



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

*Distributed  
Computing*



# Data Curation Mechanisms for Algorithm Learning

Bachelor's Thesis

Alec Pauli

paulia@ethz.ch

Distributed Computing Group  
Computer Engineering and Networks Laboratory  
ETH Zürich

## **Supervisors:**

Peter Belcák, Benjamin Estermann  
Prof. Dr. Roger Wattenhofer

July 25, 2022

# Abstract

Data Curation is one of the essential bases of machine learning. Moreover, while image or speech tasks are implementable quite well via current machine learning models, the same success remains hidden for highly structured data such as tasks on code. One fundamental reason is the lack of large enough and suitable curated datasets. Therefore, in this thesis, I wanted to develop a web-based tool that resolves the problem of efficiently and quickly curating large datasets. To solve this task, I present a webpage that allows efficient and easy curation of such large datasets.

**Keywords:** Data Curation, Data Curation website, machine learning, machine learning on code, machine learning on highly structured data, prevention of SQL injection, Aquarium, Lightly

# Contents

<b>Abstract</b>	<b>i</b>
<b>1 Preface</b>	<b>1</b>
1.1 My Motivation . . . . .	1
<b>2 Introduction</b>	<b>2</b>
2.1 Data Acquisition and Curation . . . . .	2
2.2 State of the technology . . . . .	4
2.2.1 Aquarium Learning . . . . .	4
2.2.2 Lightly . . . . .	5
2.2.3 Data Curation for Code . . . . .	5
2.3 Code Search Net . . . . .	6
<b>3 The developed application</b>	<b>8</b>
3.1 Overview over the application . . . . .	8
3.1.1 The entry point . . . . .	8
3.1.2 The progress view . . . . .	9
3.1.3 The upload view . . . . .	10
3.1.4 The detailed data view . . . . .	12
3.1.5 The preselect page . . . . .	14
3.1.6 The table view . . . . .	16
3.1.7 The settings view . . . . .	17
3.1.8 The add user view . . . . .	18
3.1.9 The add group view . . . . .	19
3.1.10 The stat user view . . . . .	20
3.1.11 The privileges of the user . . . . .	21
3.1.12 The privileges of the groups . . . . .	22
3.2 State diagram . . . . .	22

<b>4</b>	<b>Technology Stack used</b>	<b>24</b>
4.1	General Setup . . . . .	24
4.2	Concrete Setup . . . . .	24
4.2.1	Setup in Visual Studio Code . . . . .	24
4.2.2	Get started with adding functionality . . . . .	24
4.2.3	Setup of PostgreSQL . . . . .	25
4.3	Concrete setup of the database . . . . .	25
4.3.1	users . . . . .	25
4.3.2	upload . . . . .	25
4.3.3	uploadeddata[unique id] . . . . .	26
4.3.4	actionstab . . . . .	26
4.3.5	groups . . . . .	27
4.3.6	grouptable . . . . .	27
4.3.7	useraccess . . . . .	27
4.3.8	usingroup . . . . .	27
4.3.9	stattab . . . . .	28
4.4	The JavaScript implementation of ordering in the details view . . . . .	29
4.5	SQL Injection and the testing . . . . .	30
4.5.1	What is SQL Injection . . . . .	30
4.5.2	How do I find SQL Injection vulnerabilities ? . . . . .	30
4.5.3	Used tool . . . . .	31
4.5.4	Results of Model 1 . . . . .	31
4.5.5	Results of Model 2 . . . . .	31
<b>5</b>	<b>Evaluation</b>	<b>33</b>
5.1	Further Improvements . . . . .	33
5.2	Closing Words . . . . .	33
	<b>Bibliography</b>	<b>34</b>

# Preface

---

## 1.1 My Motivation

I am fascinated by the possibility that human-made machines can learn. Also, the philosophical questions that arise from that are purely outstanding and lead to understanding basic human processes better. We also need to evolve the model of what a human is and how we differentiate ourselves from learning machines as the space of machine learning evolves. However, as I am so fascinated by the whole area, I wanted to do a Bachelor's thesis in machine learning. Machine learning is not only the application of mathematical models to data. It is also about gathering high-quality and vast amounts of data sets to which the algorithms are applied. It is no coincidence that the ones with the best performing machine learning models primarily also train on the most complete data sets. Furthermore, it is a symbiosis of the data with the abstraction algorithms applied. Although there are unsupervised machine learning techniques, supervised learning is always the way to go if we have high-quality data available and superior to unsupervised learning. That is mainly because, in unsupervised models, we try to increase the probability of labels. Thus, we maximize the possibility that a particular label is really of a given class. However, in unsupervised learning, a possibility of 1 or 100 percent cannot be reached. Therefore this information gain can be maximized if we can determine the actual label of the given data. Therefore, the situation is also in the future improbable to change. That reasons get me so excited about the work of efficiently curating data. It is basically at the foundations of machine learning and will most likely remain or even increase its importance.

# Introduction

---

## 2.1 Data Acquisition and Curation

As stated previously, Data Acquisition and Curation are essential components in the machine learning pipeline and have some pitfalls and points to watch out for. In the following, I want to give a short overview over them, and later on, I also want to analyse which and how these issues are addressed by the developed data curation tool. First, let us look at the ultimate and fundamental goal of Data Curation: the structuring and labelling of data. The structuring and labelling are done in order to prevent data swamps. Data swamps are the unstructured gathering of data that cannot be used in its raw format without 'cleaning' it up. Problems of such a swamp can be, for example:

- **Bias:** Already in the underlying data, potential biases should be eliminated
- **Inaccurate or falsely represented**
- **Ambiguity**

Generally, a given Data Swamp should go through multiple steps of preprocessing to generate a good and reliable data set used for training machine learning models. Generalised, these steps include the following:

- **Formatting:** Bringing the datasets to a general representation and merging them, if scattered, into one large dataset
- **Labelling:** Annotate the label to the data fields for the intended purpose. Thus, for example, when using the MNIST dataset to classify images by their content, the labels would be the numbers from 0-9
- **Cleansing:** Suboptimal parts, such as little errors, are removed
- **Extraction:** Features are extracted for optimization. For example, if we consider the classification of chemical molecules, extracting a chemical fingerprint that already models some features is advantageous and can reduce

the overall model complexity of the machine learning algorithm considerably

The following presented and in this thesis developed data curation tool mainly helps in the labelling and cleansing step. Also, it is helpful in the merge step and the formatting step.

