and offers an end-point that receives study configuration as a JSON file and other end-points to create and authenticate user IDs.

4.3 Main-Server Architecture:

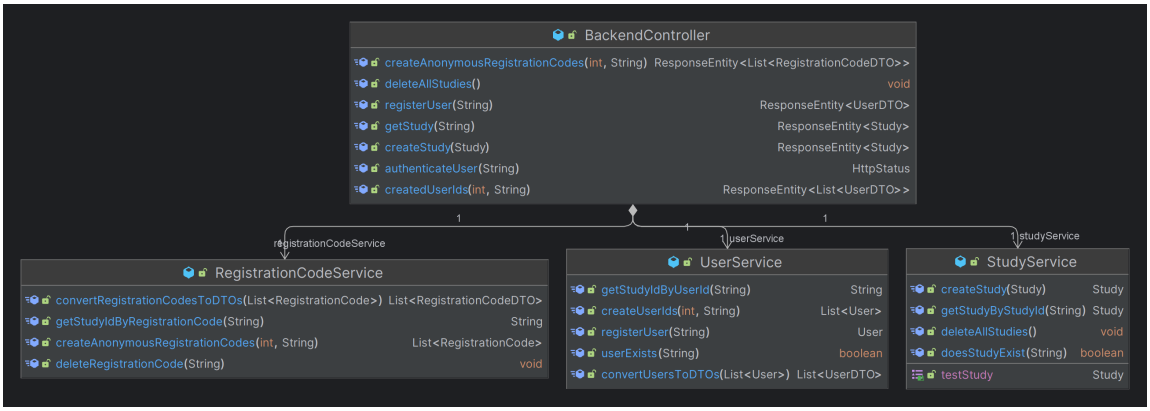

Figure 5: Excerpt of Main-Server's services

The Java Spring Boot server offers REST endpoints as an API allowing the EI-Logger client and other clients like Postman to interact with it. The server offers services that mainly serve the EI-Logger extension client, so there are no services e.g. for log analysis and no detailed endpoints for creating a study, however an endpoint "createStudy" receives a study as a JSON file. The structure of the Study will be discussed later. The server offers three services that can be viewed in Figure 5:

1. RegistrationCodeService allows researchers to create one or multiple anonymous registration codes to distribute to study participants. Those codes are associated with a study id and will be persisted in the database to be used to register users. When a searcher gets a registration code, they will be able to enter it in the EI-Logger client to authenticate with the server. The server will

check for the code's validity and get the studyId from the RegistrationCode object with fetStudyIdByRegistrationCode and upon success, a new user ID will be created using the UserService.registerUser and sent back to the user, thus guaranteeing user anonymity in the registration process if that is wished. The registration code will then be deleted from the database using deleteRegistrationCode.

2. UserService generates user IDs for users or researchers if they want to create non-anonymous registration codes and associates the codes with a study id. Furthermore, they authenticate a user ID with userExists when users log in.

3. StudyService creates a study from a JSON file and serves study configuration to users upon registering or logging in.

4.4 EI-Logger:

The EI-Logger is a flexible browser extension client that adapts to the researchers' needs through the study design. The client gets the study configuration as a JSON file from the server and adapts its GUI based on the study. The provided flexibility was implemented due to the variety of needs we discussed in the existing solutions. Some tools provided implicit logging but no explicit logging, some provided explicit logging but no pre-compiled set of tasks, and some provided a set of precompiled tasks but did not collect demographic information. The EI-Logger will optionally provide all possible combinations of those requirements based on the study configuration provided. As the Study creation tool is outside this project's scope, the study has to be fed to the server as a JSON file. I added API-Requests with two study configuration files in the source code as Postman files for testing purposes. The configuration of these studies will showcase the different features of EI-Logger and I will use those studies to explain the workflow of the EI-Logger in the next section.

To summarize the various needs discussed in previous chapters and to have a more detailed understanding of the client, I will list the requirements of the EI-Logger.

4.4.1 Requirements:

- General constraints

    1. The extension will be implemented in a client-server architecture.
    2. The communication between the client and server will be carried out on HTTP using JSON.

3. The extension's client will be written in Typescript using the React framework.
4. The extension's server will be written in Java using the Spring Boot framework.
5. The server will be stateless and thus scaleable and will be able to afford a large number of users.
6. The extension will be useable with the most famous Browsers. It should work at least on Firefox and Chrome Browsers.
7. The extension will recognize at least Google and Bing search engines.

- Functional Requirements

1. The extension will guarantee an anonymous registration process.
2. The extension will provide the ability to register using a generated ID provided by the researcher.
3. The extension will provide the ability for searchers to sign in.
4. The extension will provide a secret auto-generated ID, upon successful registration.
5. The searcher will be able to view their ID from their signed-in client.
6. The extension will be able to show a precompiled set of tasks.
7. The extension will implicitly log various searcher's interactions with the EI-Logger and assign a timestamp to the logs:
   (a) SIGNED:UP
   (b) SIGNED:IN
   (c) OPENED:DEMOGRAPHICS
   (d) SUBMITTED:DEMOGRAPHICS
   (e) STARTED:TASK
   (f) FINISHED:TASK
   (g) OPENED:PRE_QUESTIONNAIRE
   (h) SUBMITTED:PRE_QUESTIONNAIRE
   (i) OPENED:POST_QUESTIONNAIRE
   (j) SUBMITTED:POST_QUESTIONNAIRE
   (k) STARTED:LOGGING
   (l) STOPPED:LOGGING

8. The extension will identify and log queries and search engine names with time stamps.
9. The extension will associate logs with the active task if a task is active.
10. The extension will consider the following events:
    (a) TAB:OLD This event points to the logs, containing the tabs that were opened before the searcher started the logger. Already logged tabs won't be duplicated after starting the logger for the second time.
    (b) TAB:CREATED
    (c) TAB:CLOSED
    (d) TAB:URL_CHANGED this event occurs when a user changes the URL of an opened tab.
    (e) TAB:ACTIVATED an event to indicate switching tabs
    (f) TAB:BOOKMARK:ADDED this event occurs when a tab is bookmarked.
    (g) TAB:BOOKMARK:REMOVED this event occurs when a tab is removed from the bookmarks.
    (h) TAB:PINNED this event occurs when a tab is pinned to a window.
    (i) TAB:UNPINNED this event occurs when a tab is unpinned from a window.
    (j) TAB:ATTACHED:TO:WINDOW
    (k) TAB:DETACHED:FROM:WINDOW
11. The extension will have the option to provide users with a precompiled set of tasks.
12. The extension will have the option to provide the user with a precompiled set of questions.
13. The extension will be able to recognize if a search query was given to a search engine.
14. The extension will be able to extract the search query given to a search engine.
15. The extension will be able to recognize a search engine result page.
16. The extension will be able to log search engine result page contents.

