DOI: 10.5281/zenodo.5702657

Triple

Transforming Research through Innovative Practices for Linked Interdisciplinary Exploration

[30 SEPTEMBER 2021]

Advancing Open Scholarship

D5.6 – REPORT ON THE DISCOVERY SYSTEM

Version 1.0 – Final
PUBLIC

H2020-INFRAEOSC-2019
Grant Agreement 863420

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 863420

Report on the Discovery System

Project Acronym:	**TRIPLE**
Project Name:	**Transforming Research through Innovative Practices for Linked Interdisciplinary Exploration**
Grant Agreement No:	**863420**
Start Date:	**1/10/2019**
End Date:	**31/03/2023**
Contributing WP	**5**
WP Leader:	**Net7**
Deliverable identifier	**D5.6**
Contractual Delivery Date: 09/2021	**Actual Delivery Date: 09/2021**
Nature: Other	**Version: 1.0 Final**
Dissemination level	**PU**

Revision History

Version	Created/Modifier	Comments
0.1	Peter Kraker (OKMAPS), Christopher Kittel (OKMAPS), Maxi Schramm (OKMAPS), Jan Konstant (OKMAPS)	First draft
0.2	Yoann Moranville (DARIAH), Luca De Santis (Net7)	Internal review
0.3	Peter Kraker (OKMAPS), Christopher Kittel (OKMAPS)	Revisions
1.0	Peter Kraker (OKMAPS)	Final version

Table of Contents

Table of Figures

Acronyms

API	Application Programming Interface
BASE	Bielefeld Academic Search Engine
EOSC	European Open Science Cloud
ETL	Extract, Transform, Load
HTML	Hypertext Markup Language
ID	Identifier
JSON	JavaScript Object Notation
LAMP	Linux, Apache, MySQL, PHP
JS	JavaScript
MVP	Minimum Viable Product
PMC	PubMed Central
RDBMS	Relational Database Management System
REST	Representational State Transfer
SSH	Social Sciences and the Humanities
UX	User Experience
VIPER	The Visual Project Explorer
WP	Work Package

Publishable Summary

In this deliverable, we present an innovative discovery system that brings together textual and visual interface components. This discovery system is based on Head Start, an open source framework that has been developed by Open Knowledge Maps over more than six years. Head Start provides interactive, web-based overviews of research topics, authors, and projects. The overviews are based related resources, for example, the most relevant publications on a topic, or the articles and datasets published by an author or project.

Head Start is one of the innovative services that is integrated with GoTriple in order to support discovery. In comparison to a pure list-based approach, Head Start provides a bird's eye view of the contents by way of visualisation. Head Start highlights connections between research outputs using clustering. This helps users to sort the relevant from the irrelevant with respect to their research question or information need. Head Start also supports users to come to know the concepts related to a topic, and it enables the easy identification of open access content.

These features are very relevant in the SSH domain, as supported by the user research carried out in this project. Among the issues that Head Start alleviates is getting and keeping an overview of a field and obtaining a better understanding of the terminology related to a topic. Head Start gives users a starting point, especially those who are not yet experts in a field. In this way, TRIPLE can also attract new audiences beyond researchers, including students, practitioners, journalists, and citizens. Finally, using some of the machine learning algorithms included in Head Start, such as on-the-fly deduplication and missing subject inference, GoTriple can provide an engaging search experience even if the metadata is messy or incomplete.

With Head Start, GoTriple can provide a popular discovery option that has become a mainstay for users from around the world. Since 2016, a million users have used the visual search on openknowledgemaps.org and more than 400,000 knowledge maps have been created so far.

Head Start is a tightly integrated innovative service in GoTriple. The Head Start interfaces can be accessed by users as part of the core platform. The integration is modeled after popular web search engines that link to additional services and interfaces directly within their search results page of GoTriple. On the backend, Head Start has a direct connection to the ElasticSearch index of GoTriple, where GoTriple's data is stored.

In this deliverable, we discuss Head Start, its architecture as well as its integration in GoTriple in detail. Furthermore, we describe the work done in the associated task T5.6, which includes a refactoring of the Head Start backend, the development of a standalone search flow package, the integration of Head Start into GoTriple, adapting the software to the GoTriple data, and improvements and adaptations to the machine learning pipeline.

1| DESCRIPTION OF THE SERVICE

In this deliverable, we present an innovative discovery system that brings together textual and visual interface components. This discovery system is based on Head Start, an open source framework that has been developed by Open Knowledge Maps over more than six years (Kraker et al. 2019). Head Start provides interactive, web-based overviews of research topics, authors, and projects. The overviews are based related resources, for example, the most relevant publications on a topic, or the articles and datasets published by an author or project.

Figure 1 shows an example of such an overview for the topic of digital education. The title at the top indicates what the current overview is about, complete with a context line of the parameters that determined the selection of resources. These include the sorting criteria (e.g. most relevant or most recent), the time range, resource types and data source.

The visualisation on the left-hand side gives an overview of the resources. Currently, Head Start offers a knowledge map (pictured in Figure 1), which provides a topical overview of the resources, and a streamgraph, which gives a temporal overview of the resources (pictured in Figure 2). In the future, further visualisation types might be added, such as a geo map for a geographical overview. Users can interact with the interface by clicking on elements of the visualisation to, for example, see the selection of resources related to a certain concept in the visualisation.

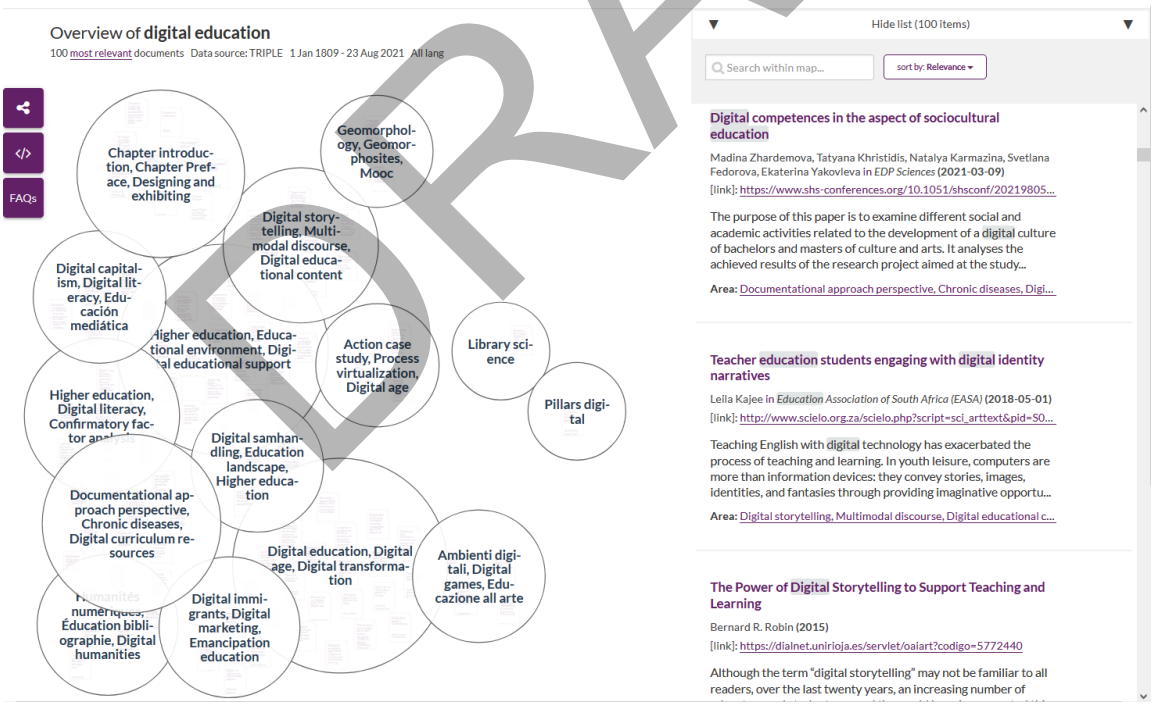

FIGURE 1. EXAMPLE OF A KNOWLEDGE MAP ON THE TERM "DIGITAL EDUCATION"

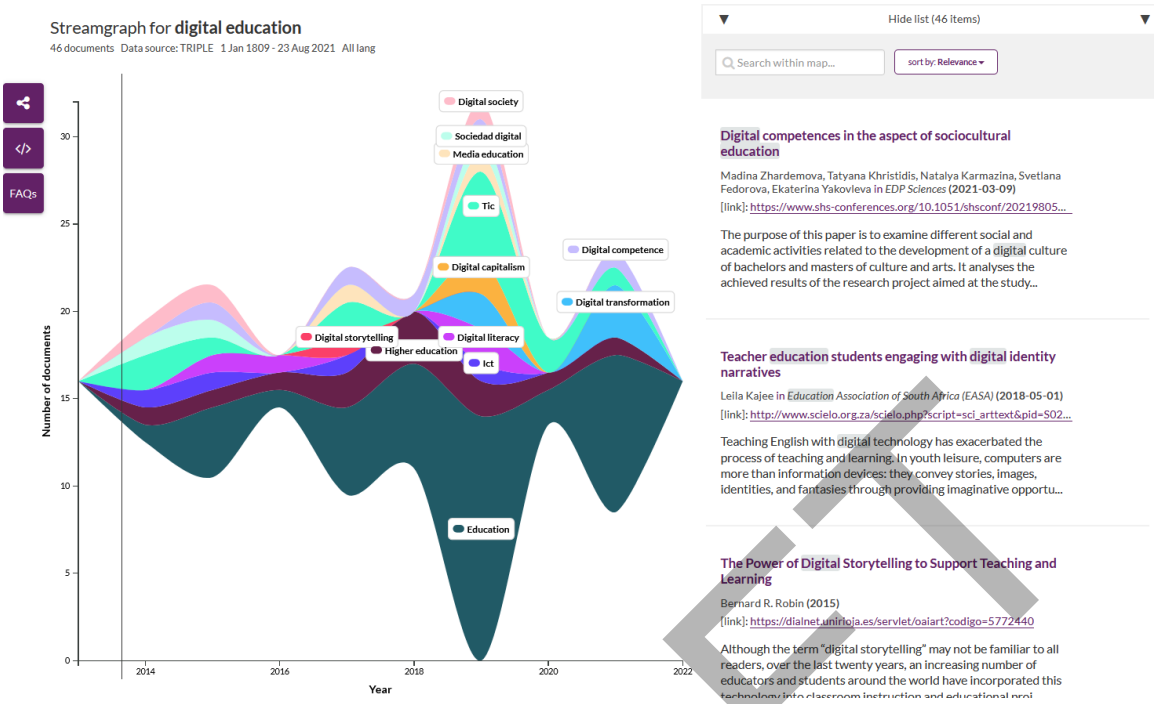

FIGURE 2. EXAMPLE OF A STREAMGRAPH ON THE TERM "DIGITAL EDUCATION"

The list on the right-hand side includes the resources included in the visualisation in sequential order. The list can be sorted by different characteristics, such as relevance, date or title. Furthermore, users are able to filter outputs according to a free text of their choosing, or by selecting a dedicated filter for open access resources. Head Start also offers a number of complementary features via the toolbar on the left-hand side. The button at the top leads to a popup that shows an embed code, for reuse of the map on websites and in dashboards. The button in the middle opens a set of sharing options, while the button on the bottom links to the FAQs for additional information on the interface.