## 2.2 State of the technology

This subsection will analyse the already existing data curation solutions and their strengths and weaknesses.

### 2.2.1 Aquarium Learning

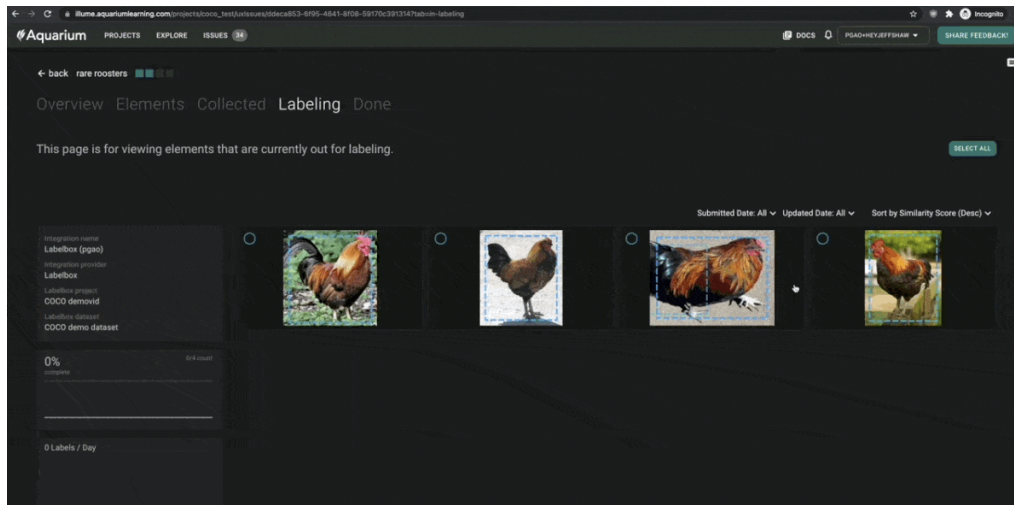


Figure 2.1: The aquarium Data Curation Tool website.

Aquarium Figure 2.1 is a Data Curation tool mainly focused on the curation and labelling of images. Although it is also possible to edit text, it is pretty unadapted for text or, more significantly, for program code curation. Its main advantage is tight integration into other ML pipelines. However, this can get a disadvantage too if the used methods are not supported by aquarium. Also, aquarium supports in their base model 'only' 100'000 data points. For each additional 100'000 data points, they charge 800 USD more per month. Thus, creating and curating large datasets that can take multiple months or even years until they are ready is hard to finance without serious funding. Also, with already a base price of 12000 USD per year for five users.



## 2.2.2 Lightly

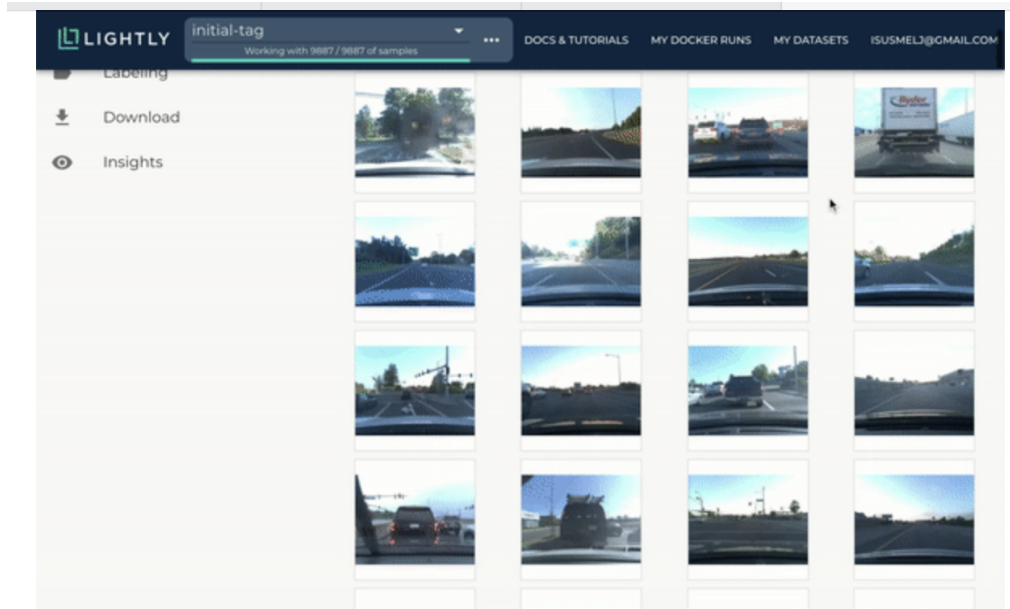


Figure 2.2: The Lightly Data Creation Tool website.

Also, Lightly's fig. 2.2 (a Zürich-based software company) main strength lies in Image Data Curation. Both tools seem to be developed with it in mind and later expanded to some text curation functionality but also here, code edit ability is relatively poor.

## 2.2.3 Data Curation for Code

Most Data Curation tools focus on image, audio or video curation. Some have a curation process for text, but this seems mostly left behind. Program Code seems to be a niche, and no Software is yet fully tailored to classify or labelling code. Although this development is not astounding, as the areas of image and audio are also the spaces where deep learning techniques are good applicable, we also want to create datasets, especially for areas where deep learning models are not yet further advanced, to enable research in the respective areas. Moreover, this is not possible without a high-quality underlying dataset.