4.4.2 GUI:

Unlike most browser extensions, the EI-Logger has multiple GUI pages with a complex transition flow that's flexible and adjustable via the study configuration. The pages are navigable as a typical web application, only with a smaller view box. A diagram that represents the possible transitions is seen in Figure 6.

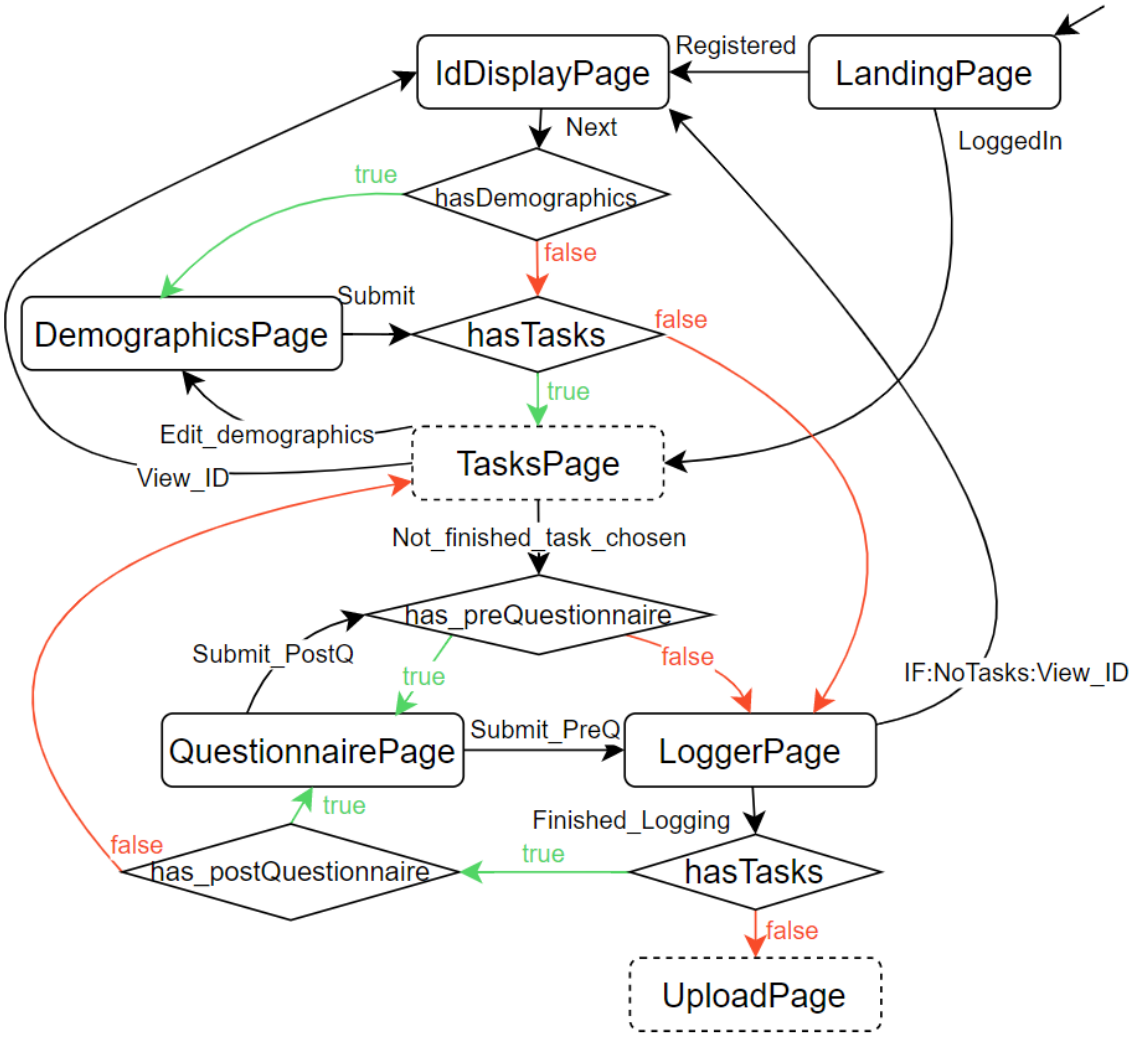

Figure 6: GUI transitions

I will use the two example study configurations I mentioned earlier in 4.4 to explain the transition diagram. Study 1 represents the goal study that we were

evaluating existing solutions upon. It is a study for evaluating Exploratory search tasks. It has a set of four pre-compiled tasks for the searcher to do and some tasks have questionnaires that represent the explicit part of the logging in addition to collecting demographic information.

The workflow for the searcher will be as follows. The searcher will get an anonymous registration code from the researcher, which the user will use on the LandingPage to register. After successful registration, the user will get another code "userId" that gets randomly generated by the server, so no one can practically map the userIds to the users. In the IdDisplayPage the user will see the newly generated ID and be prompted to copy and save it somewhere safe. From there the user will be directed to fill out the demographics form since this study contains demographics, which consist of the date of birth, job title, and gender. After successful submission, the user sees the tasks included in this study on the TasksPage. The first task contains no questionnaires so when the user clicks it, they will be directed to the LoggerPage where they can start and stop logging at their will. After finishing the task they will be directed again to the TasksPage since this task contains no post-questionnaire.