Title, visualisation and list are connected and reflect how the user interacts with the interface. If, for example, the user clicks on a certain area in the knowledge map (see Figure 3), the visualisation zooms into that area. The title is updated to reflect that the user is now looking at a specific area within the visualisation, and the list is filtered to only show the resources related to that area. Another example is that the filters applied in the list are also reflected in the visualisation, meaning that the visualisation shows only those resources that meet the current filter criteria.

Head Start comes with a backend that is capable of automatically creating the input for the user interfaces. Each visualisation starts from a seed - that is either a search term, a project id or an author id. This seed is then sent to a data endpoint. For TRIPLE, this is the GoTriple ElasticSearch index. Head Start also supports BASE and PubMed for topical searches, OpenAIRE for project overviews and LinkedCat+ for visual representations of an author's output. Most recently, we

have also added Google Sheets as a potential data source to enable visualisations based on collaborative collections of research outputs.

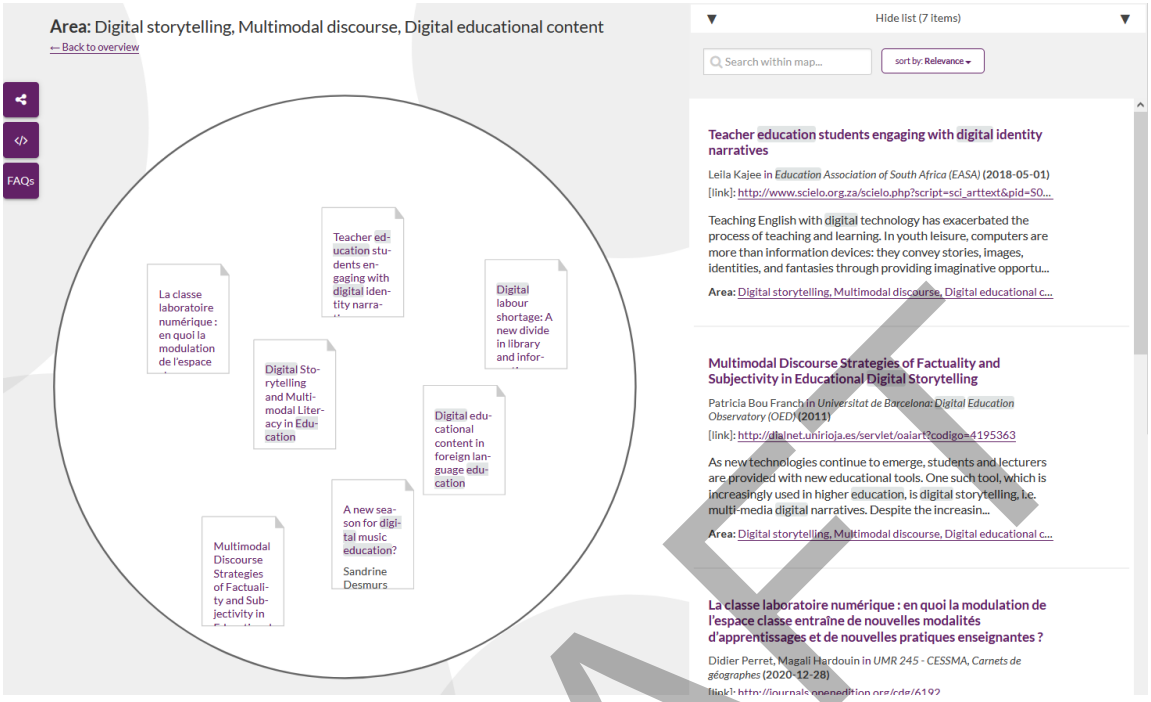

FIGURE 3. KNOWLEDGE MAP ON TERM "DIGITAL EDUCATION", ZOOMED INTO THE AREA "DIGITAL STORYTELLING, MULTIMODAL DISCOURSE, DIGITAL EDUCATIONAL CONTENT"

From each third-party service, we request the metadata for the most relevant resources related to the topic, author or project. Thereby, relevance is determined by the data provider. GoTriple, BASE and PubMed define relevance as the text similarity between your query and the article metadata. Depending on the use case and integration, certain thresholds may be enforced. For a topical overview of a search term in GoTriple, PubMed, or BASE for example, the 100 most relevant papers are used. This is primarily done to keep the cognitive load, i.e. the number of resources presented in each view and number of levels of resources included in the visualisation in total, at a manageable level. There are also technical limitations: most data providers either impose a limit on how many items can be retrieved or considerably slow down larger queries.

Nevertheless, we are investigating how to enable the exploration of larger amounts of content. At the moment, you can dig deeper into a topic by providing a more specific search query. For further considerations in that direction, please refer to chapter 6 of this deliverable.

The metadata is then cleaned, enriched, aggregated and summarized in real-time. This is done with a machine learning pipeline, which creates a representation that is used as input for the user interface. One possible enrichment at this stage is annotating the metadata with metrics data such as citations. The metrics data can then be used as additional property in the visualisation to e.g. scale the size of the bubbles in a knowledge map. We currently support CrossRef and PMC for citations and Altmetric for alternative metrics.

Head Start is used in a variety of projects and services. It powers Open Knowledge Maps, the world's largest visual search engine for research, which enables users to create knowledge maps of research topics in any discipline based on more than 270 million research outputs. Open Knowledge Maps enables a large number of interest groups to freely access scientific content. Users include researchers, students, librarians, educators, journalists and practitioners around the world. Head Start is also the basis for VIPER, The Visual Project Explorer, and CoVis, a platform to discover reliable COVID-19 research. Furthermore, Head Start is used in the LinkedCat+ platform of the Austrian Academy of Sciences, and to visualize the outcomes of the Patient and Public Involvement and Engagement projects of Ludwig Boltzmann Gesellschaft. In the past, Head Start has also been used in the H2020 project OpenUP and the Conference Navigator of University of Pittsburgh.

In this deliverable, we will describe how Head Start is integrated in GoTriple and what the benefits of this integration are. We also present the work done in task T5.6 and scenarios for future work inside and outside of TRIPLE. Note that we will focus on the backend of Head Start and the integration of textual and visual search components. As visualisations play an important role as an innovative aspect of TRIPLE overall, they have been given their own deliverable. The visualisations included in Head Start are discussed in more depth in another deliverable (D5.4 "Visualisations"), where you will also find an overview of other types of visualisations used in TRIPLE.

2| THE SERVICE IN TRIPLE

Head Start is one of the innovative services that is integrated with the GoTriple platform in order to support discovery. Within the TRIPLE project, the software package is adapted and extended to meet the specific needs and requirements of the SSH domain.

With Head Start, GoTriple can provide a popular discovery option that has become a mainstay for users from around the world. Since 2016, more than a million users have used the visual search on openknowledgemaps.org and more than 500,000 knowledge maps have been created so far.

Some of the advantages of using Head Start are summed up in Figure 4. In comparison to a pure list-based approach, Head Start provides a bird's eye view of the contents by way of visualisation. Head Start highlights connections between research outputs using clustering. For each cluster, pertinent keywords derived from the research outputs contained in this cluster are shown. This helps users to sort the relevant from the irrelevant with respect to their research question or information need. Head Start also supports users to come to know the concepts related to a topic, which is often the most difficult step in any discovery process. In this way, Head Start supports sensemaking, which is an important part of discovery that is often neglected by conventional approaches.

Head Start also enables the easy identification of relevant open access content by highlighting OA papers and books and by making it possible to preview them directly within the interface. This is especially useful for users in disadvantaged regions and for stakeholders outside of academia. Furthermore, Head Start makes it possible to stay on top of the current research with visualisations that are based on the most recent research on a topic.

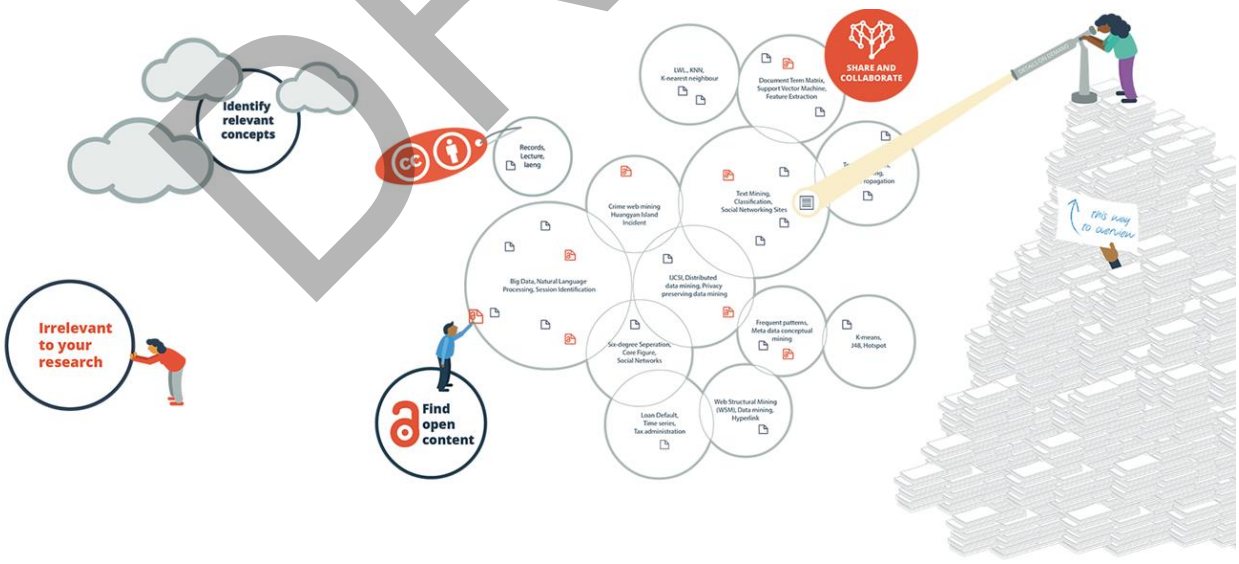

FIGURE 4. ADVANTAGES OF HEAD START FROM A USER PERSPECTIVE

These features are very relevant in the SSH domain, as supported by the user research carried out in WP3. In the workshop on visual discovery (see also Deliverable D3.4), participants noted that it is very challenging to find resources that are really relevant for their topic of interest. They also mentioned that it was difficult to stay up-to-date on relevant information for their projects. Participants also described that finding open access articles is an issue in general and getting access to (closed) sources is a problem in particular for freelance researchers who don't have access to university subscriptions. It was also expressed that the possibility on digital platforms to find ideas or get inspired when they don't know yet what they are looking for is missing.