## 2.3 Code Search Net

Since the idea of focusing on Code Curation was motivated by the paper from Husain et al. [1], I want to give a short summary of the paper in the following. The paper's authors motivated that no relevant code datasets with their respective natural language queries are available.

In the paper, they have a look at semantic code search. Semantic code search involves getting relevant code from natural language queries. It is also important to note that Deep Learning techniques are used everywhere but struggle to be useful for highly structured data such as code. Here, standard information retrieval techniques do not work because of the lack of shared vocabulary. That is also why the area of machine learning on highly structured data is of high research interest. Furthermore, the authors created the CodeSearchNet Corpus (with about 6 Million functions) and a Code Search Net Challenge. The Corpus is an expert-created dataset. The Corpus dataset was created by automatically fetching GitHub and pairing code with documentation. The necessary preprocessing steps taken for the Corpus dataset were:

1. The projects are ranked according to their popularity and licence.
2. Then consider only those methods documented  $\rightarrow$  Tuple (code, documentation)
3. The documentation is shortened to the first entire paragraph
4. Too short documentations are removed
5. test functions and defaults are removed
6. Duplicates/near duplicates are removed

The created dataset is also noisy because of different languages or forms of languages and outdated commands.

The Code Search Net Challenge consists of 99 natural language queries with about 4000 expert annotations of likely results. It is built on top of the Corpus dataset. To ensure the representatives of the natural code queries, they got common code search queries from Bing and ranked them according to the click-through rate and StaQC [2]. Many problems were incurred when fetching the code. In the following, the most imported ones are listed:

- Although the code comes from popular repositories, the code quality can be low. (for example, bad security practice, readability, slow or hard to parallelize)
- Natural Language ambiguity that is hard to parse

- Library and Project specific code is hard to filter out but of little interest in the query search
- Directionality: It was hard to differentiate, for example, the natural language queries from string to int or from int to string

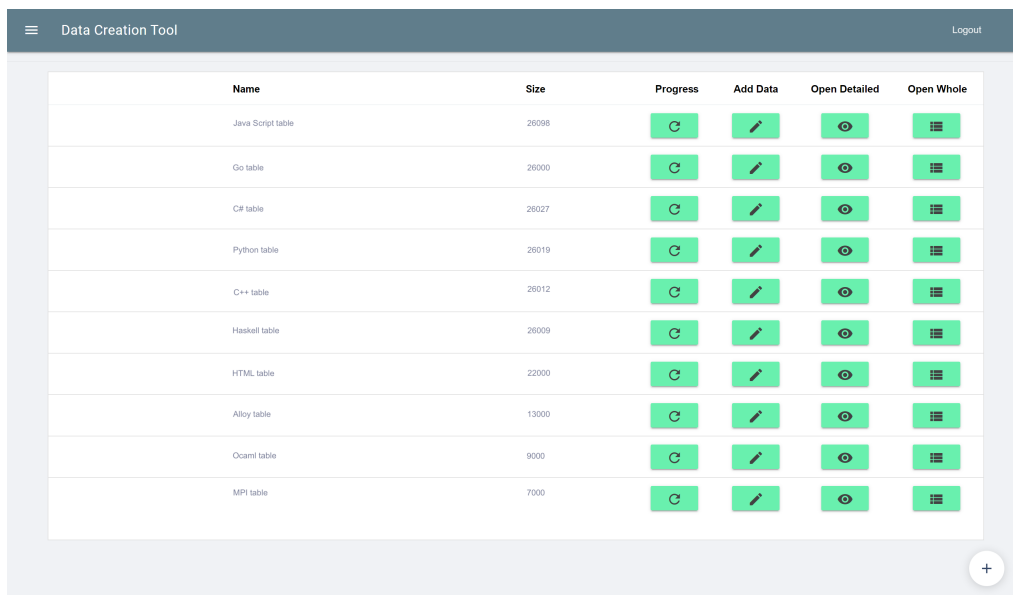
We can create a baseline with different models for the Code Search Net Challenge. The baseline was created with a 90/10/10 train-test-validate split. The models used in the created baseline are 'bag of words', RNNs, CNNs, and attentional models. This baseline is then used to facilitate the expert annotations. In the evaluations done in the study, the self-attention model performed best for the Code Search Challenge. Nevertheless, code semantics cannot be exploited by existing methods.

# The developed application

---

## 3.1 Overview over the application

### 3.1.1 The entry point



The screenshot shows the 'Data Creation Tool' interface. At the top, there is a dark blue header with a hamburger menu icon on the left, the text 'Data Creation Tool' in the center, and a 'Logout' link on the right. Below the header is a table with the following columns: 'Name', 'Size', 'Progress', 'Add Data', 'Open Detailed', and 'Open Whole'. The table contains ten rows of data, each representing a different dataset. Each row has a green button with a circular arrow icon for 'Progress', a green button with a pencil icon for 'Add Data', a green button with an eye icon for 'Open Detailed', and a green button with a list icon for 'Open Whole'. At the bottom right of the table area, there is a small white circle with a plus sign inside.

