Integration of a Logging Component into the STELLA Infrastructure

Bachelor thesis submitted for the degree

Bachelor of Sciences in the course of study Data and Information Science

at the Faculty of Information Science and Communication Studies

of TH Köln – University of Applied Sciences

Submitted by: Anh Huy Matthias Tran

Submitted to: Prof. Dr. Philipp Schaer

Second assessor: M.Sc. Timo Breuer

Cologne, 9.06.2022

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum

Rechtsverbindliche Unterschrift

Abstract

As a key part of human-computer interaction(HCI) and usability testing, the capturing and recording of key user interaction plays a center role for ensuring a reliable post-hoc analysis of collected user interaction data, thus improving the odds of insightful HCI and usability testing cycles for use cases such as the evaluation of interactive information retrieval Systems(IRR). As such, the practice of logging is of significant importance for multiple fields of study such as IIR, HCI and most recently also Living Lab approaches. Living lab approaches represent a user-centered research methodology with a focus on user involvement, experimental approaches and extensive collaboration for the sake of co-production of knowledge and as such, has a dire need for robust and easy to use logging solutions.

With past logging solutions being either expensive, hard to use or error-prone, recent conferences gave rise to new logging solutions using contemporary web technologies, which aim to improve the logging landscape within the research community. Over the course of this paper, two of these recent logging solutions, LogUI and Big Brother, are to be inspected for their key features and then evaluated, whether they are suitable logging solutions for living lab and IIR environments. Results and research indicate, that both logging solutions offer significant benefits for research using living lab and IIR approaches, with LogUI embracing many of the experimental paradigms that guide the living lab approach.

Keywords: Logging Infrastructures, Logging, User Interaction, User Study, User Behavior, Living Lab, Interactive Information Retrieval

Abstract/Deutsch

Als ein wichtiger Teil von Mensch-Computer Interaktion und Usability Testing, spielt das Aufnehmen von wichtigen Nutzeraktionen eine Hauptrolle, um eine zuverlässige Analyse von Nutzer Interaktion Daten zu versichern. Deswegen ist das Feld des Loggings/Aufnehmen sehr wichtig dafür, um ertragreiche Einblicke bei dem Bewerten von interaktiven Information Retrieval Systemen(IRR) zu erreichen. Daher ist das Feld des Logging wichtig für mehrere Studienfelder wie z.B. IIR, HCI aber auch der neuen Reallabor Methodik(Living Lab). Living lab verfolgt eine nutzerzentrierte Recherchemethodik mit einem Fokus auf Nutzerinteraktion, experimentelle Methodiken und starke Kollaboration zur Zusammenerstellung von neuem Wissen, und hat deshalb ein großes Verlangen für einfache Logging Lösungen

Dabei sind die alten Logging Lösungen heutzutage leider entweder teuer, schwer benutzbar oder fehlerhaft in der Ausführung, weswegen kürzliche Konferenzen mehrere neue Arten von modernen Logging Lösungen vorgestellt haben. Über den Lauf dieser Abschlussarbeit, werden zwei dieser neuen Logging Lösungen, LogUI und Big Brother, sich näher angeschaut und danach ausgewertet, ob diese neuen Logging Lösung angebracht für Living Lab und IIR Probleme wären. Die Ergebnisse und Recherche sagen aus, dass beide Logging Lösungen sehr starke Vorteile in einer Living lab oder IIR Umgebung abgeben würden, insbesondere LogUI, welches das experimentelle Recherchieren von Living Lab Methodiken stark mit eigenen Funktionen fördert.

Schlagwörter: Logging Infrastrukturen, Logging, Nutzerinteraktionen, Nutzerstudien, Nutzerverhalten, Reallabor, Interaktives Information Retrieval

Acronyms

API Application Programming Interface

CSS Cascading Style Sheets

CSV Comma-separated values

DOM Document Object Model

ECIR European Conference on Information Retrieval

GUI Graphical User Interface

HCI Human-computer Interaction

HTML HyperText Markup Language

IIR Interactive Information Retrieval

IR Information Retrieval

JSON JavaScript Object Notation

LL4IR Living Lab for Information Retrieval

SERP Search Engine Results Pages

SIGIR Special Interest Group on Information Retrieval

STELLA InfraSTrucutrEs for Living LAbs

UI User Interface

WWW World Wide Web

Contents

List of Tables

List of Figures

1 Introduction

The capturing and recording of key user interactions between users and systems through logging is fundamental for analyzing user behavior in web-based Interactive Information Retrieval(IRR) systems.

Current methodology for experimental systems, however, often involve researchers developing their own logging infrastructure, potentially leading to implementation mistakes or incomplete or noisy data gathered, making subsequent analysis of user behavior more error-prone and difficult than needed. Standard solutions to these problems are currently either expensive, complex to use, or not open-source. Recent conferences have introduced new logging frameworks such as LogUI (Maxwell and Hauff 2021) and Big Brother (Scells, Jimmy, and Zuccon 2021), which offer a modernized approach to logging and as more and more services and research projects adopt an online approach in tackling their challenges, the need for robust, easy to deploy and scalable logging solutions rises.

This project paper is directly involved with the 'STELLA - Infrastructures for Living Labs' project (Breuer and Schaer 2021), an infrastructure for evaluating different retrieval and recommender systems in accordance to a living lab approach (Azzopardi and Balog 2011). With STELLA and its living lab approach, different kinds of IIR systems search engines and search algorithms are evaluated for their performance by gathering and analyzing live key user interactions gathered from different kinds of research participant groups.

As such, the paper proposes the following research question:

RQ1 Do currently new logging frameworks satisfy the requirements and needs of web-based interactive information retrieval systems within a living lab approach?

In order to inspect this hypothesis and approach this project, the paper will first define concepts such as IIR, living lab and human-computer interaction(HCI) in order to identify and quantify their specific needs, followed by an in-depth inspection of logging as a historic concept. Following that, the paper will inspect and compare the key features of the following logging frameworks that have been recently introduced in international conferences such as ECIR [1] and SIGIR [2]:

- LogUI

[1] https://www.ecir2021.eu
[2] https://sigir.org/sigir2021/

- Big Brother

And then evaluate these according to their functionality, deployability and scalability. Finally, both logging frameworks, LogUI and Big Brother, will be integrated into the STELLA infrastructure in a live demo application and made available in a GitHub repository[3]. This allows live key user interaction to be gathered through these logging frameworks and the performance of these frameworks to be evaluated and compared in aspects concerning reliability and granularity within a live experimental web environment.

2 Tools and Resources

2.1 Docker

Docker[4] as an application platform is used for distributing and deploying application services in a contained manner. This enables docker users to easily deploy multiple kinds of applications in isolation with each other in an easy and scalable manner. Docker's separation of application is achieved through the creation of docker containers running on singular Linux instances, thus requiring far less resources than virtual machines while receiving the same benefits of a virtual machine. As such, Docker is used to enable the multi-container-application nature of the STELLA framework with easy deployment, as well as logging server instances from LogUI.

2.2 STELLA

STELLA[5] is a multi-container-application which provides a user-focused evaluation approach combining the concepts of the living lab methodology with IIR application (see section 2.3). It allows researchers to deploy and evaluate multiple IR systems within a live web environment by accepting the submission of either pre-computed runs or fully dockerized IR systems called STELLA APP[6]. This allows for full reproducibility and easy deployment of multiple, different STELLA APPS and their respective IR systems in order to test different conditions.

The STELLA APP allows the implementation of multiple IR Systems called 'BASE' and

[3] https://github.com/AH-Tran/STELLA_LogUI
[4] https://www.docker.elastic.co/
[5] https://stella-project.org/
[6] https://github.com/stella-project/stella-app

'EXP' within the same application, allowing researchers to easily deploy different IR implementations within the same docker environment. Upon the call of STELLA APP's indexing-endpoint, the application builds its index with the provided CORD-19 data set, after which the STELLA APP can receive and process query requests. Any query request are then sent to both the BASE IR system and the experimental EXP IR system, which then both return their respective list of relevant document-ids. The resulting search engine result page (SERP) is an interleaved mix of results from both the BASE IR system and the EXP IR system. This enables researchers later to differentiate which IR systems gained the most clicks from any participating end users, allowing them to draw judgement on the performance of either IR system.

2.3 Flask

Flask[7] is used as a micro web framework written in python, often used for web experiments for its simplicity. It does not require a large arsenal of additional tools and libraries to function, allowing researchers to build web applications for experiment purposes with fewer difficulties. In this paper, Flask was used to create the web application interface for sending queries and displaying search results retrieved from STELLA APP in a visually digestible manner and to provide a web page upon on which the implemented logging frameworks could perform log any user interaction.

2.4 CORD-19 Data Set

The 'CORD-19'[8] data set consists out of a collection of scientific articles related to COVID-19. It is retrieved from the 'ir_datasets' python package[9], provided for the purpose of a common interface and data set for IR evaluation tasks. The data set contains following fields such as doi, title or abstract.

2.5 LogUI

LogUI is a logging framework recently introduced at the SIGIR conference, allowing researchers to easily implement a server-client architecture solely dedicated for logging purposes into their web experimental apparatus. With its detailed documentation, LogUI

[7] https://flask.palletsprojects.com
[8] https://ir-datasets.com/cord19.html
[9] https://github.com/allenai/ir_datasets

offers a wide array of customization ranging from user interactions, DOM events, browser events and metadata sourcers, while also facilitating experimental measures such as A/B Testing through the implementation of applications, flights and sessions.

2.6 Big Brother

Big Brother is another logging framework recently introduced at the SIGIR conference. Similarily to LogUI, it is also reliant on a server-client architecture, this time built in Go[10]. It is optimized for easy integration into web apps and allows for the capturing and recording of user interactions, while providing and even encouraging integration into external services such as the ELK Stack[11] for further feature improvement concerning log analysis.

