DataDiver Project: Final Report, Individual Contributions & Documentation

Atilla Cetin - 5757200

August 21, 2024

1 Introduction

In our team project, we used the VuFind software (version 9.1.1) to provide scientific articles in computer science on our own platform. We integrated data from `core.ac.uk` and adapted the system to our needs. In this report, I will list my contributions and thoroughly document the work done.

2 My Contributions (In short)

- **Data Integration from `core.ac.uk`**:
  - API queries to collect computer science papers.
  - Converted JSON data into MARC21 format.
  - Developed Python scripts; automation by teammate Davide (see below).
- **Backend and Frontend Support**:
  - Solution for CSS design adjustments on different pages.
  - PHP script for URL detection and specific page adjustments.
- **Email Verification and MySQL Management**:
  - Set up email verification in VuFind.
  - Installed and configured Mailhog for local email testing.
  - Created MySQL guide for administrators.

3 Documentation

3.1 Data Integration from core.ac.uk

Data Source and API Use:
Our project aimed to obtain computer science articles from core.ac.uk.
core.ac.uk is a large search engine with over 300 million research papers from various fields. For our project, we only needed articles from computer science.

The queries to the core.ac.uk API were configured to return only relevant articles filtered by the research area "Computer Science". Examples of such API queries are:

- To retrieve 25 computer science articles:
  https://api.core.ac.uk/v3/search/works/?q=%22Computer+Science%22&limit=25
- To get 100 articles with a filter by field:
  https://api.core.ac.uk/v3/search/works/?q=+AND+
  fieldsOfStudy%3A%22computer+science%22&limit=100

These API queries provided results in JSON format.

Data Format and Conversion:
An important step was to convert the JSON data from core.ac.uk into MARC21 format. MARC21 is a format recognized by library systems like VuFind and Solr. The challenge was to correctly transfer the structure of the JSON data into MARC21 format so that the data could be easily integrated into our search system.

An example of a JSON entry from core.ac.uk looks like this:

```
{
    "acceptedDate": "2010-03-01T00:00:00",
    "arxivId": "1003.0219",
    "authors": [
        {"name": "Malioutov, Dmitry"},
        {"name": "Sanghavi, Sujay"},
        {"name": "Willsky, Alan"}
    ],
    "title": "Sequential Compressed Sensing",
    "yearPublished": 2009,
    "links": [
        {"type": "download", "url": "http://arxiv.org/abs/1003.0219"}
    ]
}
```

This data needed to be converted into MARC21 format to be included in VuFind and Solr. A first Python script was developed to convert JSON data into the desired structure for VuFind. Then a second Python script was used to convert this structured data into MARC21 format and save it in a file named output.mrc.

The `output.mrc` file could then be imported into VuFind's search system using Solr commands (`./import-marc.bat "path"` on Windows). My teammate Davide Deho later developed and automated the process so that the steps from data collection to importing into VuFind now run completely without manual intervention (see his section below).

3.2 Backend and Frontend Support

Design Challenges and Solution:

In the frontend, we faced a problem due to the uniform handling of all pages in VuFind. Since all pages are controlled by the same CSS file, it was difficult to make specific design adjustments without causing unwanted effects on other pages. For example, changes to the background color on one page could negatively impact the design of another page. To solve this problem, I developed a PHP script that detects the URL of the current page and assigns specific CSS classes. These classes were then used in the CSS files to individually adjust each page without styles affecting each other.

The PHP code I added to the `layout.phtml` file works as follows:

- The script retrieves the URL of the current page and assigns a specific CSS class to the `$bodyClass` variable based on this URL. These classes were defined as follows:
  - `body.homepage` for the homepage,
  - `body.searchpage` for the search page,
  - `body.record-page` for individual result pages,
  - Other specific pages like `body.author-page` or `body.tag-page`.