The QuestionnairePage shows a set of precompiled questions and displays either pre- or post-questionnaire questions. Depending on which questionnaire is being shown, the QuestionnairePage has a different transition flow.

The second task contains a pre-questionnaire so the user will be directed to the QuestionnairePage before landing in the LoggerPage and in the third task the user lands on the LoggerPage and then the QuestionnairePage to answer the post-questionnaire questions as it has no pre-questionnaire. In the fourth task, we have pre-, and post-questionnaire, which will lead the user to the QuestionnairePage twice, once before going to the LoggerPage and once after.

The second study has no demographics and tasks, making the user skip from the IdDisplayPage directly to the LoggerPage. Note that the DemographicsPage and IdDisplayPage can be accessed later at the user's will. When the study has tasks, then the user will see buttons to access the IdDisplayPage and DemographicsPage on the TasksPage. Only when the study does not have tasks, the buttons will appear on the LoggerPage. Of course, the option to visit and edit the demographics will only be available if the study has demographics.

The transition graph in Figure 6 represents a state machine in the extension, where the state is locally saved in the browser's local storage. This allows the extension to remember on which page it was and enables it to open the last opened page before it got closed. The transition between states or pages is done programmatically after the user fires an event e.g. a submit event. The extension will recall

existing data on demand, when the user enters the demographics page, after having submitted an answer, the user will see the old submitted information.

The questions in the QuestionnairePage have three forms. The researcher can decide to include any combination of the forms and any number of questions and the client will dynamically handle the forms and number of questions. The Forms can be viewed in Figure 7.

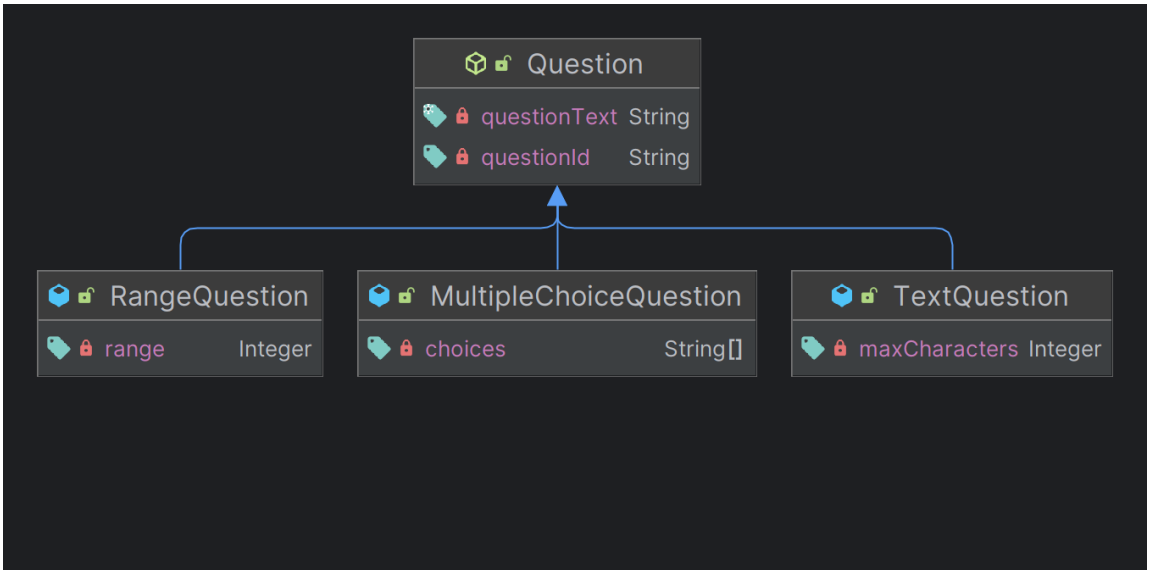

Figure 7: Class Diagram of the Questions

All Questions seen in 7 have a questionId to uniquely identify the question and questionText. Their differences are explained in the following list.

- RangeQuestion has the range attribute. A use case of this form could be questions that have an answer represented as a scale.

  An example of such a question is *"From zero to nine, how hard do you think finding an answer to the task will be?"* or *"The task will be very hard in my opinion. 1 means I disagree, 5 means I completely agree."* The scale is then displayed as a numbered slider like in Figure 8

- MultipleChoiceQuestion has the choices attribute, which is an array of the possible answers to the questions chosen by the researcher and it has no limits on the number of options.

Figure 8: Ranger question slider

- TextQuestion This form has a free text answer form with the maxCharachters attribute, limiting the length of the given answer.

Aside from the GUI transition logic, I also focused on usability and user experience to make the EI-logger easy to use as well as on the visual design to make it appealing and enjoyable to use. The extension has a minimal GUI with hover effects to give feedback when interacting with GUI elements like buttons and list elements. A notification system has also been designed to alert users to problems when they arise or to provide feedback when success occurs.

4.4.3 Logged data structure

The data won from the EI-Logger will be saved in a database to be analysed. The data will have multiple attributes that allow the grouping of records from single or multiple studies for a more general view of all studies. All records in all database tables will contain a study ID and user ID therefore they will be omitted from the following list that explains the structure of the logged data.

1. demographics: are the personal data we collect about the user and they entail the following attributes: (a) birthDate (b) job (c) sex

2. answers: are the answers to the questionnaires and contain the following attributes: (a) taskId (b) questionId (c) answer

3. tabs: represent the user's interaction with the browser from the point view of a tab and it contains the following attributes: (a) taskId (b) timestamp (c) action (d) tabId (e) tabUuid (f) groupId (g) windowId (h) tabIndex (i) title (j) URL (k) query (l) searchEngineName (m) serpIdentifier

   The action attribute represents user events mentioned in the functional requirement number 10.

   tabId denotes the local tab id in the browser, which is unique in a browser session, which means once all browser windows are closed and a new session starts,

the tabId could be duplicated. Therefore a universally unique id (tabUuid) was added.