3 Theoretical Concepts and Foundations

From an overall perspective, the practice of logging user interactions takes place in the context of use cases, wherever the evaluation of users, their behavior and their relation towards the specified user interfaces is important. This applies especially for the use case of focused user studies, where the practice of logging enables researchers to capture and record the interplay between a given test participant and the experimental apparatus that needs to be evaluated for their performances.

As a connection piece between the end user and the interface of an experimental system, logging is therefore highly relevant in connecting multiple concepts, fields of studies, and design principles such as interactive information retrieval (IIR), Living Lab and human-computer interaction(HCI) which is why it is necessary to introduce and contextualize these concepts and elaborate on their specific needs in the following subsections.

3.1 Interactive Information Retrieval

Interactive information retrieval(IRR) focuses on the study and evaluation of a user's interaction and their resulting satisfaction with a given information retrieval(IR) system. This stands in contrast to regular IR system evaluation practices with system-oriented ap-

[10] https://go.dev/
[11] https://www.elastic.co/what-is/elk-stack

proaches such as TREC [12].

Just like regular information retrieval, IRR displays a core research interest in IR performance, system interaction and the resulting user satisfaction, but with an additional increased focus on a user's journey as they navigate the IR and therefore acts as a resulting mesh between information retrieval and human-computer interaction disciplines.

As such, IRR aims to inspect a user's journey on multiple levels (Robertson and Hancock-Beaulieu 1992):

- The cognitive level (IR)
- The relevance level (IR)
- The interactive level (IIR)

Cognitive level On the cognitive level, IR systems are evaluated on their ability to receive and process the given personal information need of an end user (the query) and answer it with a result page (Robertson and Hancock-Beaulieu 1992). This is the very basis of information retrieval.

Relevance level Concerning Relevance, it is said that the query given by an end user does not equal the actual information need, which makes it is necessary to evaluate the relevance by how much a retrieved result set satisfies the actual information need beyond the query. Relevance as a metric, however, is ideally not judged in absolute or binary measures(i.e. relevant/not relevant), but with different types of scores representing relevancy according to multiple criteria relevant to an experiment's use case (Robertson and Hancock-Beaulieu 1992).

Interactive level In order to evaluate the interactive level of a given IR system, it is important to track and record a user's journey over the interface of the IR system, inspecting the actual interactive seeking and retrieval process (Robertson and Hancock-Beaulieu 1992). This includes actions such as mouse movements and clicks in relation to the time and date of these actions, and a proper analysis of this level enables a researcher to identify interaction issues, such as hard-to-understand visual elements and functions within an interface.

[12] https://trec.nist.gov/

In order to properly facilitate an IIR environment, where a clean post-hoc analysis of cognitive, relevant and interactive aspects of an IR system is possible, the IIR environment needs a robust evaluation plan and especially a robust logging infrastructure. This enables the researchers of the IIR environment to satisfy their needs, which can be quantified as follows (Borlund 2003b):

- Control (over all variables)
- Observability (over user interactions)
- Repeatability (of results)

3.2 IIR Evaluation Models

In the 2008 ECIR Keynote(Belkin 2008), a general need for a more user-orientedd IR research approach was mentioned, giving rise to alternative evaluation approaches with an increased focus on simulated work tasks(Borlund 2003a). These simulated work tasks are formalized as follows(Borlund 2003a):

Simulated work task situation describing the source of the information need, the environment of the situation, the actual problem that has to be solved and the objective of the search.

Indicative request describing the underlying information need that needs to be satisfied by the aforementioned situation-driven phenomenon

This work task template ensures that an IR system can be judged by the situational relevance of its result set, indicating a usefulness and usability of the information object in relation to the user's information need, as well as satisfy the researcher's need by enabling an IIR evaluation close to the actual user journey when interacting with an IR interface, while still providing a controlled evaluation environment.

3.3 Living Lab

Living Lab represents a user-centered research methodology focused on iterative and open innovation through collaboration with variable scale of purpose, scope and context. This methodology is defined by the following core concepts:

- Experimental approaches in real-life context
- Real participation and user involvement
- Collaboration and co-production of knowledge

This enables researchers to tackle more complex challenges that require the involvement of multiple participants that would otherwise be too hard or unfeasible to approach effectively (FISSAC 2022).

3.3.1 Living Lab and IR

As experimental evaluation is central for IR and IRR environments, recent trends have shifted towards the methodology of online evaluation, where experiments with unsuspecting, real end users take place in natural task environments.

As contrast to standard TREC tasks[13], where expert judgements are used to evaluate the performance of IR systems, IR systems utilizing a living lab environment provide a test environment within a productive online system where the lab can then employ A/B test to try and compare the performance of experimental systems on real user activity (Schaer, Schaible, and Müller 2020).

This results in a collaboration between the online platform and the researchers, with the living lab as a methodological and technical framework for online IR experiments. Recent examples of this methodology put in practice include Living labs for Information Retrieval(LL4IR) (Schuth, Balog, and Kelly 2015) and OpenSearch track at TREC (Yu 2017).

3.4 Human-computer Interaction

Human-computer interaction (HCI) researches the use of computer technology with a focus on the dynamic interplay between the end user and the application/computer by observing how the user behaves on their user journey to satisfy their specific need. This field of research plays a role in bringing multiple fields such as computer science, information science, behavioral studies and (UI) design together, and as such, is a necessary aspect to consider when integrating logging frameworks into experimental environments.

HCI tells developers on how to improve their interfaces and how to evaluate and compare different interfaces in terms of their usability. For this purpose, developers have to

[13] https://trec.nist.gov/

consider the five measurable human factors when interacting with a foreign interface, as outlined by Shneiderman(Shneiderman 1997):

- Time to learn: How long does it take for users to learn their commands?
- Speed of performance: How long does it take to carry out the tasks?
- Rates of errors by users: How many and what type of errors are users prone to make?
- Retention over time: How well do users maintain their knowledge of the interface?
- Subjective satisfaction: How much do users like using the system and its look

The importance of each factor varies depending on the given use case and type of interface system. In the use case of an IIR system within a living lab environment, the factors:

[time to learn], [speed of performance] and [subjective satisfaction]

play an increasingly more important role and should ideally be maximized and validated by a robust logging history of a user's interaction history.

3.4.1 Usability

In an outline defined by ISO[14], usability is defined as

> *"The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use"*

While this description is more appropriately applied to fields such as product design, usability as a measure can also be considered when analyzing the user-friendliness of an IR interface in relation to the end user's behavior. To that end, researchers can choose more IIR appropriate usability measures as outlined by Sandom and Harvey (Sandom and Harvey 2004):

- Effectiveness: The accuracy and completeness in which users achieve specified goals (in %)
- Efficiency: The resources expended in relation to the accuracy and completeness with which users achieve goals (Effectiveness achieved per unit of time)

[14] https://www.iso.org/obp/ui/

- Satisfaction: The comfort and acceptability of use. Retrieved by rating scales in user questionnaires or interviews.

Effectiveness describes the number of goals defined to be achieved by the system in relation to general accuracy and completeness. In the use case of a regular IR system, the IR system's effectiveness can be defined by the number of relevant documents retrieved from the top 20 result page (Sandom and Harvey 2004).

Efficiency is defined by the level of effectiveness in relation to the cost incurred in the process of achieving the aforementioned effectiveness, the cost being measured in either time, effort or monetary costs (Sandom and Harvey 2004).

$$Efficiency = \frac{Effectiveness}{Cost}$$

Satisfaction defines the perceived usability of the IR system, possibly relating and affected by the other two aforementioned measures effectiveness and efficiency (Sandom and Harvey 2004).

The practice of usability testing generally follows the outline as described by Philips and Dumas (Philips and Dumas 1990):

1. Invite the target demographic of people as participants
2. Require the participants to perform typical or critical tasks for the product as intended
3. Observe and record the participant's behavior and actions
4. Use the observation data to diagnose usability problems and recommend solutions

3.4.2 User Interactions

User interactions reflect a user's behavior and journey, as they interact with a system's interface in order to achieve their innate goal. In the context of IIR in a living lab environment, user interaction data is logged from multiple participants within the testing period, which is then promptly used for post-hoc analysis in order to identify common issues or to evaluate the performance of different systems according to predefined metrics such as the usability metrics (see Section 3.4.1).

The concept of user interaction has steadily evolved as more and more measures and methods have been developed to access new kinds of user interactions. Previously restricted to manual questionnaires and video recordings that had to be manually viewed and annotated by researchers, leading to a high amount of work for a limited amount of gained user insight, the recording and capturing of user interaction has evolved as advancements in computer, web and logging frameworks have been made. In addition to commonly known user interactions such as mouse movement and mouse clicks, recent developments even include more advanced user interactions such as touch gestures or direct browser recording (Scells, Jimmy, and Zuccon 2021).

Over the course of recent events, user interaction in web environment can be classified into two kinds of user interactions:

- Coarse-grained user interaction
- Fine-grained user interaction

Coarse-grained user interaction focusing on the capture of the broad view of a user's journey in the form of aspects such as page navigation. It is a field that has been long supported by many commercial logging infrastructures such as Matomo[15] even in the time before web logging has increased in popularity (Maxwell and Hauff 2021).

Fine-grained user interaction In contrast to coarse-grained user interaction, fine-grained user interaction focuses on the smaller details that describe a user's behaviour such as mouse clicks and mouse movement, with this type of user interaction recently getting more and more attention, as it provides a more detailed picture about the end user. This culminates in recent developments of the capturing fine-grained user interaction down to the level of singular DOM elements in a given web interface (Maxwell and Hauff 2021).

4 Logging: An Overview

As more and more research and business groups take up the task of inspecting human factors and usability testing in order to gain further insight into an end user's behavior for the purpose of developing more quality and robust systems and products, the practice

[15] https://matomo.org/

of logging user interactions has been given a steadily rising importance. Previously a field filled with unsystematic feedback loops resulting from manually-performed logging actions such as video recording and annotation, the study field of logging user interaction has grown considerably.