  Here is an excerpt from the code:

```
<?php
$currentUrl = $_SERVER['REQUEST_URI'];
$bodyClass = '';

if (strpos($currentUrl, '/vufind/Search/History') !== false) {
    $bodyClass = 'acc-search-history-page';
} elseif (strpos($currentUrl, '/vufind/Search/Advanced') !== false) {
    $bodyClass = 'advanced-search';
} elseif (strpos($currentUrl, '/vufind/Author/') !== false) {
    $bodyClass = 'author-page';
} elseif (strpos($currentUrl, '/vufind/Record/') !== false) {
    $bodyClass = 'record-page';
} elseif (strpos($currentUrl, '/vufind/Tag/') !== false) {
    $bodyClass = 'tag-page';
} elseif (strpos($currentUrl, '/vufind/MyResearch') !== false) {
    $bodyClass = 'account-page';
} elseif (strpos($currentUrl, '/vufind/Search/') !== false) {
  $bodyClass = 'search-page';
} elseif (strpos($currentUrl, '/vufind/') !== false) {
    $bodyClass = 'homepage';
```

```
}
?>
```

With this solution, we ensured that each page received an individual design. This allowed us to make specific adjustments to background colors, fonts, and other design elements without conflicts with other pages.

3.3 Email Verification and MySQL Database Management

Email Verification:

An important security measure in VuFind is email verification for new user accounts. This verification ensures that only real user accounts are created and allows users to change their email address or delete their account.

Since our VuFind system runs on a local server, it could not send real emails to users. To test this function, I installed Mailhog. Mailhog is a tool developed specifically for developers to emulate emails locally.

In the configuration file `config.ini`, I enabled email verification and adjusted the settings in the `[Mail]` section so that emails are forwarded to the Mailhog server. This way, we could test email verification locally and ensure it works.
For use on a real web server, the settings in `config.ini` would need to be adjusted to use a real email server that supports TLS/SSL.

MySQL Database Management:

To manage the database on which VuFind is based, I created a small guide for administrators. This guide includes the basic MySQL commands needed to manage the main database tables in VuFind.
Using MySQL Workbench or the MySQL Command Client, administrators can view, edit, or delete entries in various tables such as `user`, `user_list`, `tags`, and `comments`. This is particularly useful for removing inappropriate tags or deleting spam accounts.

Small introduction and guide for administrators who are not familiar with SQL:

To open the VuFind database:

```
SHOW DATABASES;
USE VUFIND;
SHOW TABLES;
```

To display all stored tags:

```
SELECT * FROM tags;
```

To delete a specific tag:

```
DELETE FROM tags WHERE tag = "tag_name";
```

These commands allow administrators to manage the database efficiently and quickly respond to support requests or remove unwanted content.

4 Conclusion

In this project, I successfully contributed to the integration and customization of the VuFind software, particularly in handling and transforming research data as well as adjusting the user interface. These experiences have deepened my understanding of data management and system customization, and have strengthened my skills in the practical application of backend technologies.

Team Project Report: Integration of CORE Data into VuFind

Davide Deho

August 21, 2024

In this project, our team integrated data from CORE.ac.uk into the VuFind discovery interface, focusing specifically on computer science papers. My primary responsibilities included backend data processing, script automation, and assistance with frontend design. Below is a detailed account of the tasks I completed during the project.

Backend: Automation of CORE Data Retrieval, Conversion, and Importing in Vufind

- **GitHub Repository Creation:** I established a GitHub repository for our project to facilitate easy sharing of code and changes among team members. This ensured that our CSS edits and other modifications were up-to-date and not overwritten or duplicated.

- **Python Script for JSON to MARC Conversion:** I used a base script provided by my colleague Atilla Cetin, who utilized the PyMARC library as a starting point. My contribution involved integrating this script as a part of my code for automatically converting JSON files to MARC21 files.

- **Data Retrieval via API:** Initially, we faced challenges with data retrieval, as we would have needed to download the entire 2TB database. I optimized the process by employing HTTP GET requests, first testing in the Boomerang Chrome extension and then implementing it with Python's Requests library. This sends a request to the core API, which then sends back the requested data. This is much better then having a hardcoded database, as it does not take up any space on the hard disk and it can import new data on the fly based on the current needs.