In case the searcher used the grouping function in the browser, the groupId attribute will give us the ID of that group.

If the user had multiple opened windows the windowId will let us keep track of the used window. Users also search using multiple tabs and the tabIndex will tell us which index a tab has in the corresponding window.

The title and URL give us extra information about the tab and if it was used to open multiple web pages, which will be noted using the action

TAB:URL_CHANGED (see 10d).

When a user visits a search engine or platform like Reddit and uses a query to search for something, the query and search engine's name will be extracted and saved using the query and searchEngineName attributes. The EI-Logger recognizes 29 search engines and platforms, that are mostly used: • Google • Bing • Yahoo • Yandex • DuckDuckGo • Brave • AOL • Gibiru • Wikipedia • Ekoru • Archive • Archive: Way back machine • Ecosia • Reddit • YouTube • Givewater • WolframAlpha • Baidu • ASK • Qwant • Search.ch • Dogpile • Metager • Excite • Metacrawler • Mojeek • Sogou • Zoo • Swisscows.

After the extension has identified a search engine and a query, the HTML containing the search result will be saved in the table serpHtml. To associate the saved HTML with the record from this table the serpIdentifier is used.

4. serpHtml: After identifying the search platform and the query used, we need to also save the returned result page so we can analyse the basis of following user decisions. Unfortunately, getting a clean data set from e.g. a search engine result page (SERP) is not very reliable, since search engine providers constantly change their HTML tags containing the SERP information, which makes it very hard to scrape. A solution is using SERP scraper paid APIs, which can be expensive but also not useful because it does not reflect what the searcher saw since the result of search engines e.g. Google depends on some variables like browser cookies and location (*How Google uses cookies* n.d.).

   An API call cannot consider users' cookies since it has no access to them and therefore won't return what the user gets. An alternative is to save the inner HTML of the SERP and analyse it later considering the changes in the HTML tags. Additionally, the inner text of the HTML is saved, which will provide the textual content the user sees on the page, which should simplify the analysis.

It includes the following attributes: (a) timestamp (b) inner HTML (c) inner text (d) serpIdentifier.

5. userExtensionInteraction: represents the interactions between the user and the EI-Logger and maps the actions from the requirement 7. It includes the following attributes: (a) action (b) timestamp (c) task ID.

4.4.4 Compatibility with other browsers

The EI-Logger is written using the cross-browser API provided by Mozilla MDN WebExtension, which is the extension technology for Firefox browsers. This API is to a large extent compatible with the Chromium-based browsers Extension API. By development, the compatibility table provided by MDN WebExtension was considered to choose only API calls that are compatible with all major browsers like Firefox, Chrome, Edge, Opera, and Safari. To take advantage of all EI-Logger's features, the use of the latest browser version is recommended.

According to the MDN documentation, the Safari browser is supported. Unfortunately, a test on Safari could not be done because of the following two reasons. First, my development environment is running on Windows and Safari is not supported on Windows (*Safari not supported on Windows* n.d.). Second, to test the extension on an Apple Mac, the extension needs to be run through Apple's IDE XCode to be able to be uploaded to Safari. This would have required additional research time to gain knowledge in XCode and Safari, which was unfortunately not available due to the thesis' time constraint.

Having said that, I will discuss a compatibility obstacle that was solved during development. When installing an extension, the browser first reads a .json file called manifest.json, which expresses the structure of the extension. It tells the browser, among other information, which files to read for what purpose and what permissions are needed. In 2018 Chrome announced Manifest Version 3 (*Chrome Manifest V3* 2022) and as of the 9th of November 2020 (*Manifest V2 [Deprecated]* 2020), the Chrome extension documentation stated that they are abandoning Manifest Version 2 and replacing it with Manifest v3. Unfortunately, the Firefox developers did not follow along quickly and only implemented Manifest Version 3 in Firefox 109 in mid-January 2023 (*Manifest v3 in Firefox* 2023). This integration was not seamless, since there are still some parts of the Manifest Version 3 that Firefox does not support. A crucial example is the *background*. Version 3 changed the Manifest definition of the background service from containing the *scripts* key that takes an array as a value to the *service_worker* key that takes a single string value *Chrome Manifest V3* 2022. This inconvenience means that any extension using the background service will have

to be built using two manifest versions, one containing the *"scripts"* key and one using the *service _ worker* key.

4.4.5 Exceptional cases and services handled by the software:

- As mentioned, if there are no tasks, then the buttons for viewing the user ID and editing demographics are displayed on the logger page.
- Once a task is finished, it cannot be started again.
- Once a questionnaire is submitted, it will disappear and thus users won't be able to change their answers after starting and pausing the logging.
- If the user changes their demographics, the old demographics on the local and remote databases will be overridden
- Once the logging starts, the logger will log all of the already opened tabs. If the logger gets paused and then restarted, only the new tabs that were opened while the logger was paused, will be saved.
- The EI-Logger communicates success and error events via notifications. For success events, the user will get a green snackbar at the bottom of the EI-Logger's GUI and for error events a red one.
- The logging state (ON/OFF) is shown on the extension's symbol in the toolbar.
- The logging function is toggled between on and off through a play/pause button that changes colour according to the logging state.

4.4.6 Known limitations

1. Uploading to the server is not yet fully implemented. This functionality is in progress and discussed in *Future work*.
2. When uploading the extension as a developer to some browsers like Opera and Firefox, a certain flag needs to be activated for the logger to be able to read the HTML content of websites and especially the search engine result page. I will explain how to activate the options for Firefox and Opera:
   (a) In Firefox, the option is to be found on the "about:addons" page. After clicking on the EI-Logger icon, choose the "permissions" option and activate the option "Access your data for all websites".

(b) On Opera after opening the "Manage extension" page, look for the EI-Logger and activate the option "Allow access to search page results".