Head Start enables GoTriple to offer a tried-and-tested search experience for its users that covers many use cases that a pure list-based search cannot support. As discussed above, this includes getting and keeping an overview of a field and obtaining a better understanding of the terminology related to a topic. This gives users a starting point, especially those who are not yet experts in a field. In this way, GoTriple can also attract new audiences beyond researchers, including students, practitioners, journalists, and citizens. With Head Start, GoTriple can provide these user groups with an intuitive, visual interface that has proven to be useful and usable across age and stakeholder groups - even schoolchildren can easily interact with the visualisations and derive information from them.

Finally, using some of the machine learning algorithms included in Head Start, such as on-the-fly deduplication and missing subject inference, GoTriple can provide an engaging search experience even if the metadata is messy or incomplete.

3| SOFTWARE ARCHITECTURE

This section gives an overview of the Head Start backend architecture. As discussed in section 1, the visualisation aspects of Head Start are discussed in a separate deliverable, D5.4, which is why the architecture of the frontend and the machine learning procedures to calculate the visualisations are also described there.

It should be noted that the Head Start backend is in a state of flux, as it is being rewritten within the TRIPLE project. As only a portion of the backend has been refactored at the time of writing, you will see references to both the new backend, called “modern backend” hereafter, and the legacy backend.

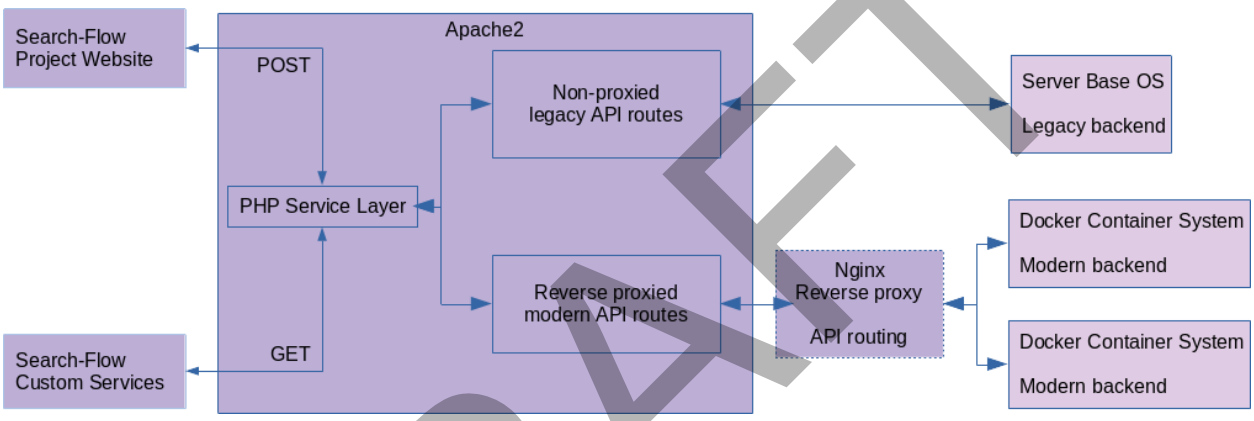

FIGURE 5. HIGH-LEVEL PERSPECTIVE OF THE HEAD START BACKEND

From a high-level perspective, the Head Start backend consists of three main components (see Figure 5):

a) The search flow
   The search flow is a modular, configurable component that provides a user interface and sends a user's search as a GET or POST request to an endpoint in the routing layer. Endpoints provide functionality for initiating a map creation for a specific search within a specific data integration, or retrieving data of newly created or historic visualisations from the database.
b) The routing/service layer
   The routing layer is based on PHP/Apache2 and handles requests. It contains some minimal logic in deciding to which backend-service a search request will be assigned, based on the endpoint to which the request was sent as well as the requests parameters. The routing layer currently also separates requests between the legacy version of the backend and the modern one. Requests to the legacy version are directly handed over to the backend, while requests to the modern API are proxied by the Apache2 webserver and passed over to an Nginx reverse proxy. The Nginx reverse proxy acts as a gateway

between the routing layer and the modern backend. It provides additional routing functionalities, for example between multiple versions of the backend.

c) The core backend services, of which currently exist a legacy version and a modern one. The legacy backend consists of R-Scripts which are directly executed on the host OS by the PHP service layer, as well as an SQLite3 database, which is also read/written directly by the PHP service layer. The modern backend follows an API-oriented microservice pattern and is running fully containerized in Docker containers.

As part of the backend refactoring, the architecture is currently in a transition from a tightly integrated client-server model to an API-based microservice model. In this transition period, both architectures are in use, depending on the type of the integration.

3.1 Description of legacy components

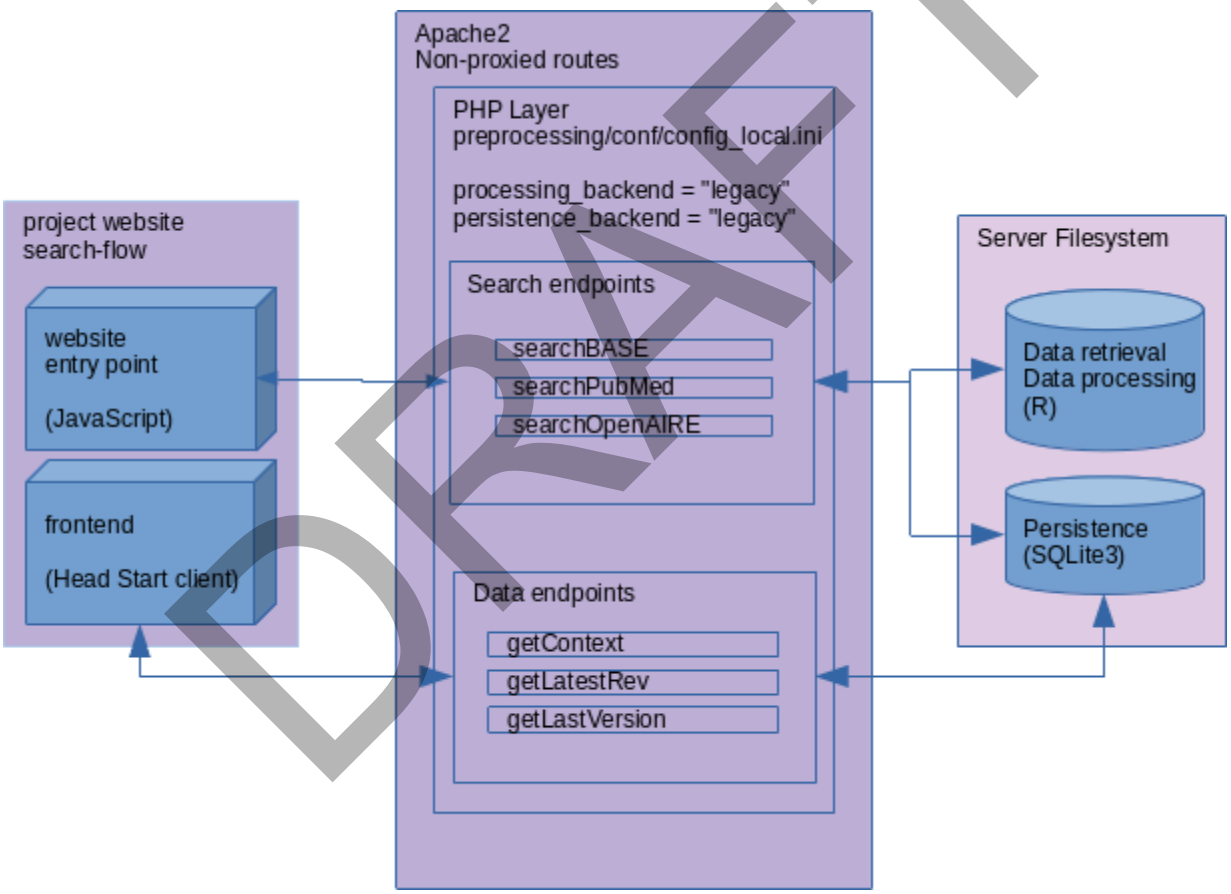

FIGURE 6. OVERVIEW OF THE LEGACY COMPONENTS OF THE HEAD START BACKEND

Figure 6 gives an architectural overview of the legacy components of the Head Start backend and their integration with the search flow and frontend. The search flow component (JavaScript, PHP, HTML/CSS) starts with a search box, where a user types in a search query and chooses

optional parameters like a date range or document type filters. Once a search is submitted, the query and parameters are sent as a POST request to the search endpoint corresponding to the data integration. The legacy version is currently maintained for the following data sources: BASE, PubMed and OpenAIRE.

The PHP service layer receives the search request and, should no cached version of the visualisation exist, executes a backend R subprocess. The R subprocess retrieves search results from the data source via data clients and produces visualisation data which it returns in JSON format to the service layer. The service layer will then persist the data on disk in a SQLite3 database under a unique identifier, and return the identifier to the search flow.

On the visualisation page, the Head Start client takes the identifier (which could also come from direct links), and requests the visualisation data from the data endpoints in the service layer. The data endpoints access the SQLite3 database and return the visualisation data in a JSON format.