This recent trend is exacerbated by more and more businesses and research groups moving towards performing their services or experiments on online platforms on an increasingly bigger scale with increasingly more difficult challenges and problems to solve. As such, logging is one of multiple important cornerstones of usability research when it comes to IRR environments.

4.1 Logging Data Requirements

As a consequence of the rising importance of logging and the then lack of available commercial options for logging in 1990, Philips and Dumas described and discussed the general functional requirements of data logging software in order to encourage more feature-complete releases of new data logging software (Philips and Dumas 1990).

1. Logging Data As it is not feasible and ideal to record and capture all user interaction taking place, it is important to define **key events** important to the research, with typical key event examples being:

- elapsed time for tasks
- successfully completed tasks
- number of error messages displayed
- time spent navigating the interface

and four different types of data that needs to be defined and recorded:

- discrete events (depicting the start and stop of a task)
- duration of events
- comments made by participants or the test team
- keystrokes/clicks made by the participants

As such, it is important that any prospective data logging software includes a feature that enables researchers to (Philips and Dumas 1990):

1. Describe both the type of the event and provide a code to record these specific events (Custom Event Coding)
2. Create codes to start and stop events
3. Keep track of duration of events, even when multiple events occur in parallel
4. Record extended comments of the participants

2. Editing the Data Log Concerning the process of editing a data log, the data logging software is required to allow the editing of log data in the case of erroneous log data retrieval, ideally as a separate function within the logging software, allowing a researcher to (Philips and Dumas 1990):

- Delete an event
- Change event code
- Change typed comments

3. Backing up the Data In order to facilitate usability tests with a degree of safety, the logging software should be able to save and backup any logs in case of untimely accidents (Philips and Dumas 1990).

4. Analyzing Data The requirements for post-hoc data analysis vary from use case to use case. As such, it is necessary for the resulting log output to be saved in a format, which supports flexible post-hoc usage for a wide variety of analysis use cases. Therefore, the resulting format also necessitates a clear data structure between variables and their respective values, thus allowing programs to more easily parse these logs for insight and context (Philips and Dumas 1990).

4.2 Logging over Time

In the fields of computer science, the concept of logging software has a long history spanning between 1990, where Philips and Dumas discussed and defined the first set of requirements for a functional data logging software (Philips and Dumas 1990) and 2021, where conferences such as ECIR[16] and SIGIR[17] introduced new logging frameworks

[16] https://www.ecir2021.eu/
[17] https://sigir.org/sigir2021/

based on the newest technological developments in Web 2.0 and beyond (Maxwell and Hauff 2021).

As such, the development history of logging methodology can be divided into three overarching categories:

1. Manual capturing
2. Automatic capturing and the introduction of platform specific interaction logging
3. Web-based interaction logging (via proxies, extensions and 'Web 2.0' technologies)

Manual Capturing Initial logging efforts were achieved through the manual observation and annotation of participants performing in video recordings (HOIEM and SULLIVAN 1994). This method displays several obvious drawbacks in comparison to contemporary, modern logging methodologies, including a difficulty of capturing and keeping a pace with each relevant key user interaction, longer time taken and an increasing risk of erroneous note-taking or personal bias skewing the resulting logging data, ultimately leading to the conception of the first automatic logging infrastructures supported by specific event coding practices as mentioned in section 4.1.

Automatic Capturing Following that, with the advent of automatic capturing in the years past 1990, platform specific solutions emerged in the form of logging software that runs as native application on specific platforms such as MS-DOS, Windows and Mac OS X (Maxwell and Hauff 2021). Specific examples include CAUSE (HOIEM and SULLIVAN 1994), ObserverXT (Zimmerman et al. 2009) and InputLogger (Trewin 1998), which focused on the first approaches towards recording specific fine-grained data without the approach of automatically assigning unique identifiers for specific key events. The raw nature of this logging data, however, made post-hoc analysis more difficult than necessary and the platform-specific nature of natively run logging applications restricts each logging software towards their own system, making their use limited in most cases.

Web-based Interaction Logging with Proxies and Extensions In the following years, new logging practices increasingly started to abandon the native application approach in favor of web-based approaches with proper client and server models with the advent of the WWW (Berners-Lee et al. 1994), the DOM[18] and the conception of EventListeners.

[18] https://dom.spec.whatwg.org/

This allowed logging activity to extend past the singular pc environment of a particular participant, eliminatint the need of installing a native application and allowing researchers to run experiments across the world through the web browser rather than a confined, experimental local set up. These early web-based interaction logging solutions, such as RWELS (Shah, Toaimy, and Jawed 2008) rely on proxy-servers that lie between the client and server and achieve logging through JavaScript injection into the relevant to the relevant web pages (Maxwell and Hauff 2021). JavaScript injection, however, is generally considered unsafe and vulnerable to external injection attacks if not properly accounted for (Larson, Liu, and Zuo 2014).

The other type of early web-based interaction logging solution relies on the installation of browser extensions on a client's web browser in order to facilitate logging activity in external web systems, one example being CrowdLogger (Feild and Allan 2013). This extension-based approach, however, increases the barrier of entry by simply requiring participants to download the extension in order to enable logging, shrinking the potential pool of participants, while also increasing the visibility of logging. Increased visibility of logging is another disadvantage ideally avoided when creating the experimental environment, as it makes any participants within the experiment aware of the fact that they are being logged, potentially skewing any following user behavior and the resulting log data set. Another contention of concern is that a general user's confidence in browser extensions might be negative, as the potential perceived danger of inviting malware through a downloaded extension is a real fear that can deter the user from participating in the first place (Ter Louw, Lim, and Venkatakrishnan 2008).

Web-based Interaction Logging with Web 2.0 In the most recent phase of logging practices, new logging frameworks aim to provide precise and customizable fine-grained logging through the use of a client server model, in which a feature-rich API and client-side scripting allows for non-visible logging in web environments without the use of proxies or browser extensions. Examples of this type of logging framework include the recently introduced LogUI (Maxwell and Hauff 2021) and Big Brother (Scells, Jimmy, and Zuccon 2021).

4.3 Choosing a Logging Solution

When considering and evaluating a logging solution, it is useful to consider the requirements proposed by the specific use case of the given experiment in addition to the general

usability principles (Sandom and Harvey 2004) and consider how they would satisfy these requirements and principles in a forward-looking fashion.

In the use case of a **living lab IIR environment** such as STELLA, the following important aspects would be useful to consider when choosing a logging framework:

- Barrier of Entry (from an end user's perspective)
- Visibility
- Effort to learn
- Customization
- Ease of Integration
- Effectivity
- Back Up and Caching Support
- Scalability and Ease of Deployment
- Web Security
- Ease of Post-hoc Analysis

5 Inspecting the Key Features of Logging Frameworks

According to Maxwell, the current landscape of available logging solutions leaves much to be desired, with many of them either relying on obsolete approaches such as browser extensions or code injection, being complex to use, being built in a highly non-modular and coupled manner that makes integration and configuration difficult or having no open source code available. This often leaves researchers to implement their own version of a logging solution while they develop the actual experimental apparatus, with the proper implementation of a robust logging solution often being an after, giving opportunity to faulty and low quality interaction logs, thus making an insightful post-hoc analysis that much harder (Maxwell and Hauff 2021).

While commercial logging solutions such as Google Analytics[19] or Matomo[20] offer a wide array of complete features, spanning from logging coarse-grained and fine-grained data to offering a complete analytics dashboard interface, these solutions are sadly expensive, non-open source and often complex to integrate within existing infrastructure, with

[19] https://analytics.google.com/
[20] https://matomo.org/

Matomo for example being especially non-modular and requiring special HTML classes to enable the recording and capturing of specific elements.

In the following section, the key features of two new logging frameworks aiming to advance and standardize the logging landscape will be inspected: **LogUI** (Maxwell and Hauff 2021) and **Big Brother** (Scells, Jimmy, and Zuccon 2021)

5.1 Key Features: LogUI

In order to alleviate researchers of the pain of implementing their own logging solution or relying on expensive, difficult commercial logging solutions, LogUI[21] was developed in order to offer them a complete, end-to-end logging solution capable of capturing and recording specific fine-grained user interaction data, while taking advantage of modern web technologies. This framework focuses on being an easy-to-use, decoupled, and up-to-date logging framework, available in open source form to researchers.

As such, LogUI focuses on delivering these **key features**:

- Offering a complete **end-to-end logging solution**
- **Framework agnostic** components
- A **simple embedding process** of the LogUI client into any web application
- **Abstraction** of complex design decisions
- A **decoupled, containerized and non-modular infrastructure**
- **Unobtrusive, silent capture** of user interaction
- **Precise and highly customizable event coding** for capturing fine-grained user interaction data

Additionally, LogUI empowers experimental research procedures such as **living lab** by storing interaction data in relation to three central key features: **Applications, Flights** and **Sessions**

Applications can be seen as a particular, overarching service offered to clients/participants that the researcher wants to track user interaction for. The Application can then be further differentiated with the concept of Flights (see Figure 1).