- **Data Structuring & Filtering:** The API's JSON response contained useless data, for example the header of the response (we only need the content) and sometimes the fields were named differently, which made it difficult to convert them properly. Furthermore, we only had to fetch the metadata of the articles for this assignement. This means the full text of the papers was unnecessary, so I removed it for performance reasons, which made the data faster to import and use and also reduced the space requirements. My colleagues Atilla Cetin and Leys Koeksal developed a script to structure the data appropriately for MARC conversion. However this first changed the JSON files and then converted them to MARC. I improved this by eliminating the middle step and mapping the fields directly from the JSON fiels to the corresponding MARC fields, without need to change the JSON file first.

- **Data Retrieval Automation:** I automated the data retrieval and import process with a script that uses the console command:

  ```
  Python GetConvertedData.py fieldsOfStudy:"computer+science" 100 [Your/VuFind/Directory/Output.mrc]
  ```

  This script fetches a specified number of entries based on a search query and saves the output in a user-defined directory. It also automatically initiates Solr in the target directory for direct data import into VuFind, provided the Solr executable resides in the same folder.
  This is all done within a single call to speed up and make easy the process of getting data.

- **Script Functionality:** The script processes data by first performing an HTTP GET request to retrieve JSON data, which is then converted to MARC format. During conversion, a language lookup map ensures correct insertion of language codes in the MARC fields.
  After the conversion the script fires up solr in the target folder specified by the user. This might not be necessary if solr has already been started, but it makes sure that solr is activated as we need it to be on for importing data in vufind.
  The script then calls the import-marc.bat script in the target folder with the file path specified by the user as an argument.
  For Debugging, the script outputs in the console how many records were taken, how much time the get request and the conversion took, and if it was done succesfully.
  After this, the process finishes and the data is ready for use in vufind.

- **How to Use the Script**
  - `fieldsOfStudy:"%22computer+science%22":` This argument specifies the query used to search for papers. In this case, it searches for papers in the field of computer science.
  - `100:` This number indicates the maximum number of papers to fetch and convert.
  - `[Your/VuFind/Directory/OutputFile.mrc]:` This is the output file where the converted MARC21 records will be saved. Replace this with the path to your desired output file location. Please keep in mind that the file must be saved in the /vufind folder, otherwise the script will be unable to open solr and import the converted files.

Modifying the Command:

- **Changing the Field of Study:** To search for papers in a different field, modify the `fieldsOfStudy` parameter.
  For example, to search for physics papers, use `fieldsOfStudy:"%22physics%22"`.
- **Adjusting the Number of Papers:** Change the `100` to any other number based on how many papers you want to retrieve and convert.
- **Output Location:** Modify the output file path to save the MARC21 records in a different location.

- **MARC Data Fine-Tuning:** I handled the integration of language fields by creating a reverse lookup table (ReverseLanguageMap.json) to match original language data to VuFind's expected format (e.g., "English" to "eng").
  Additionally, I added the document type field to the MARC records, aligning input document types (e.g., research) with VuFind's categories (e.g., article).

Frontend: Assistance to the Frontend Team

Since the backend tasks were completed ahead of schedule, I contributed to the frontend development by providing design ideas and implementing various improvements.

- **Design Brainstorming:** I collaborated with the frontend team on Canva.com to develop design concepts, such as a minimalistic homepage with a prominent search bar. These ideas were subsequently implemented by the frontend team.
- **General Enhancements:** I made several frontend adjustments, including resizing buttons, optimizing text alignment, adding hover effects, and ensuring that the color scheme was consistent across the website.

Conclusion

My main role in this project was to facilitate data retrieval and automate the testing and data processing workflows. This was crucial for allowing the rest of the team to efficiently test and modify the data, which is essential for the successful implementation and testing of the project in VuFind.

DataDiver Project Report

Leys Koeksal - 5857986

Introduction

DataDiver is a full-stack application designed as a search engine for scientific computer science papers. The primary objective of this project was to create a user-friendly platform where researchers and academics can easily search, access, and manage scientific papers. The project involved developing a robust backend system to fetch paper metadata, convert them into a standardized MARC21 format, and implement a frontend interface that allows users to interact with the data seamlessly.

This report details my contributions to the project, focusing on backend development (particularly data retrieval and conversion processes) and my involvement in the general layout and design of the website.

My Contributions

Backend Development

1. Converting Metadata to MARC21

I also contributed significantly to the module that converts the fetched metadata into the MARC21 format. This functionality was encapsulated in the `convert.py` script.