3.2 Description of modern components

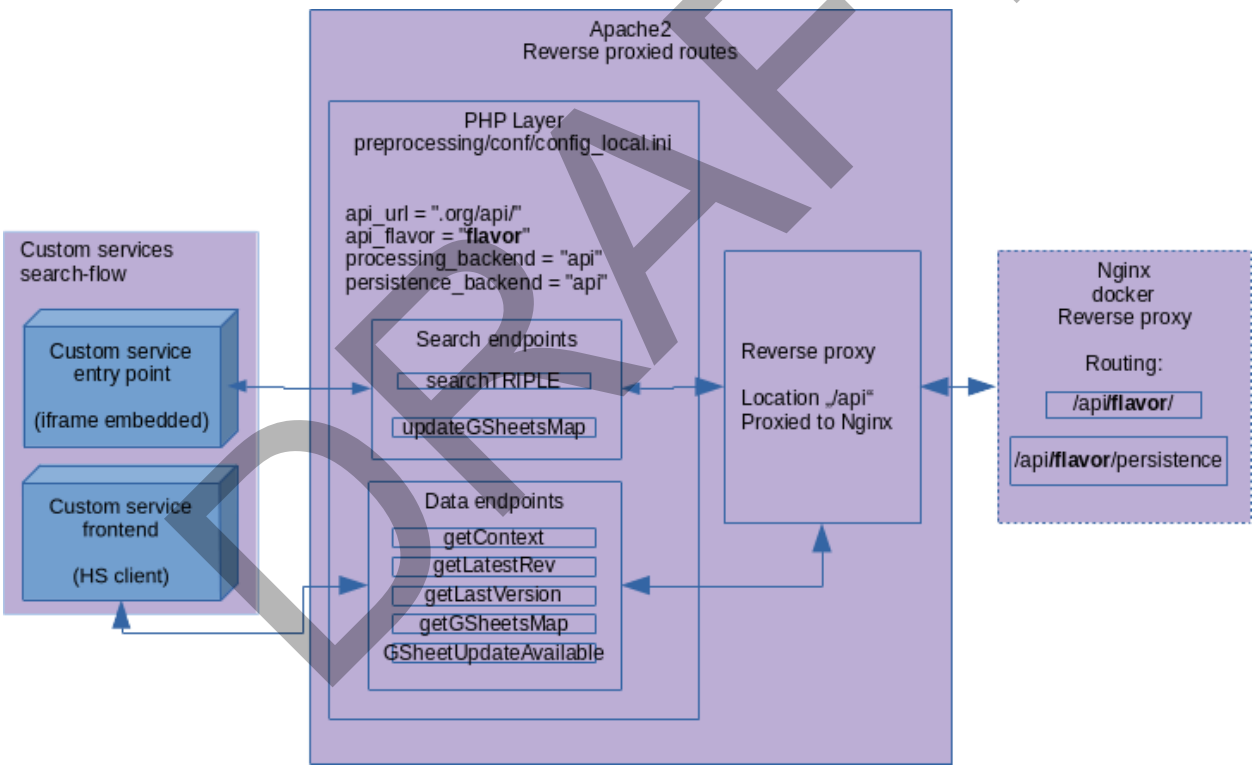

FIGURE 7. OVERVIEW OF THE MODERN COMPONENTS OF THE HEAD START BACKEND

Figure 7 gives an architectural overview of the modern components of the Head Start backend and their integration with the search flow and frontend. The modernized backend architecture adds new functionalities in each component to allow for a wider range of integration scenarios. In the search flow, a GET route can be activated, which can e.g. be reached when the search flow is embedded as an iframe. The search flow then proceeds to communicate with the service layer

and accesses the search endpoints. In the modern backend, two data integrations are maintained, one for TRIPLE and one for Google Sheets.

In contrast to the legacy backend, here the PHP service layer only functions as a thin intermediator in front of the actual API. The PHP service layer now takes the search parameters, and instead of executing subprocesses, it makes an API request to the modern backend. This API request is reverse proxied by Apache2 and handed over to an Nginx reverse proxy. The Nginx reverse proxy is already part of the containerized environment and the only gateway through which the backend infrastructure can be reached. Its purpose is to route API requests according to the version that was requested, as well as provide load-balancing capability should that become necessary in the future.

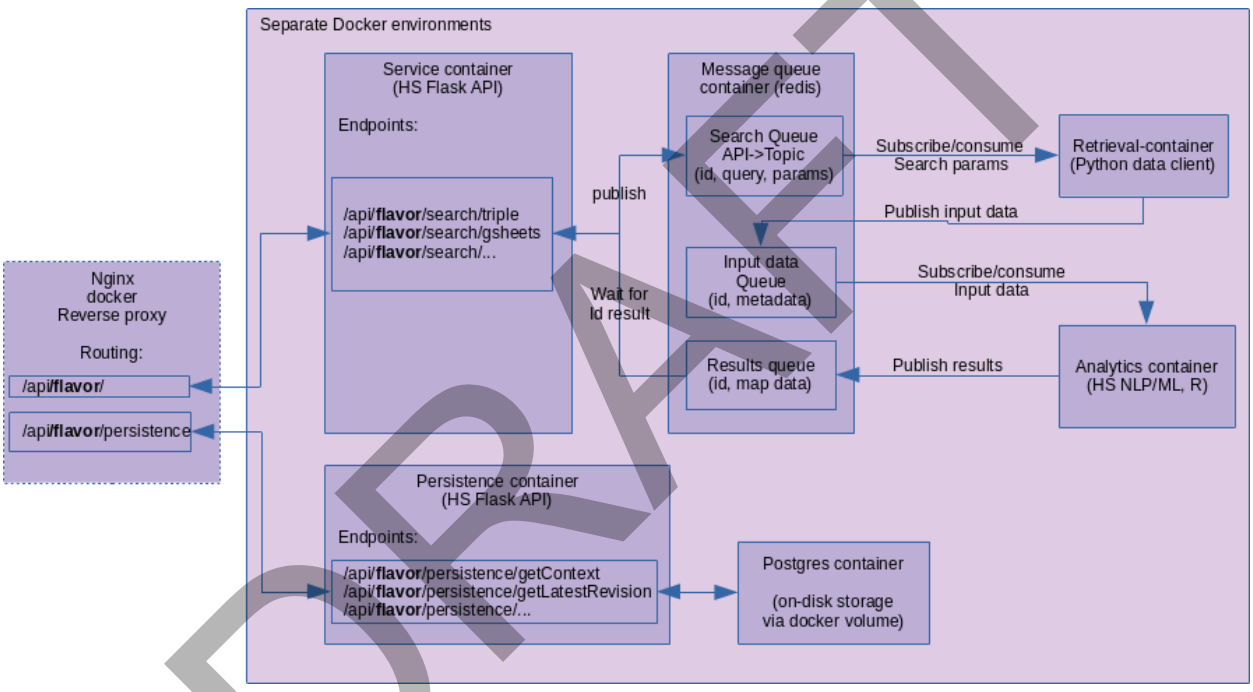

FIGURE 8. MICROSERVICE ARCHITECTURE OF THE MODERN BACKEND OF HEAD START AND THEIR INTEGRATION VIA A REVERSE PROXY SERVER

The Head Start modern backend is run fully encapsulated in a system of Docker containers (see Figure 8). All functionalities, which in the legacy backend are executed as PHP subprocesses, are instead wrapped in a Flask Python-application and RESTfully exposed. Two applications run separately, one for the data processing related endpoints, and one for the data persistence related endpoints, and the Nginx reverse proxy decides the correct routing of requests. Currently, the API endpoints correspond to the PHP service layer endpoints. In the course of the backend refactoring, we will introduce an (internal) API client that can directly interface with the

API endpoints. This API client will also be introduced in the service layer endpoints, which will then only serve as preconfigured wrappers around the API client.

Data retrieval and processing, which in the legacy backend is executed as one monolithic process, is in the modern backend split into separate microservices, each running in their own containerized environment. Microservices communicate via a message queue which allows for separate scalability. The message queue system is Redis, which is also running in a separate container. Data retrieval containers are running Python data clients, e.g. for the TRIPLE ElasticSearch index. The data analytics container is partially legacy R code in a Python wrapper, and partially pure Python code, e.g. for the streamgraph.

In the modern backend, PostgreSQL is used as database technology instead of SQLite3. By keeping the data schema of the legacy backend, it is easily possible to import/export between the two database systems.

3.3 Flow of a search

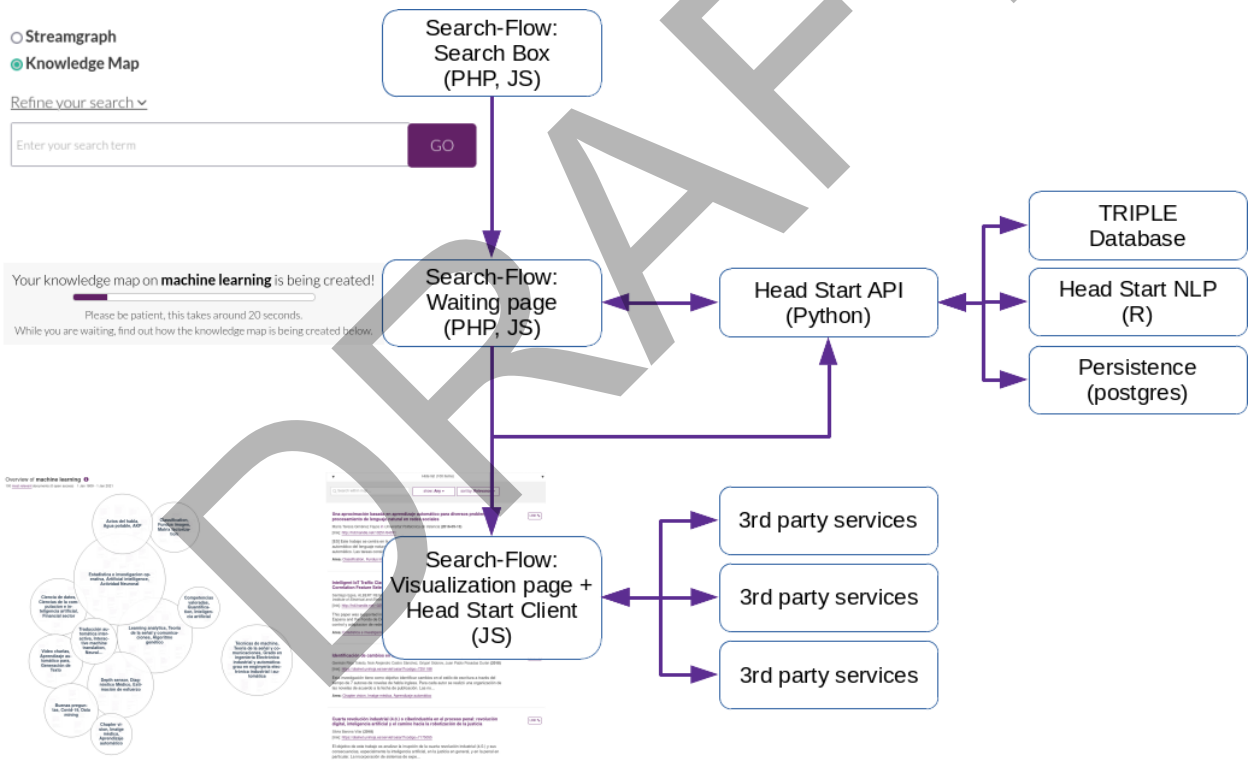

FIGURE 9. SEQUENCE DIAGRAM OF A SEARCH IN HEAD START

Figure 9 shows the flow of a search in Head Start in a sequence diagram. When a user starts a search in the search box, they proceed to a waiting page. In the background, the search flow communicates with the PHP service layer and the Head Start API. First, metadata of the search results is retrieved from the TRIPLE database. The metadata is then processed and the visualisation data stored in the Head Start database. Once the API reports a success, the user

proceeds to the visualisation page and the Head Start client initiates. During initiation, the visualisation data is retrieved via the PHP service layer and the Head Start API. The client, in turn, initiates a number of 3rd party services to enable additional capabilities, including enhanced PDF display and privacy-preserving analytics.