Flights can be seen as the experimental variations of within an application, ideal for experimental research procedures such as A/B testing. This concept, for example, allows

[21] https://github.com/logui-framework

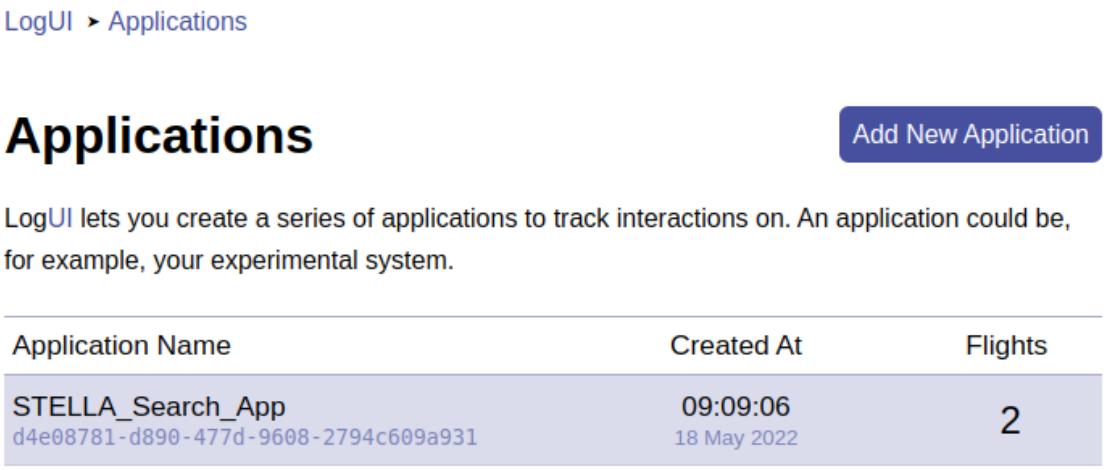

Figure 1 shows an overview of created applications assigned to the LogUI server.

the researcher to deploy the same application multiple times on different host addresses with different configuration, such as different IR systems. This results into an ordered system, where researchers can gather and differentiate user interaction for different experimental branches while still operating under the same main umbrella of the same main application. Use cases include the execution of A/B Testing in the context of two different deployed and actively logging IR systems (see Figure 2).

Figure 2 shows an overview of multiple IR variations belonging to the STELLA LOGUI application.

Sessions are associated with a specific application and flight, and essentially describes a user's unique session on the specified application on the specified flight variation. This allows a researcher to download and inspect logs of a unique session of a unique user on a specified flight with ease (see Figure 3).

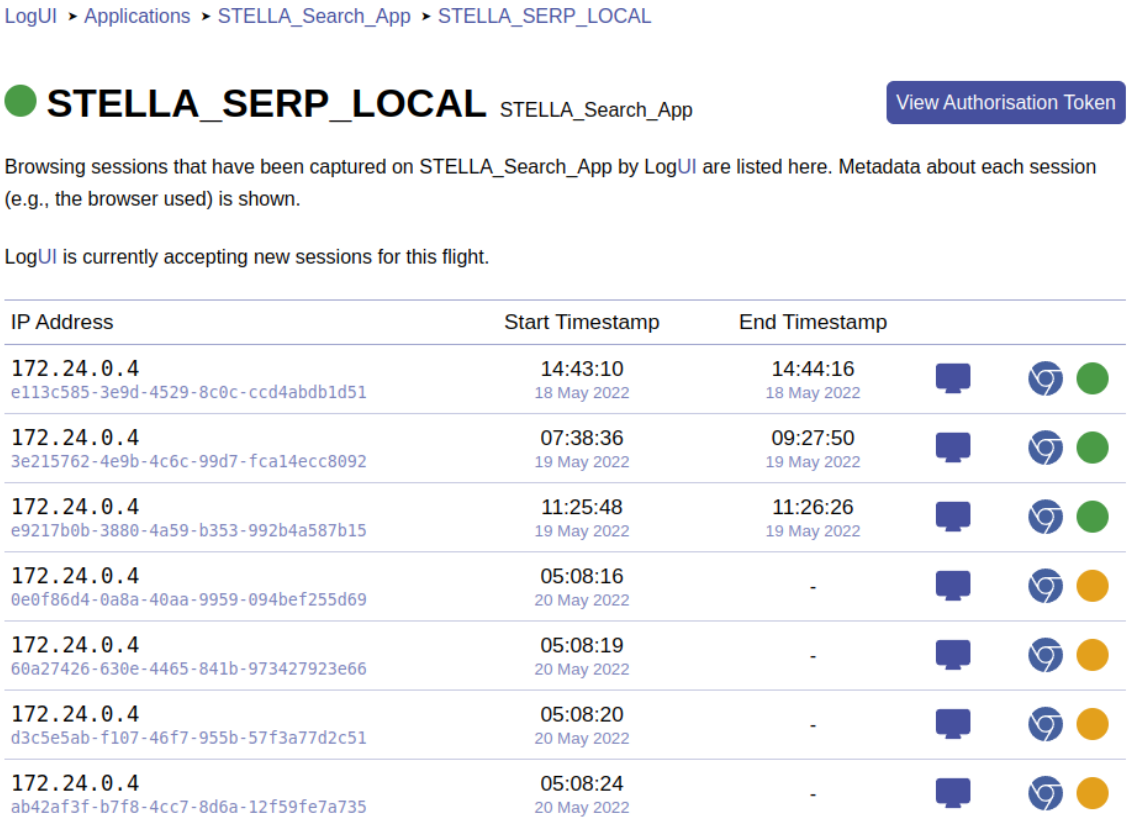

Figure 3 shows an overview over multiple unique session on the flight STELLA_SERP_LOCAL. Currently, three unique sessions are still active and marked as green.

5.1.1 LogUI Infrastructure

LogUI is separated between two main components: The **LogUI client** and the **LogUI server** with a minor **LogUI Control App** responsible for managing settings and applications.

The LogUI client can be used within any web-based application, and is built in a highly modular way. It is created via Node.js in the form of a browserified bundle, which is simply dropped and enabled in the relevant target app responsible for loading the DOM via a script tag, where it actively listens via the Web API for any changes in the DOM and automatically appends listeners to any new elements (see Appendix A.1.).

This makes the LogUI client suitable for many web frameworks such as React[22], as it works with the DOM directly, therefore being largely **framework-agnostic**. As the LogUI client is running, logged events are continuously sent to the **LogUI server**. In case of a temporary internet connection loss, logged events are temporarily cached and then sent altogether to the LogUI server as soon as the internet connection reestablishes itself, thereby avoiding a loss of logging data.

[22] https://reactjs.org/

The LogUI server receives incoming captured events from the client after establishing connection via the web socket. It is fully containerized and completely modular in nature, making it easy to scale up and start up new LogUI server instances via docker.

Any incoming logging data is stored in the **MongoDB**[23] database in a JSON format (see Appendix A.2.).

Client-Server Protocol The LogUI infrastructure allows communication between client and server via Websocket technology (Fette and Melnikov 2011) and executes the client-server protocol with the following procedure by attempting a handshaking routine with the encrypted authorisation token:

- **Case 1: Successful Connection** The LogUI client will now continuously send logging data to the LogUI server.
- **Case 2: Connection lost** The client continues capturing the events and caches them while reattempting the connection. Upon successful reconnection, all cached data will be sent as logging data to the LogUI server
- **Case 3: Connection failed** In case of a wrong websocket-endpoint or an incorrect authorisation token, the connection between the client and the server will be instantly severed. No logging data will be sent to the server.

This is preferable to a regular AJAX implementation, as it allows the LogUI server to reside apart from the client page's host, making it an important feature for experimental web environment research. Additionally, it also preserves the order of data with only one connection, leading to reduced bandwith and resource requirements (Maxwell and Hauff 2021).

5.1.2 LogUI Custom Event Coding Functionality

As originally outlined by Philips and Dumas in the data logging requirements section (see Section 4.1), a strong and robust custom event coding functionality ensures that researchers can define precisely which fine-grained user interactions should be recorded and captured, and thus dictates a logging framework's logging capability as a whole (Philips and Dumas 1990).

For this purpose, LogUI provides multiple constructs, which enable researchers to precisely and clearly define, which key interaction should be recorded in which manner.

[23] https://www.mongodb.com/

The Configuration Object is the heart piece to LogUI's custom event coding functionality. As a JavaScript object, it is automatically passed to the LogUI client once it has initialized. Its key functions include:

- Defining to which corresponding LogUI server endpoint to connect to
- Defining which authorisation token to use for the handshake with the LogUI server
- Defining which browser wide events to track
- Defining which element level events to track

A badly formed configuration object leads to many errors, which is why it is important to get it right. In addition to its very precisie custom event coding functionality, the LogUI configuration object can enrich and label any event with additional data such as **names**, **metadata** and global **application-specific data** for easier post-hoc analysis (see Appendix A.3.).

Furthermore, while LogUI allows the tracking of any kind of DOM Event[24], it also introduces the concept of Event Grouping. Event Grouping provides additional logic to event tracking and groups certain singular events such as 'mouseover' and 'mouseout' into the single event 'mousehover', reducing the level of unnecessary complexity and fine-granularity(Maxwell and Hauff 2021).

5.2 Key Features: Big Brother

Similarily to LogUI, the concurrently developed logging framework Big Brother[25] also aims to improve the logging framework landscape by offering a generic, application-independent logging server for the specific purpose of logging interactions in web pages in relation to user studies (Scells, Jimmy, and Zuccon 2021).

As such, Big Brother aims to provide these key features to potential researchers:

- A logging service with **no barrier of entry**
- With **no configuration necessary**
- A simple **drop-in javascript client**
- A **high-throughput Go Server**
- Capability for **multiple concurrent sessions**

[24] https://developer.mozilla.org/en-US/docs/Web/Events
[25] https://github.com/hscells/bigbro

- And **inbuilt tools** for visualizing interactions
- An easy integration into the **ELK stack**

for those who wish to run simple user studies without the time to invest in learning other logging frameworks for similar results.

5.2.1 Big Brother Infrastructure

Similarly to LogUI, Big Brother also works with mainly two components: A high-throughput **Go Server** and a drop-in **JavaScript client**.

The Client has been built with the intent of achieving maximum functionality while requiring a minimum amount of setup and configuration (see Appendix A.4.). As such, the client listens to all events on all elements by default, but with minimal additional config, the researcher can tell Big Brother to only track certain events such as mouse movement, click events and even a screen capture (see Appendix A.5. & Appendix A.6.).