Key Features:

- **Field Mapping:** Each field in the JSON metadata was mapped to the appropriate MARC21 fields. For instance, the paper's title was mapped to the `245 $a` field, and the authors were mapped to the `100 $a` and `700 $a` fields (`convert.py`).
- **Language Code Conversion:** The script includes a custom mapping to convert language names from the metadata into MARC21 language codes, ensuring compatibility with library cataloging standards (`convert.py`).
- **Integration with VuFind:** After converting the metadata into MARC21, the records are prepared for import into the VuFind system, a popular open-source library search engine. The script automates the process of starting the Solr service and importing the MARC21 files into VuFind (`convert.py`).

MARC21 Tags Overview:

The following is a summary of the MARC21 tags that the `convert.py` script populates for each paper:

001 - Control Number (ID)

005 - Date and Time of Latest Transaction (updatedDate)

008 - Fixed-Length Data Elements (Language code in positions 35-37)

024 - Other Standard Identifier

035 - System Control Number

041 $a - Language Code

100 $a - Main Entry - Personal Name (Primary Author)

245 $a - Title Statement

260 $c - Date of Publication

500 - General Note (Accepted Date)

502 - Dissertation/Thesis Note

505 - Formatted Contents Note

650 $a - Topical Term (Subject)

655 - Index Term - Genre/Form (Document Type)

700 $a - Added Entry - Personal Name (Additional Author)

710 $a - Added Entry - Corporate Name (Institutional Author)

730 $a - Uniform Title

773 $i - Host Item Entry (Relationship Information)

776 - Additional Physical Form Entry

780 - Preceding Entry

785 - Succeeding Entry

856 $u - Electronic Location and Access (Download URL)

022 $a - International Standard Serial Number (ISSN)

440 - Series Statement/Added Entry (Series Title)

490 - Series Statement (Series Title, untraced)

Common Subfields:

$a - Primary data for the tag (e.g., Name, Title)

$u - URL for electronic access

$c - Date in publication details

$t - Title in relationship entries

$b - Publisher or distributor

$x - ISSN, ISBN, or other standard numbers

$i - Relationship information in host items

Frontend Development

In addition to backend development, I contributed to the general layout and design of the *DataDiver* website. My work involved creating and refining the CSS to ensure the website was visually appealing and user-friendly.

Key Contributions:

- **Layout Design:** Structured the layout of the web pages to ensure a logical flow of information and ease of navigation.
- **Styling:** Applied consistent styling to all elements, including typography, color schemes, and button designs, to create a cohesive user experience.

Challenges and Solutions

Challenge: Data Conversion Complexity

One of the major challenges faced was the complexity of converting varied JSON metadata formats into the highly structured MARC21 format. This required meticulous field mapping and handling of edge cases, such as missing or incomplete data.

Solution: To address this, I developed a robust mapping function that checks for the presence of necessary fields and applies default values or skips fields as appropriate to avoid errors during conversion (`convert.py`):

Conditional Mapping: The function checks each metadata field in the JSON object to determine whether it exists and if it contains valid data. For instance, before attempting to map an author's name to the 100 $a or 700 $a fields, the function verifies that the name is present and correctly formatted.

Handling Nested and Complex Data: Some metadata fields, such as author affiliations or identifiers (like DOI), were nested within the JSON structure. The mapping function included logic to navigate these nested objects, extract the relevant information, and correctly format it for MARC21.

Challenge: Missing Keywords in Metadata

One significant challenge was the lack of keywords in the metadata retrieved from the CORE API. Keywords are vital for enhancing searchability and ensuring that the papers are correctly indexed in the library system.

In the development of DataDiver, one of the key objectives was to ensure that users could not only search for specific papers but also discover similar and relevant articles based on the content of the papers they were interested in. Initially, our approach relied heavily on subject-based matching, using the metadata provided by the CORE API. This metadata typically includes fields such as the paper's title, authors, publication year, and subjects. However, since all the papers in our database are categorized under the broad subject of "computer science," the subject-based matching approach led to suboptimal results. Specifically, the system recommended nearly every paper as similar, regardless of its specific focus within the computer science domain. This issue significantly hindered the effectiveness of the recommendation engine, reducing its value to users who were seeking papers closely related to their research interests.