3. Switching between tabs that are attached to different windows does not trigger the TAB:ACTIVATED event because it uses a different API. Although we could contact change window API, it won't give us the tab information and thus no added value.

4. Logs about creating a new tab are considered once the tab finishes loading, before that no events on the tab are logged. An improvement proposal is discussed in chapter 6 (*Future work*).

5. When saving the HTML content of the result page, some sensitive data might be saved as well, exposing the searcher's identity in case the identity was to stay anonymous. An improvement proposal is discussed in chapter 6 (*Future work*).

6. When the searchers' identity is not wished to be anonymous, the researchers will distribute user IDs that the searchers can use to directly log into the EI-Logger. This represents the main usage scenario for the login function. Another scenario would be supporting multi-device logging, where the user can use different devices to execute the study, which will reflect the real usage of people who own multiple devices. Although the user can currently login from a second device, the multi-device support is not fully functional due to a lack of synchronisation between devices. For example, if a task is completed on a device, it should be completed on all devices, which is not the case yet. An improvement proposal is discussed in chapter 6 (*Future work*).

To conclude, the EI-Logger provides support for log-analysis studies that are user-centric in collecting implicit searchers' data like the visited websites and bookmarked tabs and gathering explicit data like demographic information and searchers' feedback on tasks or used tools via questionnaires that can be presented before and/or after the task has been started. The questions have three forms to choose from and a questionnaire can contain any combination and number of question forms. Furthermore, the EI-Logger allows researchers to observe a searcher's search behaviour in the context of a task or outside of it in real-world scenarios since it can be deployed inside and outside of laboratories. The EI-Logger is a browser extension that conforms with the MDN extension API, which allows it to work on all major browsers. The ability to function in all major browsers and outside of the lab environment makes it ideal for supporting large-scale studies. The EI-Logger can be preset using

a study configuration provided by an external server, where also the logs can be uploaded for further analysis. The EI-Logger offers robust functionality with attention to small implementation details in addition to error handling. Moreover, it offers a pleasant graphical user interface, that's enjoyable and easy to use. The mentioned features make the EI-Logger compatible to support studies dealing with exploratory search evaluation because it offers implicit and explicit logging that is necessary to gather data concerning the exploratory search systems evaluation metrics discussed in section 2.4.2.

5 Evaluation

In this chapter, I will talk about the evaluation of the EI-Logger. For the EI-Logger to be considered successful it has to check some boxes. For example, robustness, correctness, and user experience.

5.1 Automated testing

Unit tests are ideal for checking the robustness and correctness of a software's functionality. On the backend, we have REST endpoints that are written in Java Springboot. Writing unit tests for those endpoints is straightforward and was easily implemented. Unfortunately, that was not the case for the EI-Logger. Although writing unit tests for a ReactJs application should also be straightforward, no unit tests were written for the EI-Logger due to a technical problem, that hindered the writing of any unit test. After multiple trials and research no answer could be found so I decided to abandon unit tests for the EI-Logger and adopt another form of testing.

5.2 Field tests

Field tests help evaluate various aspects of the software in a real-life situation. This field test will have a small sample size and therefore cannot give a definite judgment about the correctness and behaviour of the software, but it will provide an initial judgment and an overview of the state of the software and the trajectory of development that has been taken so far.

For testing, five users are going to be selected and given the software to test it on their local machine on 5 different browsers. For using the EI-Logger a user must either register or sign in, which requires a backend. Although the backend exists, it must be available for all five test users on their local machines. There are two ways

to go about this. The first one involves me giving the backend to the users and they run it on their local machines, but that would either mean giving them the source code, which will require them to download and set the needed environments for the server to run, or to dockerize the server with the backend, which will mean they would have to download docker and have some knowledge to use it. Considering the mentioned obstacles I decided to go against this approach. The second approach is to deploy the backend on a cloud server and make it available on the internet, which would relieve the users from all those obstacles. I needed to find a cloud host offering a free tier to do that. After some search, a couple of cloud hosting providers were found but the process for deploying the server onto the services was not beginner friendly and needed some experience either with the platform, docker, or deploying in general, which I do not have a background in. Although I invested a good amount of time in this matter, I realized it would require more investment, and because there is an alternative, I also decided to go against the deployment idea. Since we mainly depend on the server only for authenticating and fetching the study configuration, we could mock those functions in the code and thus get rid of the server's dependency. This alternative is doable since the functionalities we are not testing are relatively low in complexity and were covered in the backend's unit tests. Yet still we are testing the higher complexity components.

The field tests should inspect for GUI elements' correct transition and rendering. Moreover, they should examine if the GUI elements behave and function as they are supposed to. In addition to that the core logging functionality should be checked. Besides the functionality of the EI-Logger, I will ask for users' feedback on the usability and design.

Checking the correctness of the GUI's functionality and behaviour is straightforward. By recording the user's screen during the test, I can later compare the user's workflow, based on the video, with the recorded logs. I can check for GUI bugs while watching the user's screen during the test. At the end of the test, I will ask the user to visit the IdDisplayPage and DemographicsPage (if exists) manually and I will ask for feedback on usability and design.

The EI-Logger will be tested on two operating systems and five browsers by contacting five friends. One user will install the EI-Logger on MacOS and the rest on Windows operating systems. Furthermore, the five users will use a different browser from the most famous browsers. Chrome, Edge, Opera, Brave, and Firefox are the chosen browsers. Although users should be free to choose the search engine they most

feel comfortable with, each user will be prompted to use a different search engine for testing purposes. The search engines used are Google, Bing, Yahoo, DuckDuckGo, and Brave.

Since the registration process will be mocked, all users will get the same userID. Each user will get one of three studies. The first Study demands filling up the demographics and has a single look-up task without any questionnaires. The users will be asked by me to manually go into the IdDisplayPage and DemographicsPage, change the demographics, and save their changes. The text of the look-up task will be *"When was Penicillin invented and by whom?"*.