The Go server is designed to run independently, and processes interaction from users as events, ultimately paying attention to which HTML element was interacted with, how it was triggered in addition to the URL Location, the positional information (in coordinates), time and session ID of the event. As an event happens, the logging data is streamed from the web browser to the centralised server over the connected websocket, enabling real time ingestion of interactions. The go server outputs the log in a local CSV or alternatively offers to directly index the log into Elasticsearch[26].

Additionally, the GO server has already been extensively tested in a throughput benchmark, demonstrating Big Brother's ability to scale up in very large user studies to up to 70,000 events per second, an interaction write-up speed of 5 MB/s and no faults within the output logs occurring during the benchmark (Scells, Jimmy, and Zuccon 2021).

5.2.2 Additional Tools

As an addendum, Big Brother comes with inbuilt extra tools, in order to enable out-of the-box inspection and analysis of the data logs:

The screen capture function works with minimum additional JavaScript code and is enabled by clicking on a specified element and creates a series of images that can later

[26] https://www.elastic.co/

be joined into a video

The bbheat tool can be used to visualise interaction over a displayed user interface through an animated heatmap, allowing extra configuration possibilities concerning actor, time and window size.

The bbstat tool includes filtering, with which researchers can split the logs based on actor, time or aggregation right out the gate.

6 Set Up and Implementation

This section will briefly elaborate on the overall structure and workflow of the integration of logging components into the STELLA architecture. The integration and implementation of both LogUI and Big Brother into the STELLA infrastructure results in an overarching infrastructure consisting of multiple parts:

- The data set CORD-19 containing documents related to COVID
- The STELLA APP with Pyterrier
- A Flask App serving as the SERP interface
- LogUI
- Big Brother

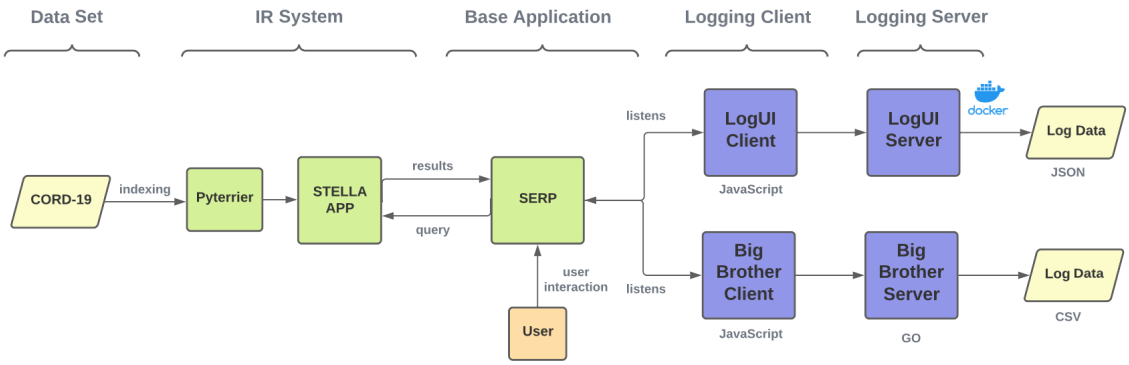

Figure 4 shows the overall interplay between STELLA, LogUI and Big Brother.

As a general overview, the work flow happens as follows:

1. The CORD-19 data set is indexed by Pyterrier within the STELLA APP
2. A user visits the base application and enters a query
3. STELLA receives the query and returns a result list in JSON format

4. The base application receives the result list and gives the user a new SERP

During which the user's interactions are continuously logged by the LogUI and Big Brother client and sent to their respective servers. Afterwards, the new log data can be retrieved from the servers and used for further analysis and visualization of user interaction and behavior (see Figure 4).

6.1 STELLA App Set Up

This implementation runs with Pyterrier, where two different kinds of IR algorithms are used interchangeably in order to gain further insight into how each IR system compares to one another in terms of user satisfaction: The **BASE IR** system and the **EXP IR** system.

Upon receiving a request in the form of:

```
localhost:8080/stella/api/v1/ranking?query=corona&rpp=10
```

STELLA will return a list of 10 search results for the query 'corona' in the JSON format in this specific example, with the search result being interleaved between the BASE IR system and the EXP IR system.

6.2 Base Application Set Up

In order to emulate a proper IIR environment, the base application is designed after a proper search environment, where a user is invited to enter their query in either the normal or the advanced search box and then gets to satisfy their information need by clicking on a relevant search result in the SERP (see Figure 5).

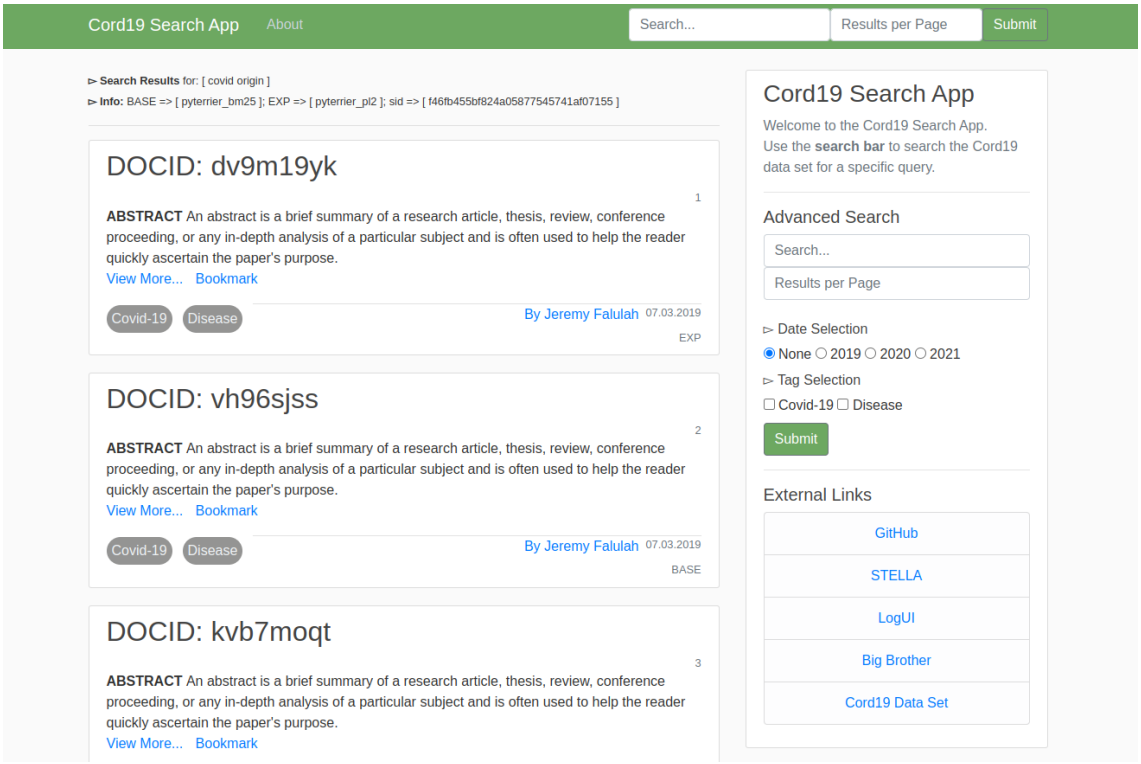

Figure 5 shows the regular search environment consisting out of an SERP and multiple kinds of search bars.

While the presentation of this website serves its purpose as a user interface with proper search functionality, the important part of this implementation lies in the underlying DOM model. The following illustration highlights all the important elements, that are later considered for logging by LogUI (see Figure 6):

Logging interactions on these specific elements is done in order to gain insight in user behavior regarding the usage of the different search bars, the queries and options selected as well as any kind of user behavior on the SERP elements, indicating a possible satisfied information need on specific ranks from a specific IR system (see Figure 6).

6.3 LogUI Setup

In order to facilitate the logging process, the LogUI client is integrated into the Flask app under the following procedure via the configuration object[27]:

1. The user enters the website and the page starts to load
2. The DOM fully loads
3. LogUI check: if not initialized, start LogUI Client and initialize handshake with LogUI Server

[27] https://github.com/AH-Tran/STELLA_LogUI/blob/main/search-app/static/logui_config.js