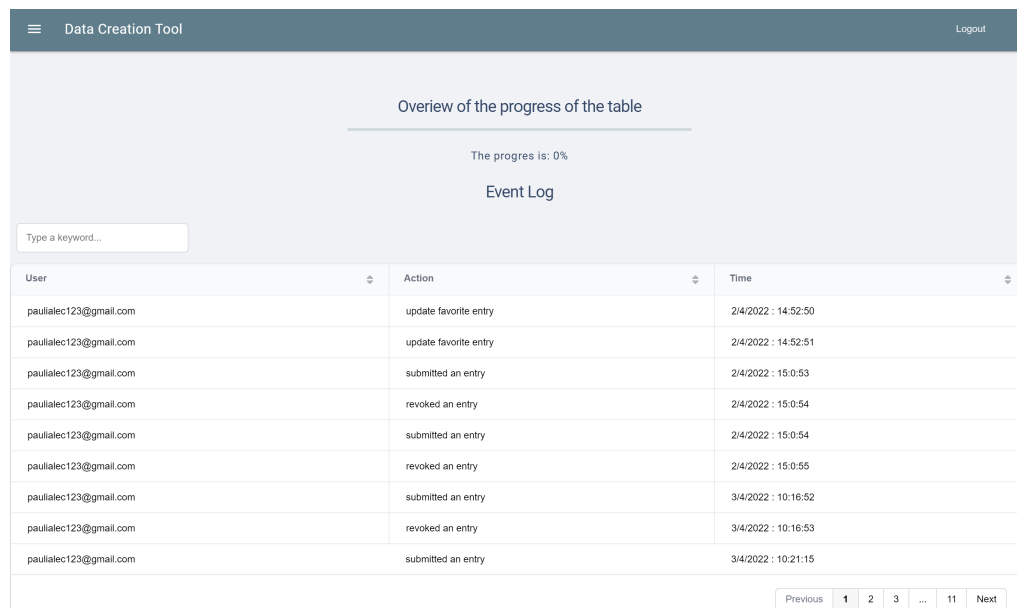
Name	Size	Progress	Add Data	Open Detailed	Open Whole
Java Script table	26098				
Go table	26000				
C# table	26027				
Python table	26019				
C++ table	26012				
Haskell table	26009				
HTML table	22000				
Alloy table	13000				
Ocaml table	9000				
MPI table	7000				

Figure 3.1: The entry point of the Curation program

When starting the application, the user gets an overview of all datasets attributed to him where he has either a read, a write, or full permission. He can quickly see the size of the datasets and has the option to start curating with a random sample out of the yet undone samples. Alternatively, the user can get a more fine-tuned overview of the dataset where the user can search or filter according to other criteria. Also, the user can access statistics of the curation process or add

data to the given dataset. In the menu bar, the user can jump to the permissions settings where he can manage his credentials or, depending on the 'privilege' mode, manage the access to datasets, the users of the groups and many other things. See the following sections, for example: 3.1.11 for further information concerning the user privileges and dataset access modes.

### 3.1.2 The progress view



User	Action	Time
paulialec123@gmail.com	update favorite entry	2/4/2022 : 14:52:50
paulialec123@gmail.com	update favorite entry	2/4/2022 : 14:52:51
paulialec123@gmail.com	submitted an entry	2/4/2022 : 15:0:53
paulialec123@gmail.com	revoked an entry	2/4/2022 : 15:0:54
paulialec123@gmail.com	submitted an entry	2/4/2022 : 15:0:54
paulialec123@gmail.com	revoked an entry	2/4/2022 : 15:0:55
paulialec123@gmail.com	submitted an entry	3/4/2022 : 10:16:52
paulialec123@gmail.com	revoked an entry	3/4/2022 : 10:16:53
paulialec123@gmail.com	submitted an entry	3/4/2022 : 10:21:15

Figure 3.2: The 'progress' view of the Curation program

This view displays a per dataset progress. Each action of all users that can do actions on the dataset is logged and presented in the table. Also, the overall progress of curating the dataset can be found.

### 3.1.3 The upload view

Figure 3.3: The 'upload' view of the Curation program

The upload view is used two times, once if a new data set is created and a second time if data is added. Here, a JSON file can be uploaded, and each field can be assigned a type. For example, it is also possible to upload images in Base64 encoding besides the basic types like code and text. In order to avoid repetitions in the uploaded dataset, the repeating part of the Base64 String ('data: image/png;base64') should not be included. With that, memory can be used even more efficiently, especially in datasets with extensive image collections. For hand designing datasets, the web page mentioned in the Bibliography [3], where images can be converted to Base64 images, comes in handy. For automatically creating datasets, for example, the following C# code [4] comes in handy.

```
using (Image image = Image.FromFile(Path))
{
    using (MemoryStream m = new MemoryStream())
    {
        image.Save(m, image.RawFormat);
        byte[] imageBytes = m.ToArray();

        // Convert byte[] to Base64 String
        string base64String = Convert.ToBase64String(imageBytes);
        return base64String;
    }
}
```

Alternatively, as for creating large structured datasets primarily, Python is used. Here is also a short python snippet [5] to create Base64 strings out of images.

```
import base64

with open('sampleimage.png', 'rb') as img_file:
    b64_string = base64.b64encode(img_file.read())
print(b64_string)
```

As in code or images, often we have some special characters, a general escaping technique with the possibility to add the character: `'''` at the beginning and the end was introduced. The advantage is that large code blocks and other long entries can quickly be escaped. Then after implementing the above design choices, I wanted to add a part of the CodeSearchNet dataset as it was one of the primary motivations to develop the Data Curation tool. However, the format of the Dataset was JSONL and not JSON. JSONL[6] is a format where each line is a valid JSON format. However, it was not compatible with the current implementation of the upload functionality. Therefore, I needed to generate a conversion mechanism. I leveraged the already underlying JSON processing and added the functionality to automatically convert JSONL files to JSON files. That gives the advantage that the user can import its file without inconvenience. However, only one interface is used in connection to the back end. Therefore, potential errors could be debugged quickly and resolved therefore faster. As for convenience, the second conversion of file type from CSV to JSON was added. It can also be expanded to more file types. These conversions happen on the client side, which is an additional advantage of this design choice since, on the back-end, a single optimized gateway can be maintained.