Solution:

We addressed this issue by integrating the Cortical API into the conversion process. This approach ensured that even papers without predefined keywords could be accurately categorized and searched (`convert.py`):

1. For each paper retrieved from the CORE API, the system checks whether an abstract is available. The abstract is a crucial element because it distills the paper's main ideas, making it an ideal source for keyword extraction.

2. If an abstract is present, the text is sent to the Cortical.io API. The API uses sophisticated semantic analysis techniques to scan the abstract and identify keywords that accurately represent the paper's key topics. This process involves analyzing the relationships between words and concepts, rather than merely counting word frequencies, which allows the API to capture the essence of the paper's content more effectively.

3. The Cortical.io API returns a set of keywords derived from the abstract. These keywords are carefully selected to reflect the main themes and subjects discussed in the paper, ensuring that they are relevant and representative of the paper's content.

4. Once the keywords are extracted, they are incorporated into the MARC21 record for the paper. Specifically, the keywords are inserted into the 650 $a field, which is designated for topical terms (i.e., subjects) in the MARC21 format. By populating this field with content-driven keywords, we enhance the metadata for each paper, providing a richer and more informative dataset for the recommendation engine to work with.

Efficiency: Because the free plan of the Cortical.io API does only allow two searches per second we had to harcode a timeout (sleep) after every paper scan for 0.5 seconds. We conducted thorough testing to assess the impact of the additional processing required for keyword

extraction. The results showed that, while there was a slight increase in processing time, the benefits of more accurate and relevant recommendations far outweighed this cost. The system remained responsive and capable of handling large volumes of data without degradation in performance.

Challenge: Handling Large API Responses

Handling large volumes of metadata efficiently, especially when dealing with extensive API responses, was another challenge.

Solution: We implemented data filtering mechanisms to discard unnecessary fields early in the processing pipeline. This approach reduced memory usage and improved performance during subsequent steps, ensuring that the system remained responsive even with large datasets (`GetCoreData.py`).

Conclusion

Contributing to the *DataDiver* project allowed me to gain extensive experience in backend development, particularly in working with APIs and data conversion processes for library systems. I also had the opportunity to apply best practices in frontend design, focusing on creating a responsive and user-friendly interface.

This project not only resulted in a fully functional search engine for scientific papers but also provided a comprehensive learning experience, allowing me to tackle real-world challenges and develop effective solutions.

DataDiver Frontend Report

Team: Ermal Balaj - 5745047, Eduard Ajupi - 5744404

20.08.2024

Frontend Design Process

Theme Selection and Custom Theme Creation

Initially, we had to choose a suitable VuFind theme to inherit from and use as the basis for our website. Since we were working with Bootstrap 4.6, we decided on the Bootstrap3 theme. This theme provided a robust foundation for responsive design, ensuring an optimal user experience across various devices.

After selecting the theme, we started developing our custom theme based on Bootstrap3. This involved creating the necessary folder structures and files, including CSS files, images, and templates. A key component was our custom configuration file, allowing us to make specific adjustments to the design and functionality to ensure seamless integration with the search engine features.

Development of Design Templates

To visualize the final look of the website, we initially created design templates using Canvas. These mockups served as visual guidelines for our final design. We drew inspiration from existing VuFind-based websites found on the VuFind website.

After reviewing several of these templates, we developed our first design (see Image 1, page 4 and 5). However, this design was too simple and did not meet our high standards for a modern and user-friendly interface, leading us to create a more complex design (see Image 2, page 4 and 5). We were so pleased with this design that we decided to change the website's name from "Youfind" to "DataDiver" as it better suited the new design and our slogan "Dive into Knowledge."

To create an even more visually striking design, we developed a third version (see Image 3, page 4 and 5). This design met all our criteria for an eye-catching interface and was chosen as the final design for our website. The color scheme, which primarily includes shades of blue and violet, creates a calming and professional atmosphere. These colors symbolize innovation, which is particularly relevant for a search engine in the computer science field.