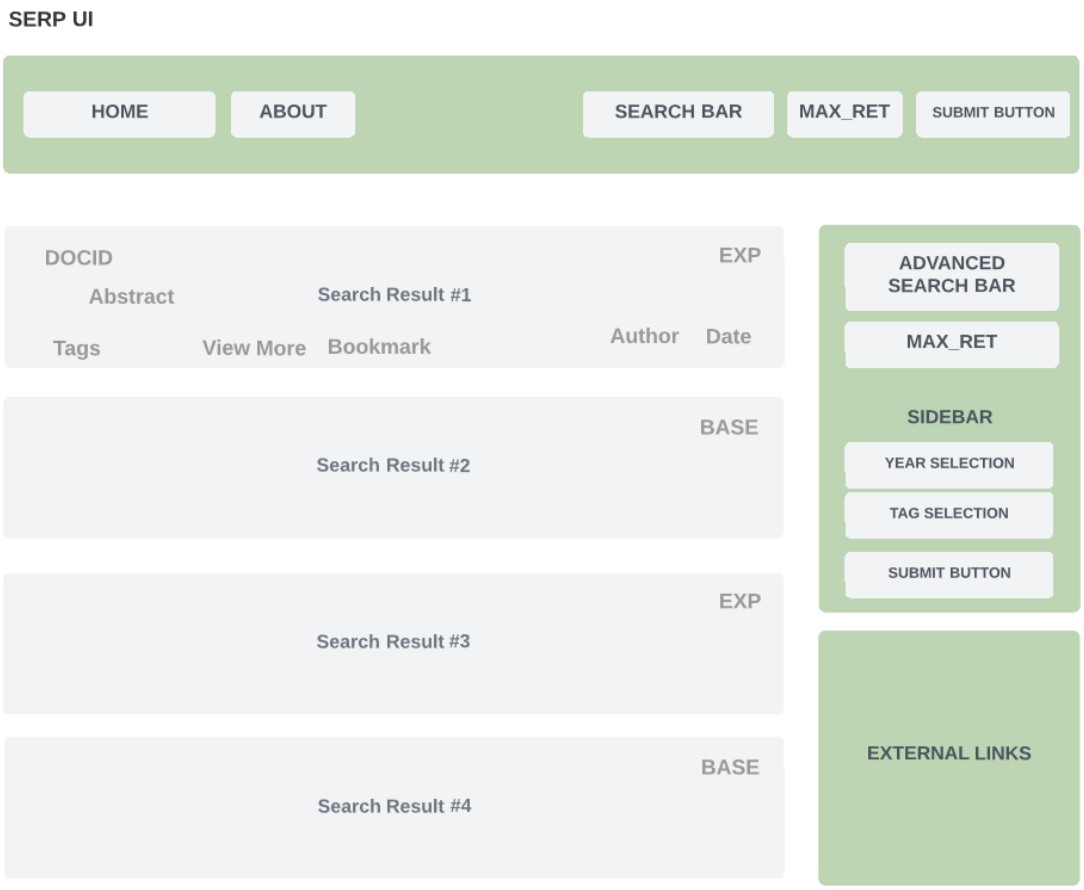

Figure 6 shows the underlying elements considered important for logging.

4. Upon successful handshake: The logging starts now
5. Upon exit or tab out: exit LogUI client
6. Session ID remains intact until user exits the website completely

Once the successful handshake has been completed, the client automatically picks up any user activity as outlined in the configuration object. In addition to regular interaction aspects such as mouse movement and time, this implementation of a configuration object observes and logs three kinds of elements with increased specificity:

1. The regular Search Bar: Where any kind of submission will lead to the **sent query** and the specified **amount of results**(MAX_RET) being actively logged (see Figure 7).

2. The advanced Search Bar: Where any kind of submission will lead to the **sent query**, the specified **amount of results**, the selected **year** and the selected **tags** will get logged (see Appendix A.7.).

3. SERP activity: Where a click on results from the SERP will lead to multiple things being logged: The **rank** of the search result and which **IR_TYPE** the search result belongs

```
'normal-query-submission': { // NORMAL SEARCH BAR
    selector: '#search-bar',
    event: 'formSubmission',
    name: 'NORMAL_QUERY_SUBMITTED',
    properties: {
        includeValues: [
            {
                nameForLog: 'completeQuery', // QUERY INPUT
                sourcer: 'elementProperty',
                selector: '#search-box',
                lookFor: 'value',
            },
            {
                nameForLog: 'rppValue', // MAX_RET INPUT
                sourcer: 'elementProperty',
                selector: '#rpp-box',
                lookFor: 'value',
            }
        ]
    }
},
```

Figure 7 shows the relevant JavaScript snippet for logging the search bar, capturing both the query and the rpp-value.

to as special metadata (see Appendix A.8.). Additionally, any additional clicks such as a click on the title, the 'view more' button, the 'bookmark' button or the tags will also get remarked with a special label within the log.

Logging these three main elements specifically allows the researcher to observe the user's journey and behavior on multiple levels: how the user acts when they are actively searching for something and how the user acts when evaluating the retrieved search results. By forcing LogUI to specifically log, which ranks and which experimental IR types get which clicks, a deeper insight into the performance of each IR system within STELLA can be gained and comparisons drawn.

6.4 Big Brother Set Up

Similarly, Big Brother is also implemented by installing the Big Brother GO server and dropping the Big Brother client into the Flask App. As designed, Big Brother works with no specific configuration necessary and thus should serve as an interesting alternative to LogUI's more elaborate logging process.

7 Experiments

This section will briefly inspect the logging data, which were retrieved from the aforementioned logging frameworks. These were created from manual sessions of user activity in order to ensure that the logging frameworks log the events just as intended. Additionally, it would also be insightful to elaborate on the logging data's structure and point out how it relates to the implementation specified in section 6.

7.1 Inspecting Activity on SERP Elements

The following snippet comes from the LogUI logging data, and displays how activity on SERP elements is logged with LogUI and represented in the resulting logging data file (see Figure 8).

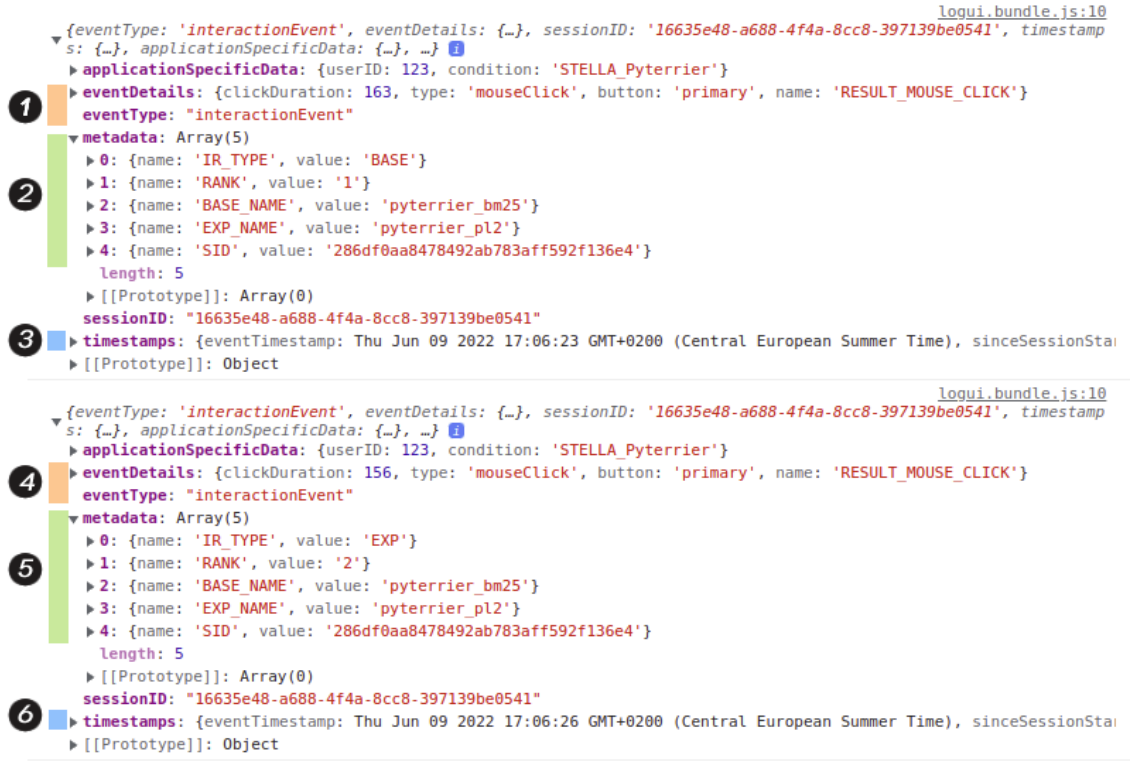

Figure 8 shows the user's activity on the first and second ranked SERP elements.

The logging data shows three points of interest per single interaction event:

1. **The Event Details** describing what specific type of event has been triggered (mouse click) and with which button.
2. **The Metadata Section** consists of extra data sourced from specified elements (see Appendix A.8.).

3. **The Timestamp** giving the logged interaction data a temporal context

With the help of LogUI's metadata functions, it is possible to enrich even simple mouse click interactions with more information, giving the mouse click that much more context and relevance for any potential post-hoc analysis. In this specific example, this interaction event does not only reveal which rank of the SERP has been clicked, but also which IR system the specific search result belongs to and which IR algorithm lies behind which IR system.

7.2 Inspecting Activity on the advanced Search Bar

The next logging data snippet comes from the LogUI logging data, which displays all additional extra information that have been submitted alongside the original query (see Appendix A.7.).


```
{ "eventType": "interactionEvent",
    "eventDetails": {
        "type": "submit",
        "name": "ADVANCED_QUERY_SUBMITTED",
        "submissionValues": [{
            "name": "completeQueryAdvanced",
            "value": "covid origin"
        }, {
            "name": "rppValueAdvanced",
            "value": "3"
        }, {
            "name": "yearValue",
            "value": "2021"
        }, {
            "name": "tagValue",
            "value": "covid-19"
        }]
}}
```


Figure 9 shows user's submission on the advanced search form.

Just by inspecting this log, researchers can observe that the user used the following information alongside their query:

- **The query:** covid origin
- **The rpp-value:** 3
- **The year-radio button:** 2021
- **The tag:** covid-19

This additional info allows for a deeper, fine-grained insight into the forms that a user can submit using the advanced search bar.

8 Evaluation

This section aims to evaluate the key features of LogUI and Big Brother (see Section 5) by comparing them with the needs of IIR, living lab, HCI (see Section 3) and the requirements of Data Logging (see Section 4.1) in order to ultimately assess how well suited each of the logging frameworks are for a living lab IIR use case such as STELLA.

These needs and requirements are summarized in the following table for the purpose of providing a good overview (see Table 1).