4| INTEGRATION STRATEGIES OF THE SERVICE IN GOTRIPLE

As depicted in Figure 10, Head Start is a tightly integrated innovative service in TRIPLE. This tight integration manifests itself in two areas:

1. On the frontend, Head Start interfaces can be accessed by users as part of the core platform.
2. On the backend, Head Start has a direct connection to the ElasticSearch index of GoTriple.

Head Start, however, does not run within the TRIPLE core. It is rather a hosted service on the Open Knowledge Maps servers, which is integrated via frontend and backend interfaces on the TRIPLE core platform. This is done to decouple the two systems and reduce complexity both for GoTriple as well as for the innovative service. How this integration works in practice is described in the following two sections.

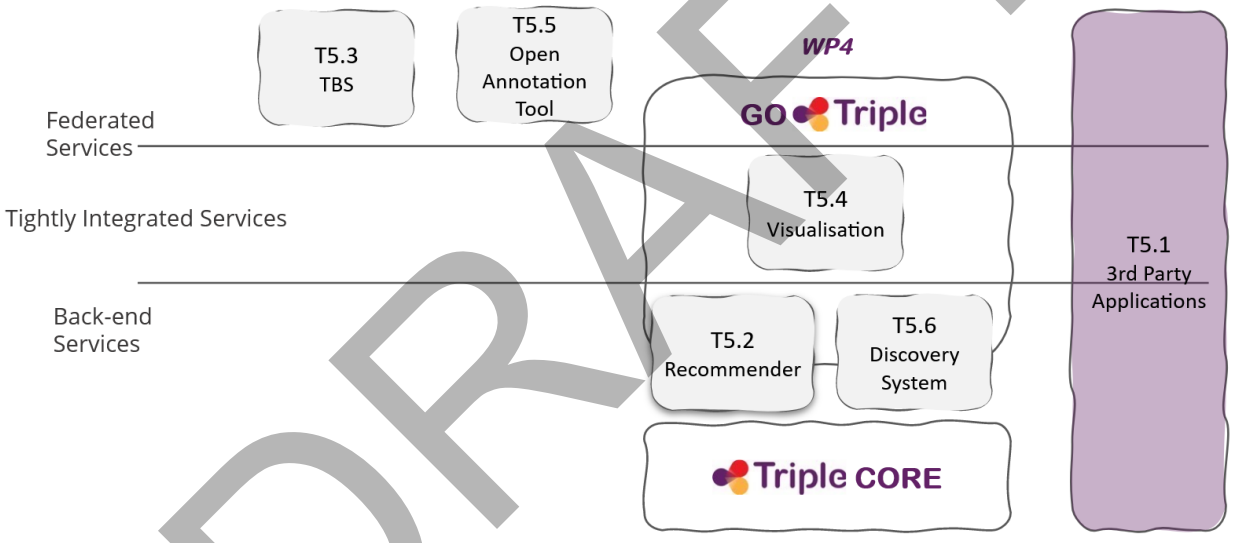

FIGURE 10. INNOVATIVE SERVICES IN TRIPLE AND THEIR LEVEL OF INTEGRATION (SOURCE: D6.2)

4.1 Frontend integration

On GoTriple, users can currently access the Head Start discovery system via the search results page. The integration is modeled after popular web search engines that link to additional services and interfaces directly within their search results page. If you enter the query “vienna” in a web search engine for example (see Figure 11), you will get a list of web pages related to the search term (e.g. the Wikipedia page on Vienna), but you will also get additional components, such as a preview of a geographical map of Vienna, images, and news stories. By clicking on any of these additional components, for example the geographical map, you are transported to the geomap service of the search engine, where you can explore the map further. As such, the

different services and interfaces are interlinked from a single access point, the main search results page.

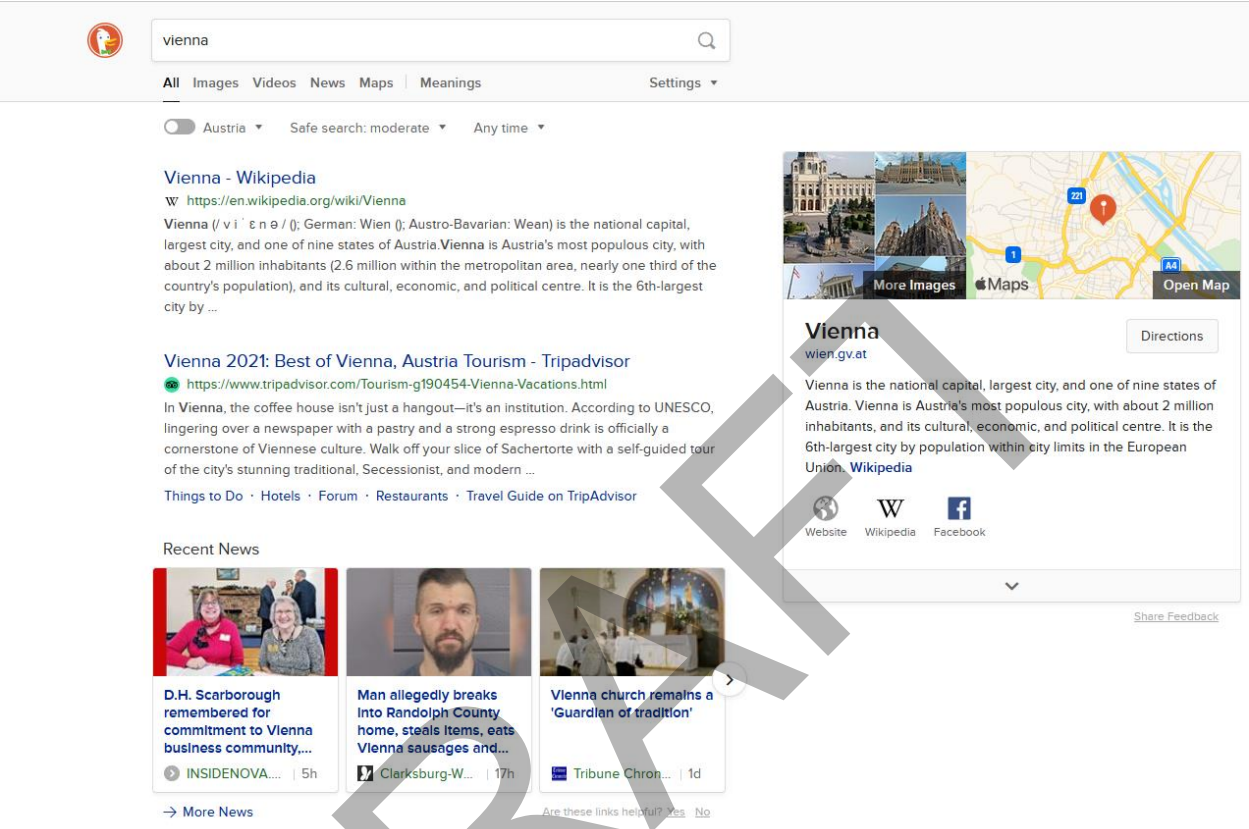

FIGURE 11. SEARCH RESULTS FOR THE TERM "VIENNA" IN DUCKDUCKGO

In GoTriple, this concept is applied to an academic search. When a user searches for a topic, they are presented with a list of search results as well as a preview image of knowledge map and streamgraph with a short description of either service (see Figure 12). When the user clicks on one of the two links (knowledge map or streamgraph), a new tab is opened showing a waiting page while the visual interface is being computed (see Figure 13). Once the computation has finished, the user is forwarded to the visual interface in the same tab (see Figure 14 for an example of a knowledge map and Figure 2 for an example of a streamgraph).