### 3.1.4 The detailed data view

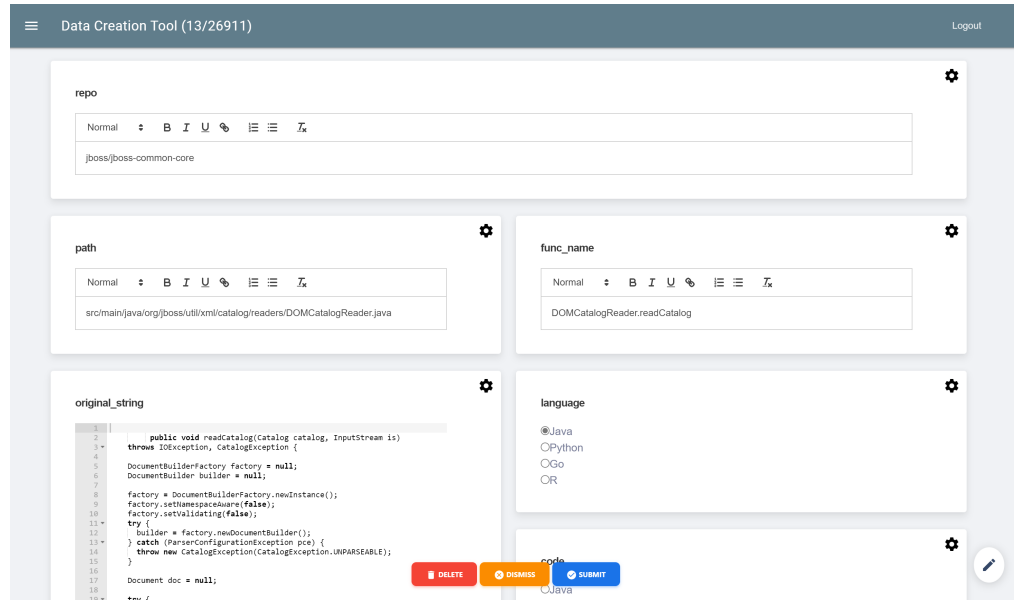


Figure 3.4: The 'details' view of the Curation program

This page is the most important one of the application. Of course, in the end, it is a combination of all different views, but here, since the curation process is mainly done in this view, it is of major importance to guarantee high customization. That is particularly hard since the data cannot be predetermined since the user can add its own data sets with variable size and order. For that, various customization options were added, such as changing the order of the elements leaving some out and changing the sizes, for example. It is also of major concern to present data compactly and cleanly. For that, typically, frameworks like Bulma [7] or Flexbox [8] are particularly suitable. However, both had major downsides that reduced efficiency in data curation. First, Bulma interpreted each row and ordered all elements according to the biggest element in the row. See fig. 3.5 for an example of the problem. For Flexbox, which is also usable out of Bulma, there is an automatic ordering algorithm. This algorithm has, at least for our usage, one disadvantage that cannot be overlooked. The elements cannot be in a fixed order. It just minimizes the overall 'lost' space.



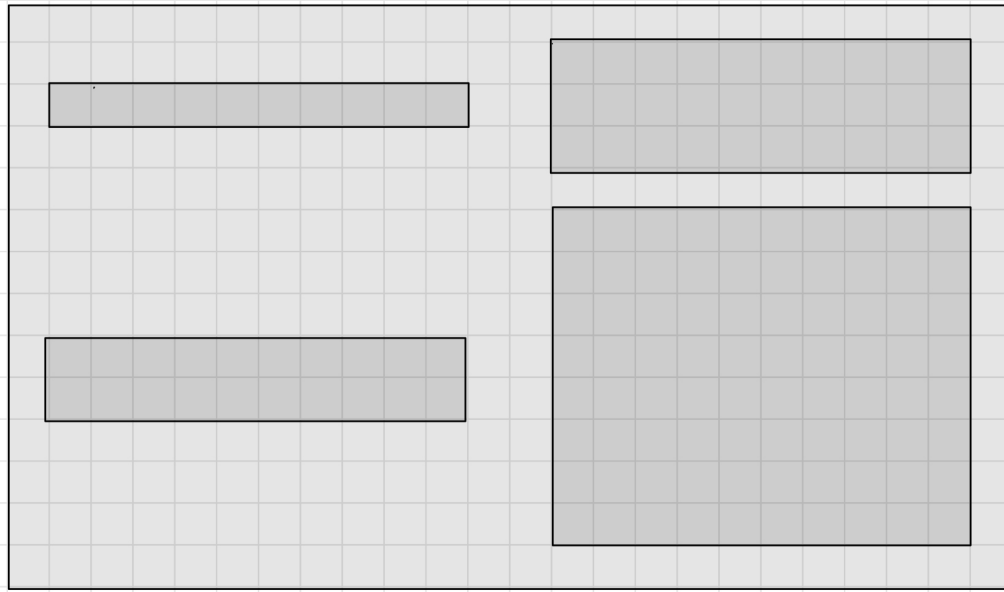


Figure 3.5: An illustration of the problem of considering all objects in a row as connected

Therefore, especially with the wish to adjust ordering, this approach could not fulfil the needs. A third approach was the idea to use the jQuery draggable and resizable attribute and give the user complete control over their wished design. Nevertheless, after using this approach, the downside of changing the size of the attributes was quickly discovered. No user design could fulfil both overview-ability and, at the same time, fit all different sizes of attributes that are changing from sample to sample. Therefore, after implementing the Bulma and the jQuery approach, the final option to dynamically customize the design each time via JavaScript was chosen. With that approach, finally, the freedom was given to customize the design as wanted fully. The JavaScript function just each time computes based on the screen size and the wished parameters from the user, such as ordering or size the optimal fitting solution. With that, the downside of the Bulma framework is overcome, and the productivity of the detailed view is increased. Also, to further increase productivity, it was observed that many entries that need to be changed are out of a small set of values. Therefore, a 'quick' choice option was added. This option allows for each field to quickly choose predefined custom options. To not have to change the value when we are satisfied, I added the auto-selection of the feature. The auto-selection means that the given entry is compared to the user-defined discrete set of options, and the most likely one is chosen. Also, with the buttons, I tried to focus on usability. Since the full view can be compared to a questionnaire, I was inspired by the following analysis of the positioning of buttons[9]. According to it, the most

intuitive way, or at least the way that reduces the overall time to go through the questions, is to use a horizontal design with next/ forward on the right side. This approach was also chosen with a third option, 'dismiss' I decided to place this button between the other two buttons since the cursor is when going quickly to samples already in that area. Also, it is most likely the least used option. Or at least it was when I tested the design's usability. Therefore, it is best to place it at a prominent location but not at the two corners where the two most used buttons should be located.