	Requirement/Need	Description
IIR	Control	Control over all possible variables within an environment
	Observability	Ability to observe any important key event
	Repeatability	Reproducibility of any achieved result
Living Lab	Experimental Approaches	Support for deploying multiple experimental systems at once
	Participation & User Involvement	Ability to involve a large number of participants
	Collaboration and Co-production of Knowledge	Ease of collaboration between multiple researchers and users
HCI	Satisfaction	Comfort and accessibility to use
Data Logging	Logging Data	Ability to clearly define key events (Custom Event Coding)
	Editing the Data Log	Ability to edit and correct logging data in case of errors
	Backing up the Data	Ability to savekeep data in case of errors or untimely accidents
	Analyzing Data	Ability to analyze the logging data; Ease of parsing for Insight and Context
Miscellaneous	Barrier of Entry	Ability to gain proficience without much effort
	Visibility	The visibility towards the end-users and participants
	Scalability	Ease of deployment of multiple systems at once
	Web Security	Grade and number of security checks

Table 1 summarizing the needs and requirements across disciplines and concepts for logging solutions.

8.1 Evaluation: LogUI

LogUI's key features lies in offering a complete, modular end-to-end logging solution, with a specialization in providing highly customizable custom event coding options for precise, relevant and fine-grained interaction logging and offering an experimental flight system, with which it is easy to deploy and log multiple experimental systems at the same time.

IIR Requirements Concerning the outlined requirements and needs of IIR environments, LogUI fulfills all three needs. Thanks to the highly customizable configuration object enforcing custom event coding practices, there is little that a researcher could not control nor observe on the DOM level with LogUI. The only thing that LogUI is missing is a higher level of control in the LogUI Control Application, as it is currently very bare bones in its available features, such as account management and LogUI server settings.

Living Lab Requirements As living lab environments highly value experimental approaches in real life context and encourage highly active participant and user involvement for the sake of collaboration and co-production of knowledge, LogUI's rich feature set for application, flight and session control would be very helpful in improving and encouraging experimental development spaces and A/B testing procedures. This is further improved by LogUI's containerized nature, allowing researchers to easily deploy more servers as the need for more experimental spaces and variations arises.

HCI Requirements When it comes to user satisfaction in the aspects of comfort and accessibility, it is LogUI's large set of customization features that sometimes gets in the way of completely satisfying the HCI requirement. A robust default setting with no configuration required would come a long way in easing up the barrier of entry for using LogUI. On the other hand, however, LogUI wins some points for offering a graphical user interface with the LogUI Control App, making the management of applications, flights, sessions and the downloading of logs more accessible to the common user.

Data Logging Requirements When comparing LogUI's feature set with the data logging requirements proposed by Philips and Duman, it is hard to see many shortcomings on LogUI's side(Philips and Dumas 1990). Concerning the aspect of logging data, LogUI's configuration object and its flexible custom event coding more than fulfills the criteria of being able to clearly define key events. By saving any log data in the JSON file, LogUI also allows for an easier way of editing and analyzing potential data. Last but not least, Philip's Concerns over back-ups and lost data are also addressed by LogUI's efforts to use cache techniques in order to ensure that no data is lost in cases of accidents and internet instability.

Additionally, LogUI offers near invisible logging capabilities, allowing users to interact with the interface with no interruption or risk of skewing user behavior by making them actively aware they are being logged. It also provides a robust web security protocol by only allowing authenticated client systems to directly interact with the LogUI server.

8.2 Evaluation: Big Brother

In contrast to LogUI's key feature, Big Brother focuses on offering a powerful and simple logging solution right out of the gate at the cost of limited customization and custom event coding options.

IIR Requirements Moving on to requirements and needs of IIR environments, Big Brother does not offer very much on that front. As this logging service sacrifices customization in favor of a much lower barrier of entry, Big Brother is not very suitable for research environments which prioritize complete control over their tools.

Living Lab Requirements While Big Brother does not offer an innate flight and experimental environment support like LogUI does, it does offer a highly robust and error-proof high throughput server, which comes in handy for bigger experimental development spaces where collaboration between many researchers and participants is required in order to solve harder challenges.

HCI Requirements With Big Brother's easy implementation and default configuration, the HCI need for ease of access is easily satisfied. This is further emboldened by Big Brother's inclusion of additional tools such as bbheat, making it easy to start analyzing logging data right out of the gate. The only missing feature on Big Brother's side is the inclusion of a graphical user interface, as controlling the Big Brother server via command line is not accessible at all for the common end user.

Data Logging Requirements On the topic of data logging requirements, Big Brother satisfies most of them, except the need to be able to clearly define key events via custom event coding. It does, however, make post-hoc analysis of data much easier by including analysis tools right out of the gate.

Similarly to LogUI, Big Brother also operates invisible in the background, ensuring that user behavior remains undisturbed by the logging process.

9 Conclusion

After a thorough elaboration of the specific needs and requirements of living labs and IIR environments, it can be said that the original research question:

RQ1 Do currently new logging frameworks satisfy the requirements and needs of web-based interactive information retrieval systems within a living lab approach?

Can definitely be affirmed positively, as both LogUI and Big Brother offer considerable benefits for living lab IIR systems in need of a robust logging feature.

LogUI offers an elaborate, feature-rich server-client architecture that allows the researcher to define and pinpoint the necessary user interaction capture points down to the singular element within a web application's UI, while also allowing the researcher to label them accordingly for a better and easier post-hoc analysis after the test period. This is further enhanced by the addition of metadata sourcers, from which the researcher can enrich singular user interactions with attributes sourced from the web application. Additionally, LogUI also easily enables the execution of A/B testing procedures by allowing the researcher to divide the logs between different types of flights with different kinds of conditions, thereby splitting the consequent logs into the appropriate experimental type even before the post-hoc analysis. The containerized implementation of LogUI allows researchers to easily scale up and deploy more LogUI servers as it becomes necessary, while also providing a more stable development environment for eventual troubleshooting. All in all, LogUI satisfies many needs of living lab and IIR environments such as STELLA, promoting the concept of simultaneous live user experiments and offers many possibilities when it comes to deploying and testing environmental set-ups in a live web environment.

Big Brother, on the other hand, only allows a considerable smaller customization than LogUI and no real innate focus on the deployment of multiple paralleled experimental websites, but makes up for it by the ease of integration and the addition of inbuilt analysis features such as heatmap, allowing researchers to start logging and analyzing user interaction right away and with fewer barriers to overcome. It also features web browser recording and an encouraged integration into the ELK Stack, both features which are missing in LogUI.

10 Future Research

For future research, it would be of utmost interest, to put the experimental live service support from LogUI to a true test by creating multiple different flight variations of the same IIR system, thus using its logging capability to its fullest.

Additionally, it would also be interesting to pursue additional dash boarding tools that would automatically receive and visualize logging data from LogUI or Big Brother and thus create a live analytics dashboard from which researchers could inspect and observe

user behavior in real time. There already exists multiple possible options such as Apache Spark[28] or Kibana[29] from the ELK Stack, making this a question whether a proper integration between these components would be feasible. In that regard, Big Brother already encourages and supports integration into the ELK Stack, making this an interesting direction and topic for future research.

11 Thanks and Acknowledgements

Many thanks to Timo Breuer and the Information Retrieval Research Group[30] for developing and providing the STELLA Infrastructure for this thesis, with which it was possible to integrate logging frameworks in a readily available living lab environment.

Another thanks to David Maxwell for developing and providing the LogUI framework, but also the very detailed LogUI documentation, with which it became easier to integrate LogUI into the STELLA framework and troubleshoot problems in case of unforeseen errors.

And a final thanks to Harry Scells, Jimmy and Guido Zuccon for providing the Big Brother service, that I could use as a comparison with the LogUI framework when it comes to modern, contemporary logging solutions with a decidedly different focus.

[28] https://spark.apache.org/
[29] https://www.elastic.co/kibana/
[30] https://ir.web.th-koeln.de/index.html

12 References

Philips, Brian H. and Joseph S. Dumas (Oct. 1, 1990). “Usability Testing: Identifying Functional Requirements for Data Logging Software”. In: *Proceedings of the Human Factors Society Annual Meeting* 34.4. Publisher: SAGE Publications, pp. 295–299. ISSN: 0163-5182. DOI: 10.1177/154193129003400412. URL: https://doi.org/10.1177/154193129003400412 (visited on 06/09/2022).

Robertson, S. E. and M. M. Hancock-Beaulieu (Mar. 1, 1992). “On the evaluation of IR systems”. In: *Information Processing and Management: an International Journal* 28.4, pp. 457–466. ISSN: 0306-4573. DOI: 10.1016/0306-4573(92)90004-J. URL: https://doi.org/10.1016/0306-4573(92)90004-J (visited on 06/08/2022).

Berners-Lee, Tim et al. (Aug. 1, 1994). “The World-Wide Web”. In: *Communications of the ACM* 37.8, pp. 76–82. ISSN: 0001-0782. DOI: 10.1145/179606.179671. URL: https://doi.org/10.1145/179606.179671 (visited on 06/09/2022).

HOIEM, DEREK E. and KENT D. SULLIVAN (Jan. 1, 1994). “Designing and using integrated data collection and analysis tools: challenges and considerations”. In: *Behaviour & Information Technology* 13.1. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01449299408914595, pp. 160–170. ISSN: 0144-929X. DOI: 10.1080/01449299408914595. URL: https://doi.org/10.1080/01449299408914595 (visited on 06/09/2022).

Shneiderman, Ben (1997). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 3rd. USA: Addison-Wesley Longman Publishing Co., Inc. 639 pp. ISBN: 978-0-201-69497-0.

Trewin, Shari (1998). “InputLogger: General-purpose logging of keyboard and mouse events on an Apple Macintosh”. In: *Behavior Research Methods, Instruments, & Computers* 30.2, pp. 327–331. ISSN: 0743-3808. URL: https://www.academia.edu/12249596/InputLogger_General_purpose_logging_of_keyboard_and_mouse_events_on_an_Apple_Macintosh (visited on 06/09/2022).

Borlund, Pia (Aug. 1, 2003a). “The concept of relevance in IR”. In: *JASIST* 54, pp. 913–925. DOI: 10.1002/asi.10286.