Elements like animated background (GIF/WEBP), usage of glass effects and the harmonious color scheme give the website a modern look, inspired by the "Frutiger Aero" style popular in the 2000s, but with a modern twist.

Logo Design

A central element of our design was the creation of a logo that reflects the character and function of the website. The logo consists of a diving mask, with air bubbles representing the lens of a magnifying glass. This visual metaphor combines the concept of diving (in terms of deep research) with the functionality of a search engine. The logo thus symbolizes not only the act of "diving" into knowledge but also the efficiency and precision of searching for academic papers. The creative use of symbols makes the logo memorable and reinforces the website's brand identity.

Design Implementation

Once the final design was established, we began implementing it in code. We first modified the raw Bootstrap3 theme by, for example, editing the `footer.phtml` file to remove the footer entirely. This allowed us to center the search bar and logo, enhancing the website's usability and visual appeal.

We made further CSS modifications to refine the design. A key feature of the homepage is the animated background (GIF/WEBP) and the use of glass effects, along with a harmonious color scheme.

One challenge we faced was that CSS does not support MP4 files. To overcome this, we converted the background video to the WebP format. The glass effects were added to reinforce the website's diving theme, as they resemble an aquarium, thus enhancing the concept of "diving" into a world of knowledge.

Button Design and CSS Effects

Another focus was on the design of the buttons, which play a central role in user navigation. To achieve the best possible design, we analyzed various websites featuring a wide range of button designs. From these inspirations, we derived the final design of our buttons.

To incorporate the desired glass effects into the buttons, we used various CSS techniques, including `linear-gradient` for the background and `backdrop-filter` to achieve the desired translucent effect. The combination of these techniques resulted in buttons that are not only functional but also visually appealing.

Example of the CSS code for a button:

```
.result-body {
    background: linear-gradient(135deg, rgb(235 50 255 / 42%), rgba(255, 255, 255,
    backdrop-filter: blur(10px);
    -webkit-backdrop-filter: blur(10px);
    ...
}

.btn-primary {
    color: #fff;
    /* background-color: #ff3532; */
    background: linear-gradient(to right, #ff3131, #ff5d3e, #ff8b4b);
    font-weight: bold;
    font-size: 1.6vw;
```

```
    ...
}
```

Custom Fonts

To further enhance the website's visual identity, we used custom fonts for both the text and the headings, which we embedded in CSS. For the headings, we chose the "Orbitron" font, and for the text, we used "Jaturat." These fonts had to be downloaded and placed in the CSS folder. They were then linked as a resource in the `modified.css` file. The corresponding CSS code is as follows:

```
@font-face {
  font-family: 'CustomFont'; /* The name of the font */
  src: url("../css/fonts/Jaturat.ttf") format('truetype');
  font-weight: normal; /* Or 'bold', depending on the font */
  font-style: normal; /* Or 'italic', depending on the font */
}
```

The use of these fonts significantly contributes to the website's aesthetics, as they emphasize the modern and technical appearance and thus contribute to the overall impact of the site.

Fine-Tuning and Optimization

Once the basic structure of the website was established, we focused on fine-tuning. This included adjusting the login and registration windows, editing the "Search Advanced" page, and other detailed tasks. Many of these adjustments were efficiently implemented by reusing and extending previously written code.

During the development process, we closely worked with the developer tools of various browsers, which helped us ensure a functioning responsive design. Media Queries were particularly useful in adapting the layout to different screen sizes.

```
@media (min-width: 1200px) {
    .container, footer {
        width: 80vw;
    }
}
```

Conclusion

Through close collaboration and iterative design processes, we created a website that is both aesthetically pleasing and user-friendly. The final design is not only visually appealing but also supports the functionality of the search engine by providing users with an intuitive and pleasant interface. The logo, which effectively represents the website's character, along with the carefully designed buttons and the thoughtful selection of custom fonts, significantly contributes to the positive user experience. It also helped us to gain experience in the field of Frontend and how to deal with various tasks that doesn't need just to write code as a solution, but it's also needed

to understand how things are connected together and how they depend on each other. We also want to thank the Backend team for helping out for some of the problems we've been facing(see the Backend report).