### 3.1.5 The preselect page

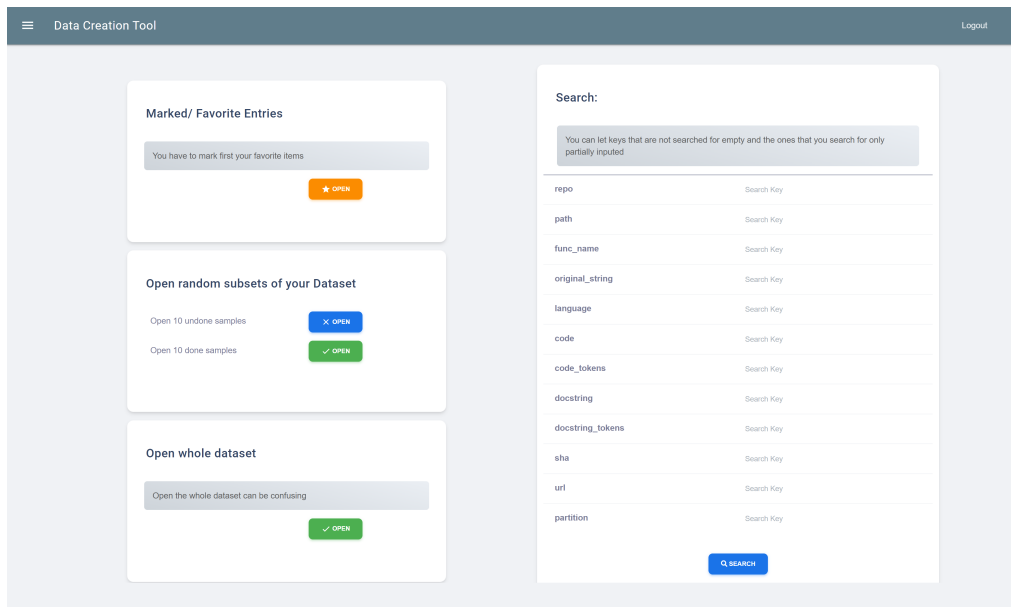


Figure 3.6: The 'preselect' view of the Curation program

This page acts as an intermediary between special entries of the dataset and the user. It has the option to get the favourite entries, previously selected, quickly. Get done entries or undone entries, or search for a wanted attribute in the search functionality. In the following, I attached a short draft of this site before it was developed. This draft was created before I decided to switch to another sidebar design. Therefore the "old" sidebar was still integrated.