— (2003b). *The IIR evaluation model: a framework for evaluation of interactive information retrieval systems*. ISSN: 1368-1613 Publisher: Professor T.D. Wilson. URL: http://informationr.net/ir/8-3/paper152.html (visited on 06/08/2022).

Sandom, Carl and Roger S Harvey (2004). “Human Factors for Engineers”. In: *Human Factors*, p. 389.

Belkin, Nicholas (June 1, 2008). “ECIR KEYNOTE Some(what) Grand Challenges for Information Retrieval 1”. In: SIGIR Forum. Vol. 42, pp. 47–54. ISBN: 978-3-540-78645-0. DOI: 10.1007/978-3-540-78646-7_1.

Shah, Inayatullah, L. Toaimy, and M. Jawed (Dec. 31, 2008). “RWELS: A Remote Web Event Logging System”. In: *Journal of King Saud University - Computer and Information Sciences* 20, pp. 1–11. DOI: 10.1016/S1319-1578(08)80001-8.

Ter Louw, Mike, Jin Soon Lim, and V. N. Venkatakrishnan (Aug. 2008). “Enhancing web browser security against malware extensions”. In: *Journal in Computer Virology* 4.3, pp. 179–195. ISSN: 1772-9890, 1772-9904. DOI: 10.1007/s11416-007-0078-5. URL: http://link.springer.com/10.1007/s11416-007-0078-5 (visited on 06/09/2022).

Zimmerman, Patrick H. et al. (Aug. 2009). “The Observer XT: a tool for the integration and synchronization of multimodal signals”. In: *Behavior Research Methods* 41.3, pp. 731–735. ISSN: 1554-351X. DOI: 10.3758/BRM.41.3.731.

Azzopardi, Leif and Krisztian Balog (2011). “Towards a Living Lab for Information Retrieval Research and Development”. In: *Multilingual and Multimodal Information Access Evaluation*. Ed. by Pamela Forner et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 26–37. ISBN: 978-3-642-23708-9. DOI: 10.1007/978-3-642-23708-9_5.

Fette, Ian and A. Melnikov (2011). “The WebSocket Protocol”. In: *RFC*. DOI: 10.17487/rfc6455.

Feild, Henry A. and James Allan (Nov. 1, 2013). “Using CrowdLogger for in situ information retrieval system evaluation”. In: *Proceedings of the 2013 workshop on Living labs for information retrieval evaluation*. LivingLab ’13. New York, NY, USA: Association for Computing Machinery, pp. 15–18. ISBN: 978-1-4503-2420-5. DOI: 10.1145/2513150.2513164. URL: https://doi.org/10.1145/2513150.2513164 (visited on 06/09/2022).

Larson, David, Jigang Liu, and Yanjun Zuo (June 2014). “Performance analysis of javascript injection detection techniques”. In: *IEEE International Conference on Electro/Information Technology*. IEEE International Conference on Electro/Information Technology. ISSN: 2154-0373, pp. 140–148. DOI: 10.1109/EIT.2014.6871752.

Schuth, Anne, Krisztian Balog, and Liadh Kelly (2015). “Extended Overview of the Living Labs for Information Retrieval Evaluation (LL4IR) CLEF Lab 2015”. In: p. 22.

Yu, Xiaoming (2017). “ICTNET at TREC2017 OpenSearch Track”. In: p. 3.

Schaer, Philipp, Johann Schaible, and Bernd Müller (Apr. 14, 2020). “Living Labs for Academic Search at CLEF 2020”. In: *Advances in Information Retrieval: 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020,*

Proceedings, Part II. Berlin, Heidelberg: Springer-Verlag, pp. 580–586. ISBN: 978-3-030-45441-8. DOI: 10.1007/978-3-030-45442-5_75. URL: https://doi.org/10.1007/978-3-030-45442-5_75 (visited on 06/08/2022).

Breuer, Timo and Philipp Schaer (2021). "A Living Lab Architecture for Reproducible Shared Task Experimentation". In: *Emerging Technologies*, p. 15.

Maxwell, David and Claudia Hauff (2021). "Developing Contemporary Web-Based Interaction Logging Infrastructure: The Design and Challenges of LogUI". In: p. 11.

Scells, Harrisen, Jimmy, and Guido Zuccon (July 11, 2021). "*Big Brother:* A Drop-In Website Interaction Logging Service". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. Virtual Event Canada: ACM, pp. 2590–2594. ISBN: 978-1-4503-8037-9. DOI: 10.1145/3404835.3462781. URL: https://dl.acm.org/doi/10.1145/3404835.3462781 (visited on 06/07/2022).

FISSAC (2022). *Living Labs | FISSAC*. URL: https://fissacproject.eu/en/living-labs/ (visited on 06/08/2022).

Appendix

A.1. LogUI Bundle and Configuration File Location

```
...
</body>
<!-- LogUI Script Bundle -->
<script src="./static/js/logui.bundle.js"></script>
<!-- LogUI Configuration Object -->
<script src="../static/logui_config.js"></script>
</html>
```

A.2. Log Data in JSON format[31]

```
[
  // First Event
  {
    "eventType": "statusEvent",
    "eventDetails": {
      "type": "started",
      "browserAgentString": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6)
          AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90
          Safari/537.36",
      "screenResolution": {
        "width": 1920,
        "height": 1080,
        "depth": 24
      },
      "viewportResolution": {
        "width": 1920,
        "height": 889
      }
    },
    "timestamps": {
      "eventTimestamp": "2021-04-24T20:19:15.541Z",
      "sinceSessionStartMillis": 27,
      "sinceLogUILoadMillis": 27
    },
    "applicationSpecificData": {
```

[31] https://github.com/logui-framework/server/wiki/Parsing-Logs

```
      "userId": "user185",
      "groupId": "60847d43ae062cf467fdb9fe",
      "variant": "mid"
    },
    "applicationID": "e386d221-1b45-481d-bd3a-ef9d2679e60b",
    "flightID": "4246f547-e229-4f58-a5b1-d8ad7c3e0e42",
    "sessionID": "e72deba6-bf54-4311-884a-305273c734d1"
  },
]
```

A.3. JavaScript Configuration Object as an Example

```
let configurationObject = {
                logUIConfiguration: {
                endpoint: 'ws://localhost:8000/ws/endpoint/',
                authorisationToken: 'Exampletoken',
                verbose: true,
            },
            applicationSpecificData: {
                userID: USER_ID,
                condition: 'STELLA_Pyterrier',
            },
            trackingConfiguration: {
                // SEARCH BAR CLICKS
                'searchbar-click': {
                    selector: '#search-bar', // Selector: id(#), class(.)
                    event: 'mouseClick',// Event (clicks, forms, etc)
                    properties: {
                        primary: {
                            name: 'SEARCH_PRIMARY_MOUSE_CLICK',
                        },
                        secondary: {
                            name: 'SEARCH_SECONDARY_MOUSE_CLICK',
                        },
                        auxiliary: {
                            name: 'SEARCH_AUX_MOUSE_CLICK',
                        },
                    },
                },
            }
```

```
        }
```

A.4. Big Brother Client Integration

```
< script src = " bigbro . js " ></ script >
< script type = " text / javascript " >
BigBro . init ( session_id , " localhost :8080 ");
</ script >
```

A.5. Big Brother Client Integration with Configuration

```
BigBro . init ( session_id ,
" localhost :8080 " ,
[ " mousemove " , " onload " ]) ;
```

A.6. Big Brother Client Integration with Screen Capture

```
let bb = BigBro . init ( session_id ,
" localhost :8080 " ) ;
bb . startCaptureOnClick (" capture ") ;
```

A.7. JavaScript Snippet for Logging the advanced Search Bar

```
'advanced-query-submission': { // ADVANCED SEARCH BAR
    selector: '#search-bar-advanced',
    event: 'formSubmission',
    name: 'ADVANCED_QUERY_SUBMITTED',
    properties: {
        includeValues: [
            {
                nameForLog: 'completeQueryAdvanced', //QUERY INPUT
                sourcer: 'elementProperty',
                selector: '#search-box-advanced',
                lookFor: 'value',
            },
            {
                nameForLog: 'rppValueAdvanced', //RPP INPUT
                sourcer: 'elementProperty',
                selector: '#rpp-box-advanced',
                lookFor: 'value',
            },
```

```
            {
                nameForLog: 'yearValue', //RADIO YEAR INPUT
                sourcer: 'elementAttribute',
                selector: '#searchbar-advanced-radio-id input:checked',
                lookFor: 'value',
            },
            {
                nameForLog: 'tagValue', //CHECKBOX TAG INPUT
                sourcer: 'elementAttribute',
                selector: '#searchbar-advanced-checkbox-id input:checked',
                lookFor: 'value',
            },
        ]
    }
},
```

A.8. JavaScript Snippet for Logging SERP rank and associated Metadata

```
'result1-click': { // Mapping name (between element(s) and event)
    selector: '#result-1', // Selector: id(#), class(.)
    event: 'mouseClick',// Event (clicks, forms, etc)
    properties: {
        primary: {
            name: 'RESULT_MOUSE_CLICK',
        },
        secondary: {
            name: 'RESULT_MOUSE_CLICK',
        },
        auxiliary: {
            name: 'RESULT_MOUSE_CLICK',
        },
    },
    metadata: [
        {
            nameForLog: 'IR_TYPE',
            sourcer: 'elementAttribute',
            lookFor: 'ir_type',
        },
        {
            nameForLog: 'RANK',
```

```
            sourcer: 'elementAttribute',
            lookFor: 'result_rank',
        },
        {
            nameForLog: 'BASE_NAME',
            sourcer: 'elementAttribute',
            lookFor: 'base_ir',
        },
        {
            nameForLog: 'EXP_NAME',
            sourcer: 'elementAttribute',
            lookFor: 'exp_ir',
        },
        {
            nameForLog: 'SID',
            sourcer: 'elementAttribute',
            lookFor: 'sid',
        },
    ],
},
```