The second study does not require filling up the demographics and has a single exploratory search task with pre- and post-questionnaires containing all three available question forms. The exploratory search task will have the following text: *"Mr Johnson and his family are planning a trip to Paris. Please write a summary of the following: What cultural events are there (for example on August 2023)? What sightseeing to do? Please also cover hotels, flights, travelling to Paris, and weather."*

The task will have a pre-questionnaire of a single question of the form RangeQuestion with the text *"From one to five, how fast, in your opinion, will you find the answer?"*, and a post-questionnaire of two questions, the first of the form MultipleChoiceQuestion with the text *"After trying to use the internet to get your information, would you rather get that information from, a) the internet, b) a social network like Facebook or Reddit where you post your question and expect people to answer, or c) through a travelling agent?"*. The second question will be of the form TextQuestion with the text *"What was the hardest part of finding your answer?"*.

The third study will contain no tasks or questionnaires, which means after registering the user will be directed to the LoggerReadyPage to start logging outside the context of a task.

5.2.1 Metrics and criteria of success

To evaluate the EI-Logger through the tests, we have to set some metrics and criteria to judge it by. The following list contains the three chosen metrics:

1. Compatibility: The compatibility test entails two parts: compatibility with different operating systems and browsers. This metric will be cleared once the EI-Logger has been installed and initially operated on the different proposed operating systems and browsers.

2. Functionality: This is a big umbrella term housing multiple aspects. It is expected that the users will face no problems using the EI-Logger and that it will show the correct GUI pages and offer the correct options based on the study configuration. The EI-Logger will render the correct task, questions and buttons and will log and save the user's information.

   In addition to that the extension must correctly recognise the used search engines, queries, and SERPs. It is also expected that the extension will log the user's interaction with the extension and browser and will also log all queries and web page visits. To check this functionality the user's screen will be recorded during the test and later compared against the saved logs. The logs will be read out from the browser's local IndexedDB. Moreover, the logger must save its state and always open the last opened GUI page before it was closed.

   User's feedback on functionality and possible bugs will be verbally taken and the logs and workflow will be manually checked from the screen recording as described before.

3. Usability: This is a subjective metric and is solely based on the user's opinions and feedback. The EI-Logger should be easy and intuitive to use and must behave how the user expects it to behave. Most importantly the user must not be interrupted by the EI-Logger while logging e.g. the logger must not be in the way or deteriorate the browser's performance.

5.2.2 Examining the result of the field tests

In this section, we will list the result of the tests. All listed problems and enhancement suggestions were corrected unless otherwise noted.

1. Compatibility: The EI-Logger was installed on Windows and Apple computers. The users managed to successfully operate the EI-Logger on Chrome, Edge, Firefox, Brave, and Opera. On Opera and Firefox, the users needed to manually allow the extension to have access to the HTML content of web pages, especially search engine result pages. The way to manually do that is described in section 4.4.6.

2. Functionality: Some bugs were detected during the tests. The most intriguing problem was that Bing and Google searches weren't getting recognised by the logger. The reason for that was apparently that Google and Bing, and possibly other search engines, use different URL structures depending on where the

query came from. If the query came from the text field on the website, then the key that indicates the query in the URL is "?q" but if the query came from the browser's search bar then the key will be "&q". This problem was solved by reducing the key to the invariant part of it, which is the "q".

However, one bug remains unfixed. During the test, a user wanted to use YouTube beside Google, which led to the first unsolved bug. Usually when navigating inside a website, for example going from the "Main page" to the "About page", the website will invoke the onCompleted event, which indicates that a tab finished loading its content. YouTube does not invoke the onCompleted event when e.g. opening a video. The EI-Logger depends on this event to track when a tab is opened or has changed its URL. This problem is solvable by leveraging the onChange event which detects URL changes. Although YouTube invokes this event and it can solve the problem, it could also unfortunately cause some duplication in the logs due to interference with the onCompleted event and how the EI-Logger handles those events. Due to time constraints, this problem stands unsolved.

After solving the discovered Bugs, the EI-Logger performed all its functionality correctly. It navigated through GUI pages correctly and displayed buttons that depend on variables to be rendered in the right way. In addition to that, it handled both studies with and without tasks, as expected. The EI-Logger logged all user's actions that it should log. Moreover, it identified the search engines used for the tests and logged the query and returned SERP successfully.

3. Usability: In the context of usability, all users found the EI-Logger easy and intuitive to use. Although there were some improvement suggestions, the styling was pleasant for all users. Some users presented suggestions that were considered and implemented. One suggestion was to display the task's text on the LoggerReadyPage so that users get familiar with the task and remind themselves of some aspects of it without stopping the logger and going back to the TasksPage. An Information panel was added to the TasksPage to give a hint about choosing a task to execute and forgetting to start the logger after a task has been chosen. Another user suggested some changes to the warning text phrasing and colouring. Aside from the suggestion, there were two inconsistencies in rendering the EI-Logger on Firefox. The first one was that the input elements' labels were in black instead of grey, which didn't affect the readability. The second one was that the data picker symbol was rendered but not

interactive, which presents no functional problem since the date input consists of a date picker symbol and a date input field, which is still operable.

In total, the unit tests were implemented for the backend but not for the EI-Logger due to technical problems. The field tests gave us a positive initial impression of the compatibility, functionality, and usability of the EI-Logger. The field tests uncovered some bugs that were fixed and improvement ideas that were implemented. The EI-Logger is compatible with the major browsers and search engines, delivers the functionality that's presented in the functional requirements and provides flexibility for the researchers, which was discussed in the previous chapter, and offers an intuitive and pleasant experience to the users.

6 Future Work

As mentioned before, the EI-Logger is still a young software and has a large room for improvement. Before, I covered that the EI-Logger is a single part of an undeveloped environment. In this chapter, I will shed light on features that were in the backlog but did not make it to the development process.