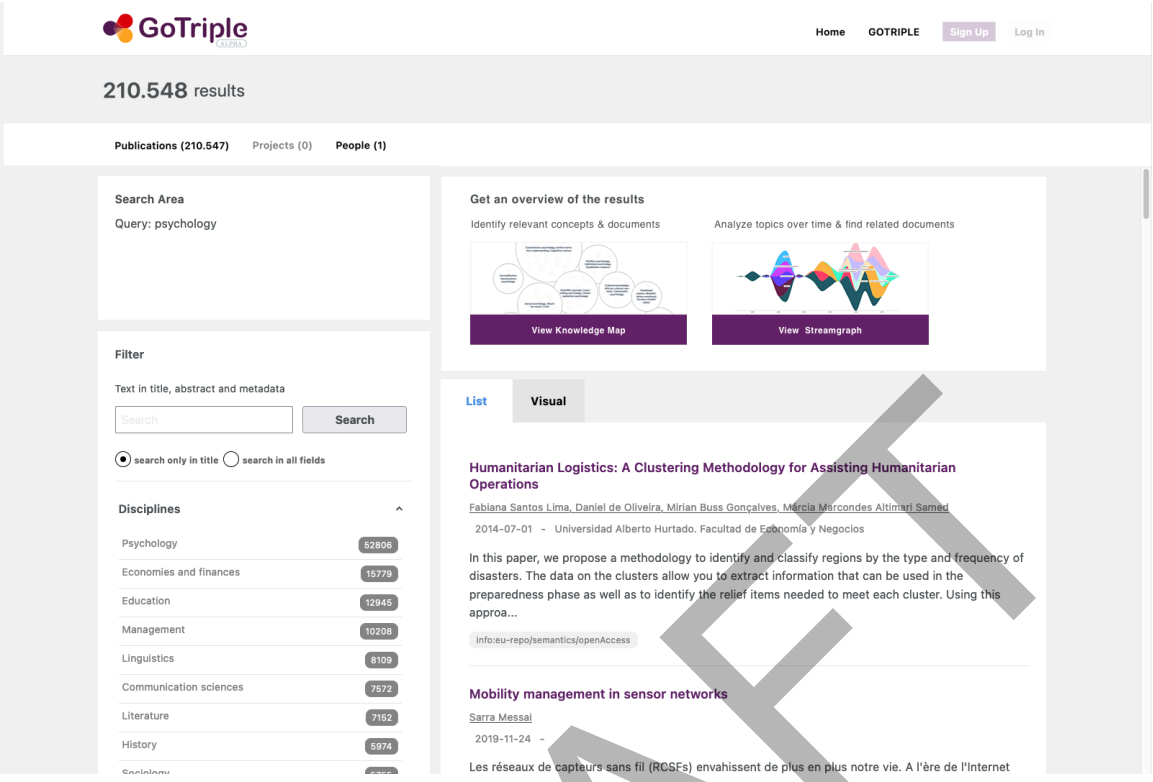

FIGURE 12. SKETCH OF THE INTEGRATION OF THE HEAD START DISCOVERY SYSTEM ON THE GOTRIPLE SEARCH RESULTS PAGE

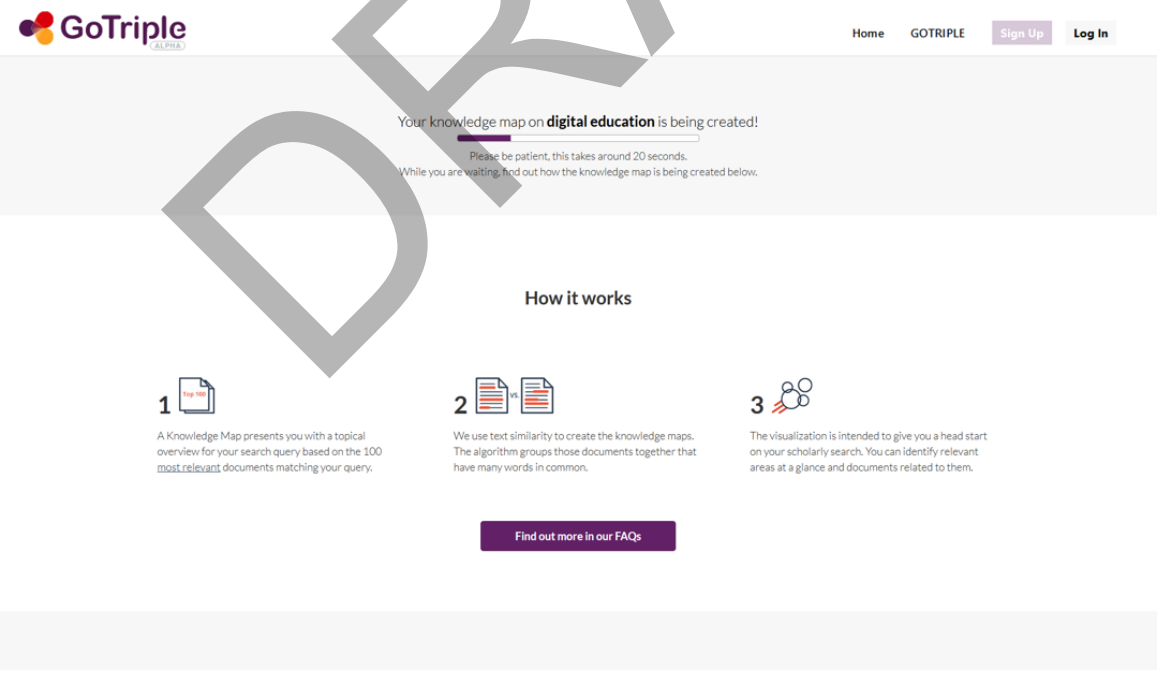

FIGURE 13. SCREENSHOT OF THE WAITING PAGE OF THE SEARCH FLOW ON GOTRIPLE

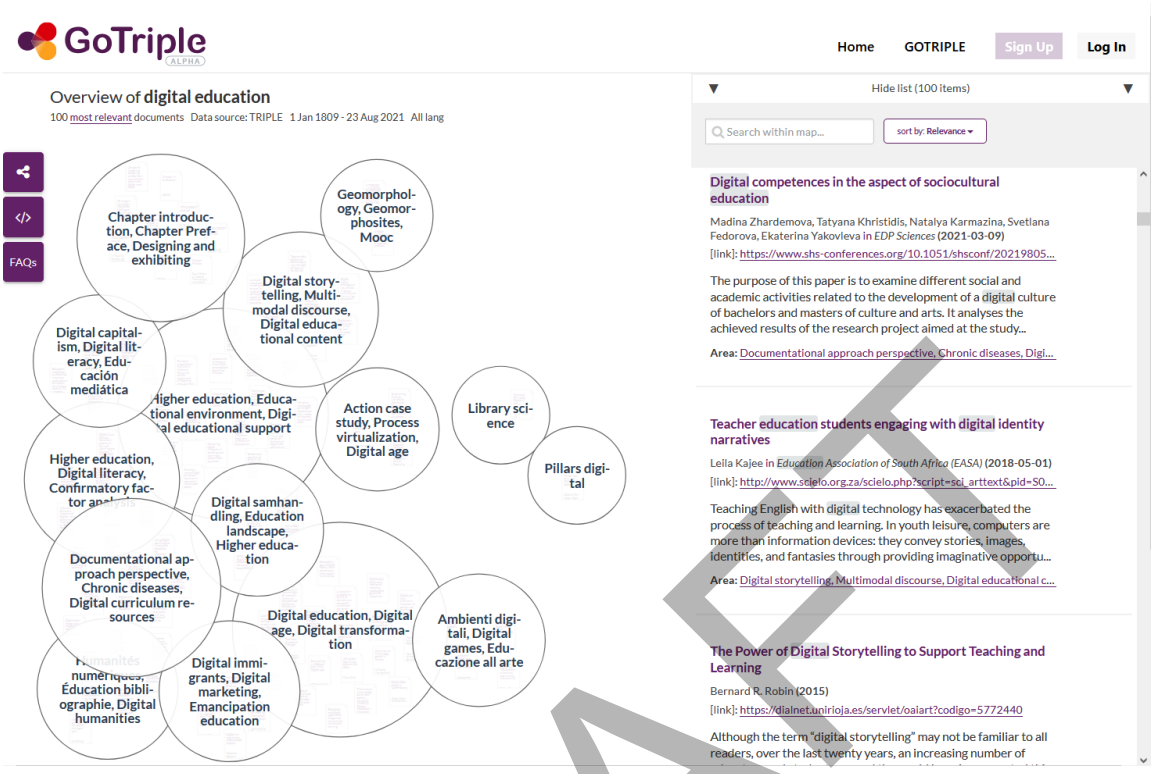

FIGURE 14. SCREENSHOT OF A KNOWLEDGE MAP ON THE TERM "DIGITAL EDUCATION"

4.2 Backend integration

Once the user clicks on one of the buttons/images in Figure 12, TRIPLE opens a new tab and forwards the search query and the search parameters to the Head Start search flow via a GET request in an iframe (see Figure 13). The search flow component parses the query string and calls the correct backend service (see chapter 3). The backend sends a search request to the TRIPLE ElasticSearch index. The results of the search are then fed into the Head Start machine learning pipeline. Once the representation of the visualisation is produced, it is stored in the database and the persistent visualisation ID is returned to the search flow. Finally, the visualisation interface is sent to the iframe (see Figure 14 and Figure 2).

For our development, we employ a multi-stage deployment workflow that includes all components (search flow, client, and containerized backend). This means there are separate deployments for e.g. feature-branches and a development review version, which gradually move towards production-readiness with a stable version for the actual integration. This is necessary to provide a version with a lower frequency of changes to TRIPLE, while at the same time allowing us to internally work on, test and review with a higher frequency of changes that will not impact external dependencies.

4.3 Lessons learned

The chosen approach on the backend of providing a hosted service with multiple instances with varying levels of stability to GoTriple has proven to be very successful. Changes in Head Start only need to be deployed on our servers, and depending on the state of the development, we can seamlessly switch between different integrations. This of course benefits our development cycle, but the main beneficiaries are partners who want to use our system within their platform. Compared to a self-hosted, on premise-solution, this considerably reduces the setup and maintenance effort on the partners' side.

This is in line with our experiences in other projects, and we have therefore suggested to move the GoTriple integration to a new infrastructure from Open Knowledge Maps, the so-called Custom Services. The Custom Services provide Head Start's visualisation capabilities as customizable cloud components on a shared, open and not-for-profit platform. This proposal is described in more detail in section 6.

Another successful approach on our end was to work with minimum viable products (MVPs). In the integration with the GoTriple platform, for example, we prioritized completing the pipeline over perfecting the individual parts of the pipeline. As such, any evaluation and testing can be done on a working pipeline, which makes design and development more efficient, and enabled us to improve the quality of the integration. This approach also helps with better managing our time, dependencies, and workload.

5| DESCRIPTION OF THE WORK DONE IN TASK D5.6

5.1 Refactoring of the Head Start backend

Within the project, we are performing a refactoring of the Head Start backend, which was not fit for the task in TRIPLE. This refactoring will be ongoing until the end of the project. In the first 24 months, we achieved the state that is described in chapter 3.