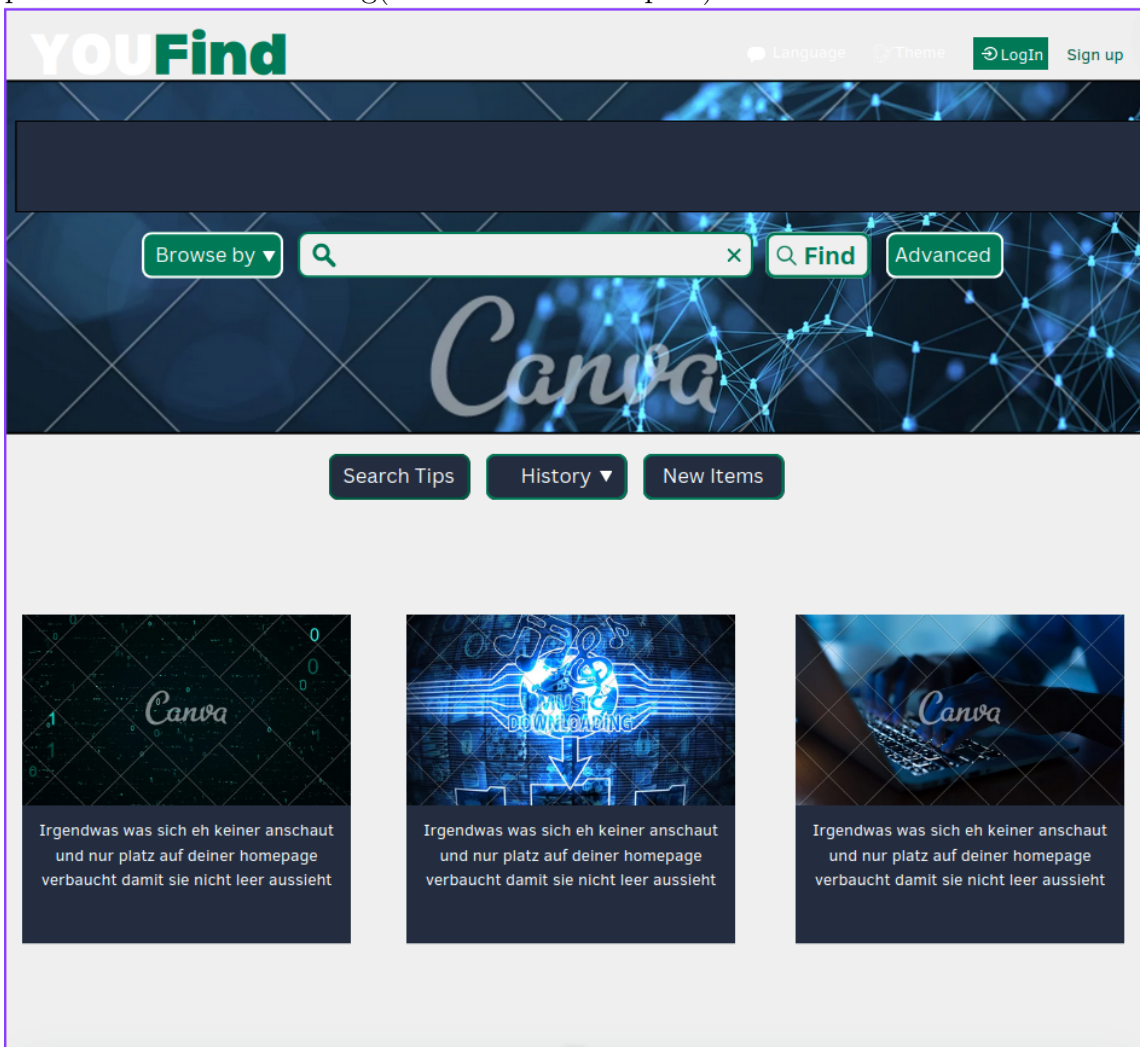

Image 1

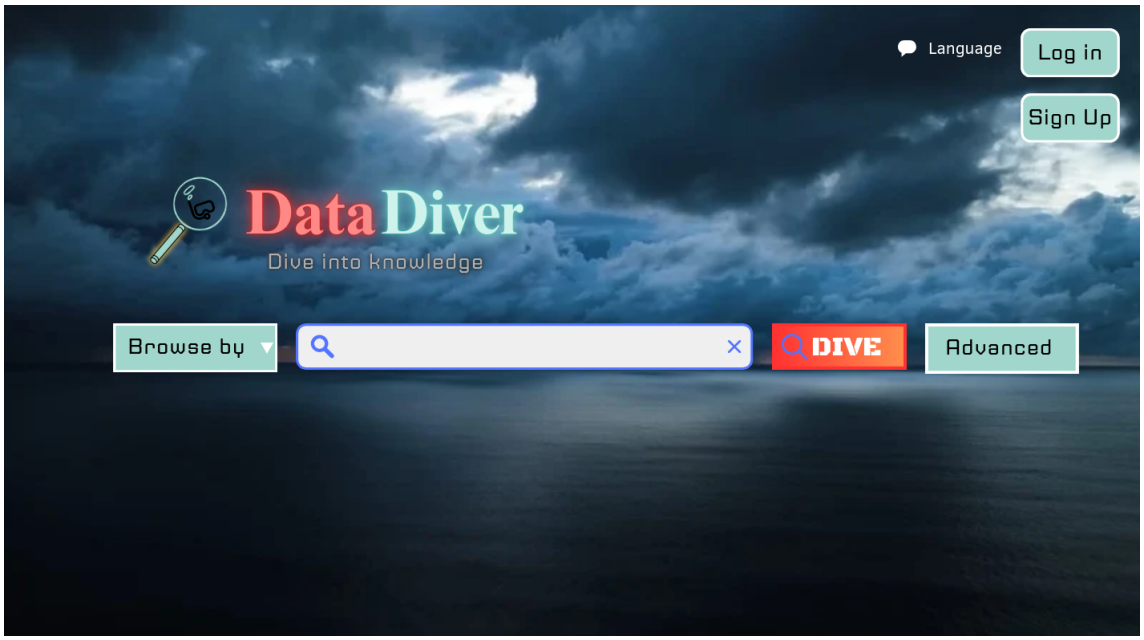