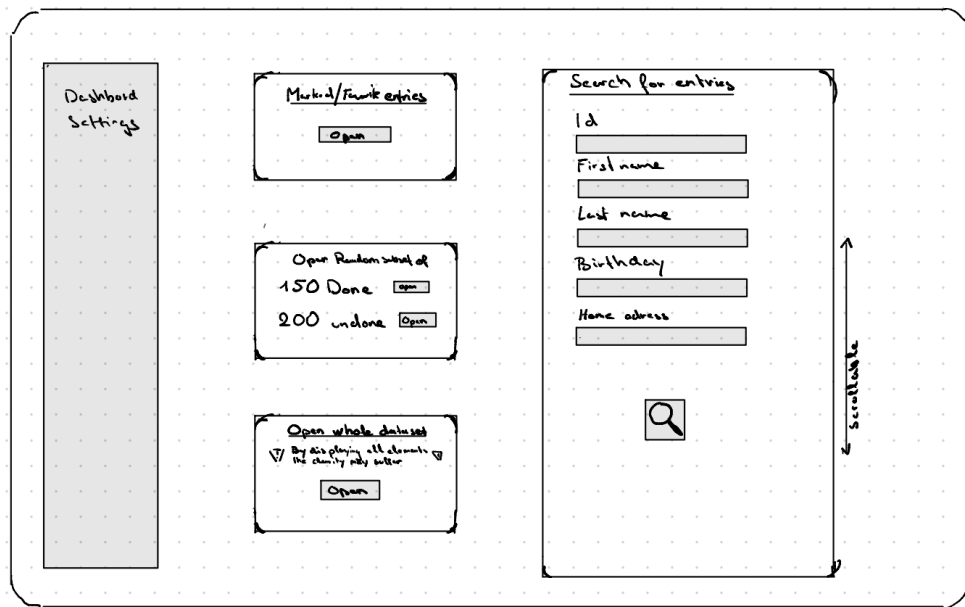


Figure 3.7: The 'preselect sketch' view of the Curation program

### 3.1.6 The table view

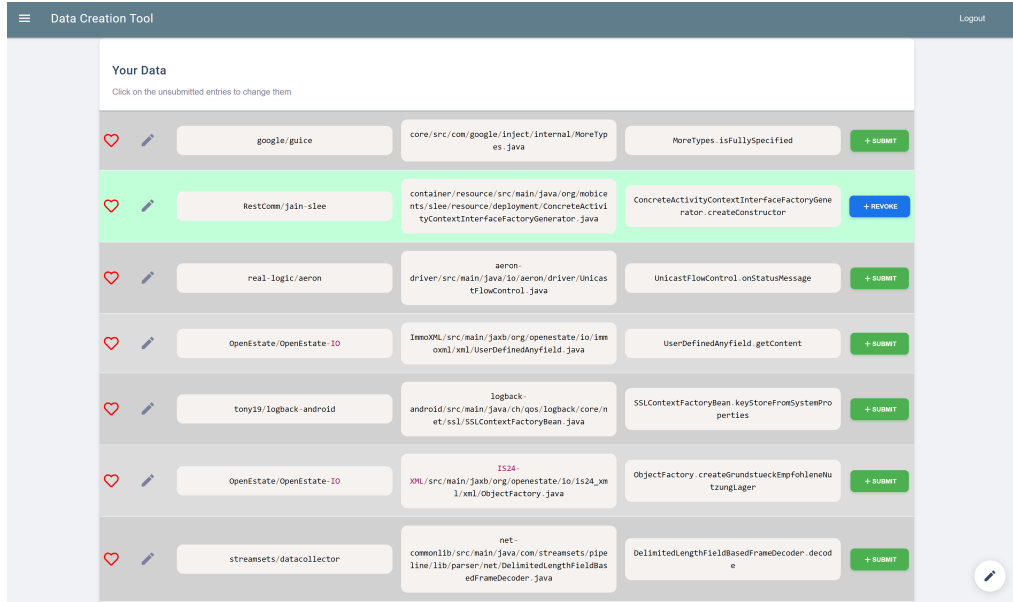


Figure 3.8: The 'table' view of the Curation program

This view gets opened after the prefilter view. In it, the user can quickly submit or revoke entries without actually opening them or interacting with them in other ways. This view primarily aims to facilitate quick interactions if the corresponding entry is correct. To change the fields of the corresponding entry, a quick option to edit the fields is provided. That option loads the entry into the detailed view, where the user can edit the entry as usual. Furthermore, it is possible to mark specific entries as favourites for faster access in the future. Another feature included here is, as in the detailed view, the possibility to limit the table to specific entries and shorten certain long fields to increase overview ability. That increase in overview ability comes especially handy if we have a dataset with a large number of fields.

### 3.1.7 The settings view

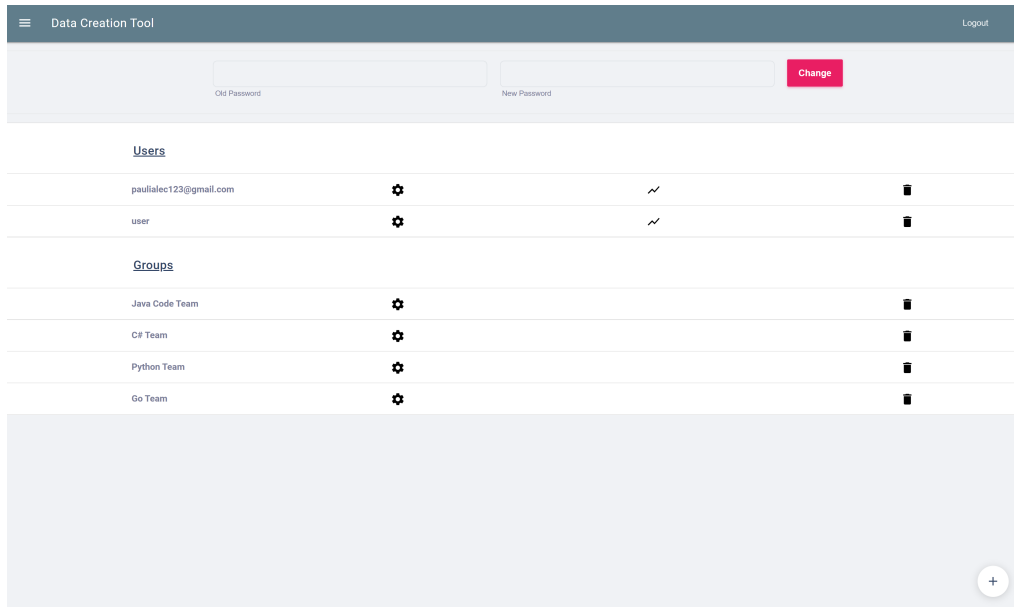


Figure 3.9: The 'settings' view of the Curation program

Foremost, here the user password can be updated. Also, there are more functionalities for admin users. They are able to add groups or users. A corresponding modal is opened through the add button in the right bottom corner. Also, they are shown an overview of all groups and users in the system. From there on, each user can be assigned individual privileges for each data set or via groups a whole list of users can be assigned a group of privileges. For each user, there is additionally the possibility to get detailed statistics on his or her usage of the application and how he or she interacts with the datasets.

### 3.1.8 The add user view

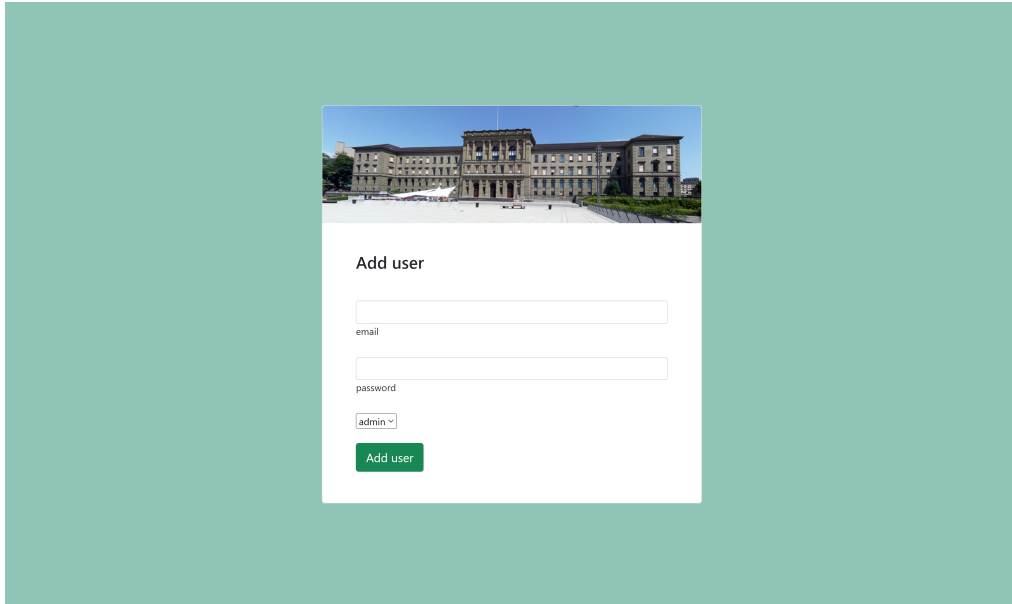


Figure 3.10: The 'add user' view of the Curation program

Here, a user with the respecting role can be added. The roles of admin and a regular user are available.