1. Uploading logs to the server: The uploading functionality is in progress but not yet fully implemented. The feature branch for uploading can be found in the GitHub repository (Clone the EI-Logger repository Al-Mustafa 2023) and on the Jira board (To get access to the Jira board please contact me on hossam.almustafa0@gmail.com). The progress made so far is committed to the branch. If wished the next developer can continue with the suggested implementation or start completely new.

2. Internationalization: To cover a wider range of possible study probands, multi-language support can be added to the EI-Logger. The EI-Logger could depend on the browser's chosen language but also can offer the option to switch languages mid-execution. The EI-Logger is written in React which has a large body of libraries and helpful documentation for achieving this goal. Although this idea is easy to implement, it is not trivial in the case of the EI-Logger. The internationalization of EI-Logger would not only mean internationalizing the displayed hard-coded texts but also translating the text of the tasks and questionnaires, which would mean either integrating a translation library or API or prompting the researcher to manually do so, which is ineffective. Since

there is still no client for creating a study, integrating internationalization in EI-Logger at the moment brings no benefits, however, this could be implemented in further development of the extension.

3. Multi-device synchronization: the searcher can log in from multiple devices. To make the multi-device experience better, both devices need to synchronize so that if a task or a questionnaire is completed on one of the devices, it is also finished on the other devices. Furthermore, a device ID can be added to associate the logs with the device they were logged from.

4. Logging tabs before they finish loading: Tabs are considered only after they finish loading for the first time, before that they cause no event by the logger. This is a limitation that prevents logging some possible events that could happen before a tab finishes loading, such as switching to or closing the tab before it has finished loading. An API to detect tabs once they get instantiated exists and can be utilized to improve the extension's logging abilities.

5. Idle state detection: a scenario where the user left the logger on and forgot about its activity is possible and disadvantageous to the ongoing experiment and user. The disadvantage comes from the case where the user continues his or her normal browser use after the extension has been forgotten. This might expose the supposed anonymous user's identity and add dirty logs to the database that are irrelevant to the ongoing task.

   Although the extension's icon on the toolbar always shows the logging state, either ON or OFF, this scenario could still occur. An idle state detection combined with a notification signal, will surely solve the problem. An experimental time value could be set and adjusted or analysed for the idle state from previous logs. Another option is allowing the user to set the idle time manually. As for the notification, the already implemented notification snackbar is not helpful in this case, since it only appears on the EI-Logger GUI pages, therefore, the notification should appear as a system notification. Luckily the MDN notification documentation offers a solution that can be explored.

6. Logging Clipboard activity: Logging clipboard read and write activity is a feature that's implemented in other loggers like the Search Logger (Singer et al. 2011) among others. Adding it to the EI-Logger will only strengthen its feature collection and make it more appealing and helpful to researchers. Relevant resources can also be found in the MDN clipboard documentation

7. More options for demographics: currently the information demanded from the demographic page is the date of birth, job title, and sex. These were the obvious pieces of information during the development of this feature, but certainly, there will be other study-specific information that will be required or rather omitted. Giving the researcher the ability to choose from several available attributes or better yet, letting them create their attributes, is a useful feature to add. Currently, the date of birth is collected through a calendar interface that can be also adjusted through a text field. The sex is collected through a drop-down list and the job title is through a text field. As the questionnaire questions were generalized through three input forms, the same generalization could be done for the demographics input forms.

8. Extracting search engine information to the server: At the moment, the search engine information is hardcoded in the EI-Logger. To enable researchers to easily manipulate this list, we can store the list on the server and make the EI-Logger call it, when it needs to start logging.

9. Increasing security, privacy, and transparency: Security and users' data protection should come as a priority but it was unfortunately time-consuming to implement. In the following, I'll propose four optimizations that the EI-Logger and the server can use:

   (a) HTTPS: The communication between the EI-Logger and the server uses HTTP. Replacing that with HTTPS will protect user data with an extra layer of encryption.

   (b) Excluding URLs: Sometimes the user might go into private websites open like E-Mail providers. Not logging those pages adds an extra layer of data privacy for the user and protects their identity even more.

   (c) Viewing logged Data: Users have the right to know what data is logged about them. To guarantee this right for users and increase the transparency between them and the extension, the EI-Logger could be extended with a function that allows the user to view the logs saved in the database.

   (d) Excluding the searcher's E-Mail from the saved SERP: if the logger detects a SERP, the HTML content will be saved. In that content user's data might be inside, like name and/or E-Mail address. Finding a way to exclude those information or be more precise in saving SERP information offers more security and robust functionality.

7 Conclusion

This thesis introduced a new logging tool, that supports log-analysis and user-centric studies. The explicit and implicit logger (EI-Logger) is a browser extension that is compatible, in particular with studies for evaluating exploratory search systems.

We decided on the implemented functionality in the EI-Logger by understanding the two information-extracting methods, namely information retrieving and information seeking. We learned that search engines support to a great degree information retrieval, that answers questions based on "when", "where" and "who", talked about the need to go beyond this model and motivated the information-seeking model that also answers questions starting with "what", "how", and "why". We went into detail explaining the two parts of the information-seeking model which are the information-seeking problem and the information-seeking processes. We shed light on the differences between the information-retrieval and information-seeking models and what their evaluation metrics and methods look like. After that, we focused more on the information-seeking evaluation methods and deduced the functionality needed for the evaluation tool from the features of the information-seeking model. We then discussed similar existing tools and judged them based on the deduced features.

Based on the deduced functionalities and the shortcomings of the discussed tools, the EI-Logger puts forward three main functionalities. The first functionality offers researchers the option to provide searchers with a precompiled set of search tasks, which allows the researchers to study users' events in a specific context of their choosing.

The second functionality is implicitly collecting logs about events triggered by users while engaging in search processes. Events such as opening a new tab, visiting a new website, or interactions with the browser like pinning or bookmarking a tab. The implicit data logging will allow researchers to gain the necessary information to understand people's search behaviour. The EI-Logger is compatible with at least the major browsers such as Chrome, Edge, Opera, Firefox, and Safari. It also recognises the queries searched on 29 search engines and platforms like Reddit and Wikipedia.