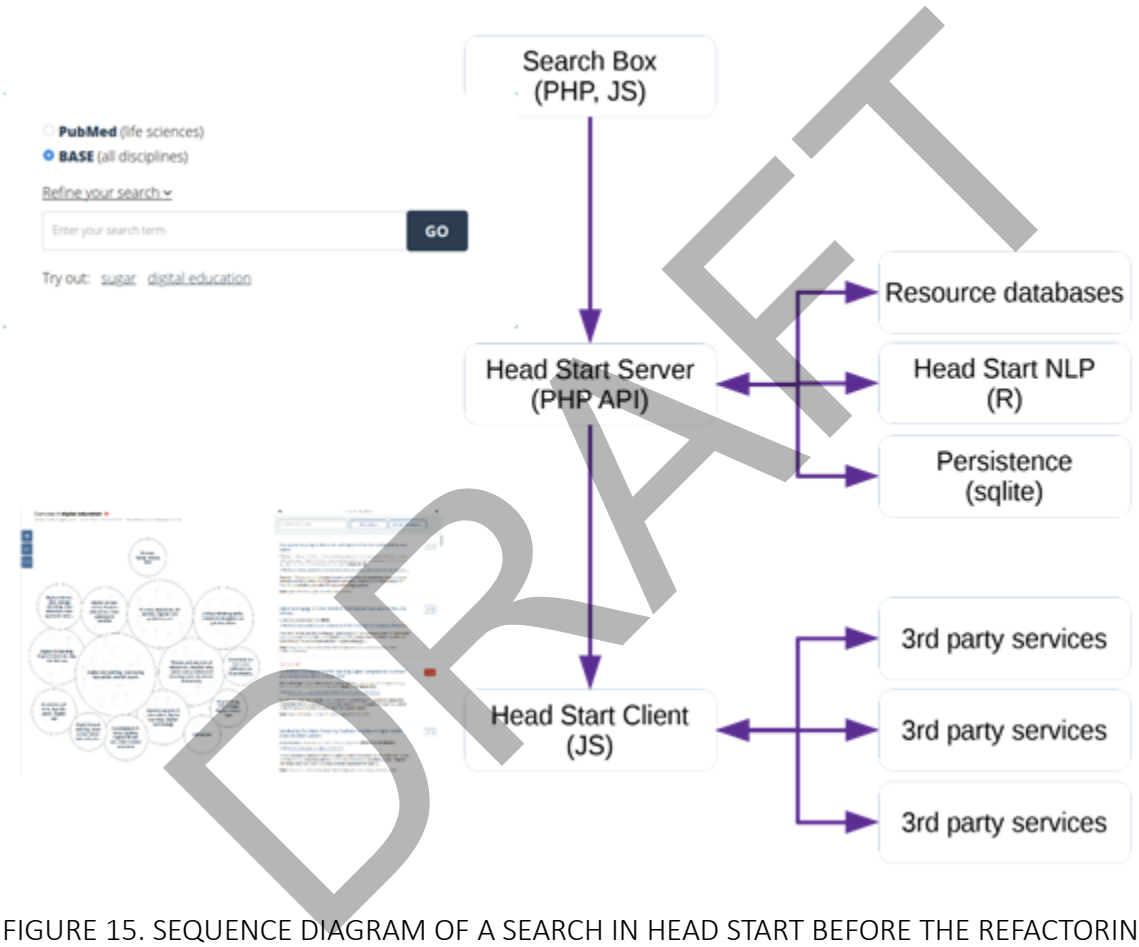

FIGURE 15. SEQUENCE DIAGRAM OF A SEARCH IN HEAD START BEFORE THE REFACTORING (CF. THE SEQUENCE DIAGRAM AFTER REFACTORING IN FIGURE 9)

An overview of the basic architecture of Head Start before the refactoring can be obtained from the sequence diagram of a search in Figure 15. The main components were a search box, a server instance and a browser-based client. The server component interfaces with data sources, computes knowledge map representations, and stores and retrieves map representations and associated metadata. The client component takes a map representation and enables users to

interactively explore the map. The search box was custom-created for each data source and provides specific search and filtering options and manages the search flow.

The search box component and the client component communicate with the server via a RESTful API-layer implemented in PHP. The server component takes search queries and search options, retrieves metadata from the associated data source, performs basic data cleaning and metadata enrichment, clusters documents according to their subject similarity, calculates the two-dimensional positioning of documents, and generates a content summary based on the most relevant keywords for each cluster. This map representation is then stored in an SQLite database together with search query and map-metadata, where it can be versioned.

The server component required a LAMP environment with additionally NodeJS and an R environment. The backend data processing components of the legacy version are executed directly on disk, as described in section "Software Architecture". Providing the R-dependencies directly on the host OS-level requires a convoluted setup procedure to avoid version conflicts with other software. This is a huge additional burden for initial deployment, regular software maintenance, and especially an impediment for development of additional features.

The monolithic system architecture reached its limitations. The close coupling of data-processing, persistence, and services made maintenance and extension of individual features more difficult. Additionally, SQLite is a limited persistence technology and was expected to be a performance bottleneck further down the line.

To overcome these issues, we directly addressed the shortcomings in a targeted refactoring. We introduced a microservice architecture to improve maintainability and extendability. All services were migrated into Docker containers to facilitate reproducible development environments and faster deployment. We also switched to PostgreSQL for a higher-performance persistence solution.

5.1.1 Microservice architecture

The drawback of a monolithic architecture is that it is difficult to make encapsulated changes. The organic growth over time of individual steps performed by the backend lead to a situation in which small changes, e.g. feature changes or changes in input data, could lead to unpredictable changes in behavior. In a monolithic architecture, small changes cascade through the system and root causes are difficult to identify. We managed to gain a higher level of control of the system through a change to a microservice architecture (see Figure 16), where specific parts of the pipeline, for example data retrieval, data processing, and persistence operations, are clearly separated and only interact through standardized interfaces and data formats.

Data is moved between micro-services via a message queue, for which we use Redis. Individual micro-services don't need to know about each other or the rest of the system. An upstream service or the API publishes a message onto a certain queue, and a downstream service that is subscribed to the same queue waits for data to arrive, processes it, and writes it to the next queue.

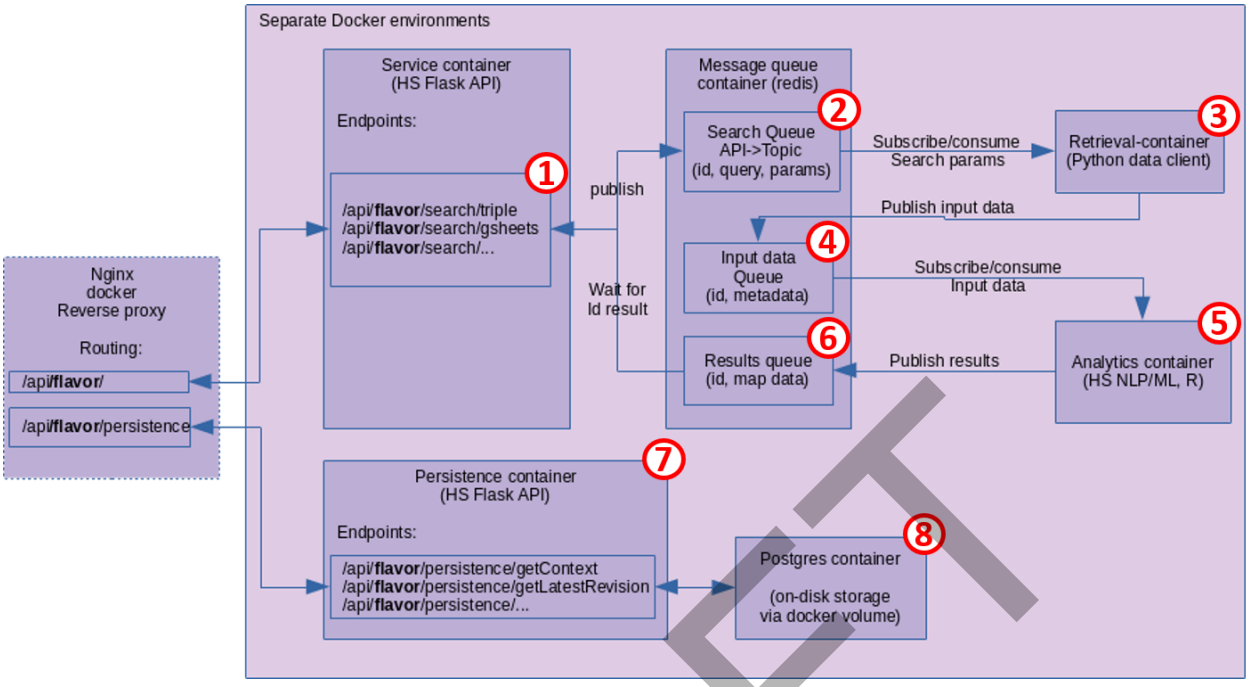

FIGURE 16: MICROSERVICE ARCHITECTURE OF THE MODERN BACKEND OF HEAD START AND THEIR INTEGRATION VIA A REVERSE PROXY SERVER, ANNOTATED WITH PROCESS STEPS

For example, a search from a user is sent as a POST request to the API layer (Figure 16, box 1). The API publishes it to the triple-queue, serialized in JSON format (box 2). From there the triple-data client consumes it, retrieves search results from the TRIPLE database, cleans and pre-processes the metadata (box 3), and publishes the pre-processed data to the input data queue (box 4). The data analytics service (box 5) consumes from the input data queue, and produces visualisation data not only for the TRIPLE integration, but for any other integration that could be running in the same environment. The visualisation data is published to the results queue (box 6), from where the waiting API consumes it and finally answers the request (box 1).

Compared with a monolithic architecture, each micro service has a clear purpose distinct from the others. Processing resources are also used more efficiently, as services can immediately start working on another task, without needing to wait until the rest of the pipeline has finished.

During the backend refactoring, data retrieval and processing steps also were separated from data persistence steps, which are now running as a separate service (boxes 7 and 8). This increases the reliability and robustness of the system.

5.1.2 Containerization with Docker

Containerization brings a number of benefits, e.g. separating conflicting dependencies, providing reproducible environments, and adding a layer of security by isolating microservices. It also

allows for horizontal scaling, should this be required at a future point: bottlenecks can be addressed in a targeted way, e.g. by adding more containers for the data processing step.

A few minor dev-ops improvements have been made as well, for example tagging containers with commit-ID during the build process. The deployment of software running in containers can now be controlled down to the level of an individual commit. It also creates the capability to run multiple versions of backend in parallel distinguished by their commit-ID, should this be required during review processes or during long-term cloud deployments.

5.1.3 Replacing SQLite with PostgreSQL

SQLite is a lightweight *embedded* relational database management system (RDBMS) that works well for many situations. There are, however, a few limitations that motivated the decision to switch to PostgreSQL, a *client-server* RDBMS. The main reason is to avoid having services directly reading and writing from an embedded database, but through a controlled API layer. Additional reasons are the limited availability of data types supported in SQLite, concurrency restrictions and limited authentication protocols and access controls. Since both are relational databases, it is easily possible to retain the data schema and migrate from one system to the other.