Image 2

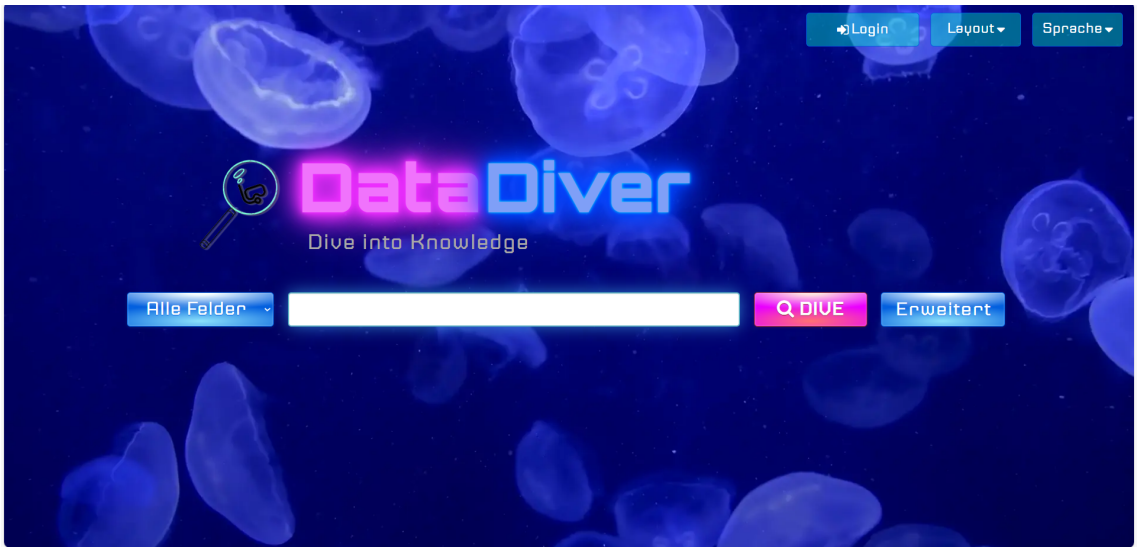

Image 3