The third functionality focuses on the user as an actor by explicitly collecting the user's feedback on certain topics, especially on the provided search tasks. These explicit logs are collected through questionnaires, which could be displayed before and/or after the logging process and can contain any number of questions that have the following forms: RangeQuestion which renders a slider with a numeral lower and

upper limit, TextQuestion renders an input text field that allows a certain amount of characters, and MultipleChoiceQuestion which renders any number of predetermined choices.

The core functionality was tested through field tests, which gave a positive result. Besides that, the test participants endorsed the graphical user interface of the EI-Logger saying it is intuitive and easy to use.

8 Eigenständigkeitserklärung

Ich, Hossam Al Mustafa, Matrikelnummer 3053109, versichere durch meine untenstehende Unterschrift, dass ich die vorliegende Arbeit selbstständig ohne fremde Hilfe angefertigt habe und dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus fremden Quellen entnommen sind, entsprechend als Zitate gekennzeichnet habe und dass ich ausschließlich die angegebenen Quellen (Literatur, Online-Ressourcen, sonstige Hilfsmittel) verwendet habe.

Hossam Al Mustafa, Essen den 10, Juli, 2023

References

Bard (n.d.). https://bard.google.com/. (Accessed on 07/06/2023).

Bhattacharya, Nilavra and Jacek Gwizdka (2021). "YASBIL: Yet Another Search Behaviour (and) Interaction Logger". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. DOI: 10.1145/3404835.3462800.

Bierman, Gavin, Martín Abadi, and Mads Torgersen (July 2014). "Understanding TypeScript". In: vol. 8586, pp. 257–281. ISBN: 978-3-662-44201-2.

Blair, D. C. (June 1992). "Information Retrieval and the Philosophy of Language". In: *The Computer Journal* 35.3, pp. 200–207. ISSN: 0010-4620. DOI: 10.1093/comjnl/35.3.200. eprint: https://academic.oup.com/comjnl/article-pdf/35/3/200/1406180/35-3-200.pdf. URL: https://doi.org/10.1093/comjnl/35.3.200.

Chrome Manifest V3 (Sept. 2022). https://developer.chrome.com/docs/extensions/mv3/intro/. (Accessed on 07/06/2023).

Fox et al. (Apr. 2005). "Evaluating implicit measures to improve Web search". In: *ACM Trans. Inf. Syst.*, pp. 147–.

How Google uses cookies (n.d.). https://policies.google.com/technologies/cookies?hl=en-US. (Accessed on 07/07/2023).

Jansen, Bernard J et al. (2006). "Wrapper: An application for evaluating exploratory searching outside of the lab". In: *EESS 2006* 14.

JongHak-Seo (July 2023). *Jonghakseo/chrome-extension-boilerplate-react-vite: Chrome Extension Boilerplate with React + Vite + Typescript*. https://github.com/Jonghakseo/chrome-extension-boilerplate-react-vite. (Accessed on 07/06/2023).

Lewandowski, Dirk and N. Höchstötter (Jan. 2008). "Web Searching: A Quality Measurement Perspective". In: pp. 309–340. ISBN: 978-3-540-75828-0.

Manifest V2 [Deprecated] (Nov. 2020). https://developer.chrome.com/docs/extensions/mv2/. (Accessed on 07/06/2023).

Manifest v3 in Firefox (Mar. 2023). https://extensionworkshop.com/documentation/develop/manifest-v3-migration-guide/. (Accessed on 07/06/2023).

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Marchionini, Gary (Apr. 2006). "Marchionini, G.: Exploratory search: from finding to understanding. Comm. ACM 49(4), 41-46". In: *Commun. ACM* 49, pp. 41–46.

Maxwell, David and Claudia Hauff (2021). "LogUI: Contemporary Logging Infrastructure for Web-Based Experiments". In: *Advances in Information Retrieval:*

43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 – April 1, 2021, Proceedings, Part II. Berlin, Heidelberg: Springer-Verlag, pp. 525–530. ISBN: 978-3-030-72239-5. URL: https://doi.org/10.1007/978-3-030-72240-1_59.

MDN-contributers (June 2023a). *Dynamic typing - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing. (Accessed on 07/06/2023).

— (July 2023b). *JavaScript | MDN*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/. (Accessed on 07/06/2023).

Al-Mustafa, Hossam (July 2023). *EI_Logger_BA*. https://github.com/Samustafa/EI_Logger_BA.git. (Accessed on 07/09/2023).

OpenAI (n.d.). https://openai.com/. (Accessed on 07/06/2023).

Persson, Morgan (2020). "JavaScript DOM Manipulation Performance: Comparing Vanilla JavaScript and Leading JavaScript Front-end Frameworks". In: p. 40.

Safari not supported on Windows (n.d.). https://support.apple.com/en-us/HT204416. (Accessed on 07/06/2023).

Singer, Georg et al. (Mar. 2011). "Search-logger analyzing exploratory search tasks". In: pp. 751–756.

Stack-Overflow (2022). *Stack Overflow Developer Survey 2022*. https://survey.stackoverflow.co/2022#section-most-loved-dreaded-and-wanted-web-frameworks-and-technologies/. (Accessed on 07/06/2023).

Sünkler, Sebastian and Dirk Lewandowski (Oct. 2017). "Does it matter which search engine is used? A user study using post-task relevance judgments". In: *Proceedings of the Association for Information Science and Technology* 54.

White, Ryen and Resa Roth (Jan. 2009). *Exploratory Search: Beyond the Query-Response Paradigm*. Vol. 1.

White, Ryen W., Gheorghe Muresan, and Gary Marchionini (Dec. 2006). "Report on ACM SIGIR 2006 Workshop on Evaluating Exploratory Search Systems". In: *SIGIR Forum* 40.2, pp. 52–60. ISSN: 0163-5840. URL: https://doi.org/10.1145/1189702.1189711.