The PostgreSQL persistence has been directly implemented as a containerized micro-service, which is accessed via a restful API. Data is persisted in Docker volumes, which also provide functionality for data backups.

5.1.4 Other improvements

Overall performance improvements of the R-data analytics code were achieved through a combination of parallelization and refactoring of inefficient code.

Lastly, as a precautionary quality-of-service measure, a basic form of rate limiting was implemented. In the legacy version of Head Start, it limits the number of concurrent backend processing jobs to guarantee that already running jobs receive enough computing resources and finish in an acceptable amount of time. In the refactored version, the length of the input data queue is limited to a comparable value. In both versions, users receive a notification should their search run into a rate limit.

5.2 Standalone search flow package

5.2.1 Development of the search flow package

To connect GoTriple with Head Start in a standardized way, we created a standalone search flow package described in section 3. The search flow is a modular, customizable package that models all steps of the user's workflow: search form, waiting page and visualisation page.

Before the search flow package, this workflow was implemented for each data integration separately. This meant that the workflow could be easily customized to each use case, but also

that a lot of code was copied between different data integrations and that improvements of common components of the search flow for one data integration could not easily be transferred to another data integration. In addition, we had recently invested time to improve the workflow of our main discovery service on openknowledgemaps.org and we wanted to bring these improvements to TRIPLE and other data integrations.

Early on, we decided not to rewrite the whole code, but to extract existing code from the most advanced integration on openknowledgemaps.org and build on this code base. This was done due to time and resource constraints and the inherent risks of a complete rewrite. As a result, the search flow is a mix of PHP, JavaScript and HTML. PHP is used for component management and server-side functionality, JavaScript (JS) for client-side functionality and HTML as templates.

After the code was extracted, it was refactored where necessary and reorganized into PHP components that can be included in a data integration as needed. Common functions were moved into a shared code base, and a library of external dependencies was created. In the next step, we created a configuration for the whole package. Following the convention over configuration design pattern, users of the search-flow package must only override those parameters that they want to change from the standard configuration.

In addition, a new route to create Head Start interfaces was implemented: the GET route. Before the introduction of this route, visualisations could only be created using a POST request from a search form or by directly using the R scripts in the backend. The GET route enables triggering a search via a parameterized call to the waiting page. This behaviour is very useful for TRIPLE and enables the tight integration of Head Start in GoTriple described in chapter 4.

5.2.2 Improvement of search flow error messages

Helpful error messages are crucial to any interface and even more important in innovative tools. These tools offer features that are entirely new to the users and that might require a learning curve. Feedback and tips can considerably increase user interaction and lower any frustrations that users may experience. After a careful review of our searchflow we realized that our error message heuristic is still too basic and doesn't necessarily provide any helpful feedback for our users in order to continue their user journey ("Sorry we couldn't find any results. Please try again."). As a result, first time users might not return after they have encountered this basic error message on their first try.

Helpful feedback includes a clear description of what went wrong, possible reason(s) for the error (e.g. the time frame for your search was too narrow) and more specific tips on what to do next (e.g. increase the time frame in the filter options). We followed that structure in the error message for each error specified.

In the first step, we divided the reasons for a failed search into broad categories. To do so, we analysed the backend server logs, which are anonymously produced for each search. For each search, we log the search parameters, intermediate output from the data retrieval and machine learning pipeline, and the final outcome, be it success or failure. Logs are then aggregated and

their time series statistically analysed to flag errors, anomalies and outliers, which are then manually inspected. We arrived at the following overarching categories: no results from the API, a connection error, a timeout (e.g. when the response from the API takes too long), an error from the search API, an unexpected error on the server, and other errors (e.g. calling the waiting page without having entered a search term).

Second, we defined more specific reasons when there were not enough results. To do so, we looked into the subset of failed searches and compared it with the subset of successful searches. We used basic descriptive statistics to compare distributions and characteristics of search parameters between the two subsets. This included e.g. character length or number of tokens of query strings, length of date range between "from" and "to" dates, or use of optional parameters where applicable. This comparison helped us to identify failure reasons that have a high probability of appearing systematically. When a search now yields zero search results, we apply an error detection heuristic on the search parameters and provide a list of likely reasons such as typos or queries that are too long.

We then formulated tips as to how to improve the query depending on the error reasons. We also decided to include our search box alongside the message so users can either immediately correct their mistake or try another search based on the tips we provide them. The settings of their first search remain intact.

Finally, we also created an early prototype of a "Did you mean" feature, which is often employed in search engines to help users arrive at a search result. This feature suggests a spell-corrected query in case no results are returned. The prototype first attempts to determine the language of the query using a language detection library and then makes suggestions as to how to improve the query using a spell-checking library. Our first evaluation of the prototype showed that both routines are error-prone with short texts such as search queries. We therefore need to spend more time to evaluate and improve this feature before it can be used in a production setting.

5.3 Integration of Head Start interfaces into GoTriple

5.3.1 Design perspective

We have provided sketches of our suggestion on how to integrate links to our services (Knowledge Map and Streamgraph) on the search results page (see Figure 12) to the WP3 platform design team. The idea behind these sketches is to add context and a call to action as well as images or icons, so users get a clear idea of what they can expect from the Head Start services. Most users are, for example, not familiar with the terms Knowledge Map or Streamgraph.

Since then, we have discussed several ways of realizing the above points without overloading the results page. The results page will provide many functionalities which will "fight" for the users attention. It will be important to place these functionalities carefully along the workflow of the platform's users.

At the moment, the platform design team of WP3 is working to provide sketches for the different scenarios that were discussed before we make a final decision.

5.3.2 Technical perspective

From a technical point of view, the integration approach was agreed to follow a cloud-based concept. The service runs on OKMaps infrastructure and is being maintained by OKMaps, with easy integration and re-use by TRIPLE. From a technical point of view, we undertook a series of preparatory steps: First, coordination with the TRIPLE development team in a series of online meetings and a workshop at the consortium meeting. Second, defining requirements of a MVP (minimum viable product) that contains the essential features from a user perspective: typing in a search term within the GoTriple interface, and being able to navigate to a knowledge map or streamgraph visualisation.

The actual provision of the development version consisted of two complementary tasks: providing a TRIPLE-specific service running on the OKMaps development server, and an integration of the MVP via embedded iFrames in the alpha version of GoTriple. The GoTriple-specific service was deployed after conclusion of the refactoring tasks described in 5.1. The integration of the MVP in the GoTriple alpha version was achieved in coordination with the TRIPLE development team. The MVP successfully demonstrated the feasibility of the overall technical concept and served as a basis for the development and UX improvements that followed.

5.4 Data adaptations

An important task within TRIPLE is to adapt Head Start to the needs and requirements of the SSH disciplines. This also includes adapting the software to the GoTriple data. The work on these so-called data adaptations is described below.

Work on data adaptations started early in the project, with coordination and communication with work packages WP2 and WP4. Topics of consideration were the metadata schema, desired data properties in the TRIPLE ElasticSearch index, and expected content of the enrichment pipeline (language detection, enrichment from controlled vocabularies, classifications assigned by machine learning). The main challenge and opportunity here was and is the multilinguality of TRIPLE data.

Statistical analyses of distributions of metadata, e.g. overall coverage of parallel corpora, are a regular part of our rapid prototyping process, to better quantify possible usefulness for individual end users. Our own statistical analyses, together with results of user studies conducted as part of WP3 and input on future metadata from WP2, inform any specifications for adding value to knowledge maps and streamgraphs. Possible features which are enabled specifically by GoTriple data include re-use of multi-language concepts from controlled vocabularies and re-use of metadata fields available in multiple languages. An initial set of adaptations in this regard have been implemented in the machine learning pipeline and data

client. These include extract-transform-load (ETL) operations between the TRIPLE ElasticSearch metadata schema and our internal metadata schema, laying groundwork in our internal metadata schema for multilingual metadata, and preparing both backend and frontend data input-output operations for handling the new schema.

Given the iterative nature of the current development process, the GoTriple data client requires continuous updating with respect to upstream changes in the metadata, especially when new features are added (e.g. normalization of certain fields like publication date) or the actual content of the index changes, for example when new data aggregators have been added to the harvesting pipeline. To deal with the rapid iterative process, OKMAPS is running multiple development instances in parallel and aims for a fast and efficient build and deployment cycle.

6| NEXT STEPS AND EVOLUTIONS

Based on the positive experiences that we have made with providing Head Start as a hosted solution in TRIPLE as well as in other projects (see section 4.3), we have made a proposal to the TRIPLE consortium to move the GoTriple integration to a new infrastructure from Open Knowledge Maps, the so-called Custom Services. The Custom Services provide Head Start's visualisation capabilities as customizable cloud components on a shared, open and not-for-profit platform. Switching to the Custom Services would mean that any future improvements of Head Start would automatically be available to GoTriple, even beyond the end of the TRIPLE project. It would also mean that the integration and maintenance costs are drastically reduced on the GoTriple side as the Custom Services are not running on GoTriple servers, but are provided as cloud services by Open Knowledge Maps.

However, in the Custom Services model, GoTriple data would not be directly accessed by Head Start, but rather consumed via BASE, the Bielefeld Academic Search Engine. The plan would therefore require to create a mapping between the data schemas of GoTriple and BASE as well as implementing an OAI-PMH endpoint to the GoTriple data. The proposal to use the Custom Services model is still under discussion by the TRIPLE consortium, and as a result, the plan for M24-40 is not yet fully clear. Below, we have listed developments in case the Custom Services plan is accepted and in case it is rejected.

In case of acceptance of the Custom Services plan, one of the main tasks will be migrating the BASE data client into the new architecture so that it can be used within GoTriple. We will also extend the retrieval and analytics containers so that streamgraphs can be created on top of BASE data. Additionally, we will migrate multi-language capabilities from the GoTriple integration to the Custom Services. In case of proceeding with the original plan, hence without the Custom Services, the main task will be to continue to adapt the integration to new types of metadata in the backend and to improve its stability by for example enhancing our existing fall-back solutions. We will also extend the visual search capabilities of the GoTriple integration such as enabling boolean queries. Additionally, we will start the discussion around the deployment of Head Start on GoTriple servers. Tasks that will happen in either of the cases are a continued refactoring of the search flow, the backend service/API layer, and the data processing services.

REFERENCES

Kraker, P., Schramm, M., & Kittel, C. (2019). Open Knowledge Maps: Visual Discovery Based on the Principles of Open Science. Communications of the Association of Austrian Librarians, 72(2), 460–477. https://doi.org/10.31263/voebm.v72i2.3202