### 3.1.9 The add group view

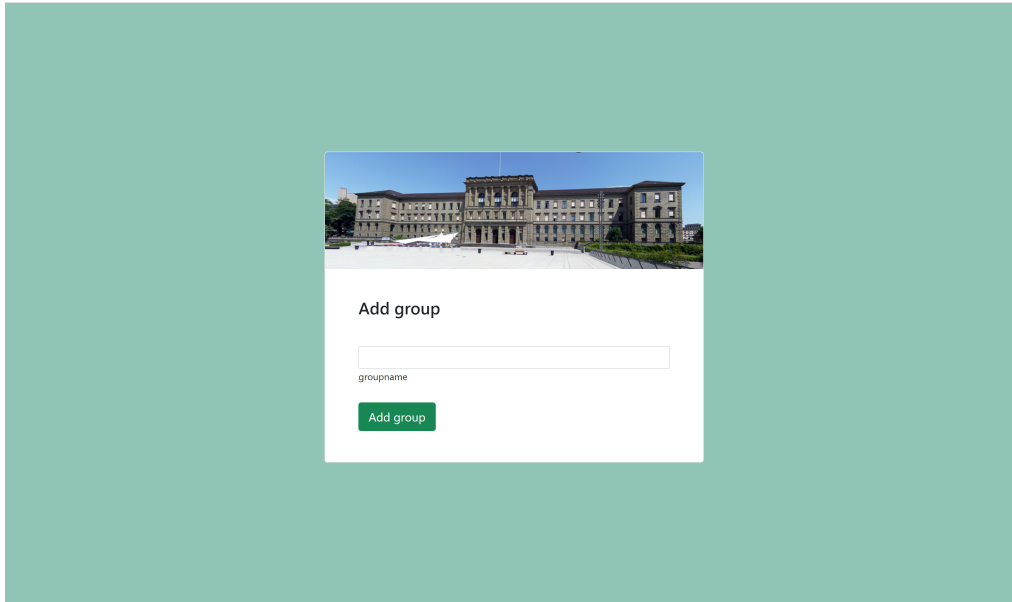


Figure 3.11: The 'add group' view of the Curation program

Here, a group can be created. For then creating the access rights and the corresponding rights on the dataset in the background, the following SQL query is used (in the following, I added the writeable check):

```
Select * FROM upload INNER JOIN ((Select useraccess.tableid,
useraccess.userid FROM useraccess where (state = 'Wd' or
state ='Writeable' ) and userid='" + id + "' )
UNION (SELECT grouptable.tableid, usingroup.userid
FROM grouptable INNER JOIN usingroup
ON usingroup.groupid = grouptable.groupid where
usingroup.state = 'True'
and (grouptable.state = 'Wd' or grouptable.state = 'Writeable')
and userid='" + id + "')) AS cr ON cr.tableid =
cast ( upload.id as text);
```

This query generates for each user the list for which tables she or he has the writeable permission. Permission is given if the user has either direct permission via the user view or the user is in a group where the corresponding group has the ability to do this action. The highest permission gets extracted in the back end for each API call applied in a second query that uses these results.

### 3.1.10 The stat user view

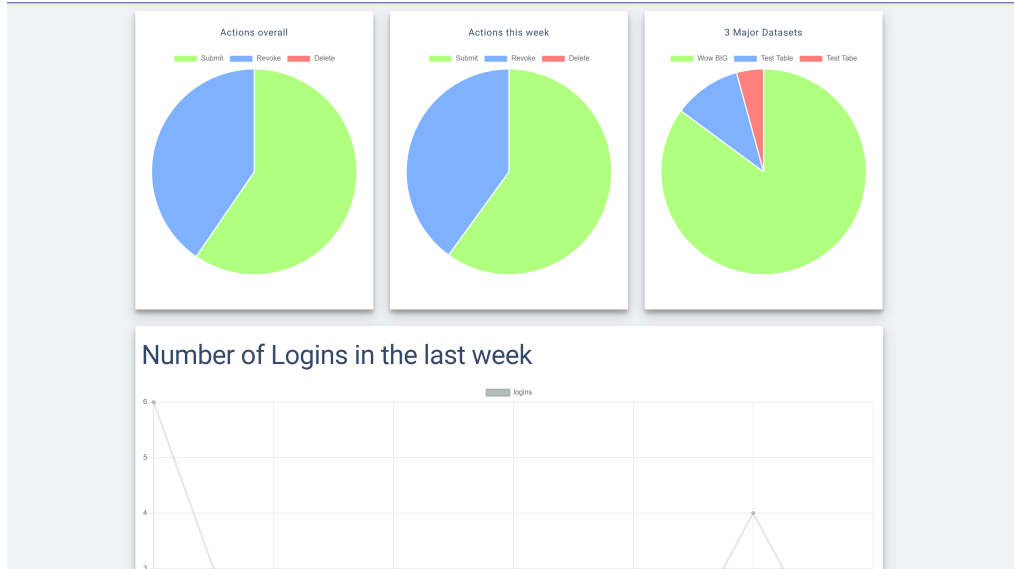


Figure 3.12: The 'stat' view of the Curation program

Here, a detailed analysis of the user's interaction with the datasets is given. The last logins are recorded and graphically presented. We also have three informative pie charts showing the datasets with which the user most interacted. The first pie chart shows the actions over all datasets. The following pie chart limits the time to the last week. (Here, last week is always the last seven days). Moreover, the third pie chart shows the three datasets with whom the user interacted most.



### 3.1.11 The privileges of the user

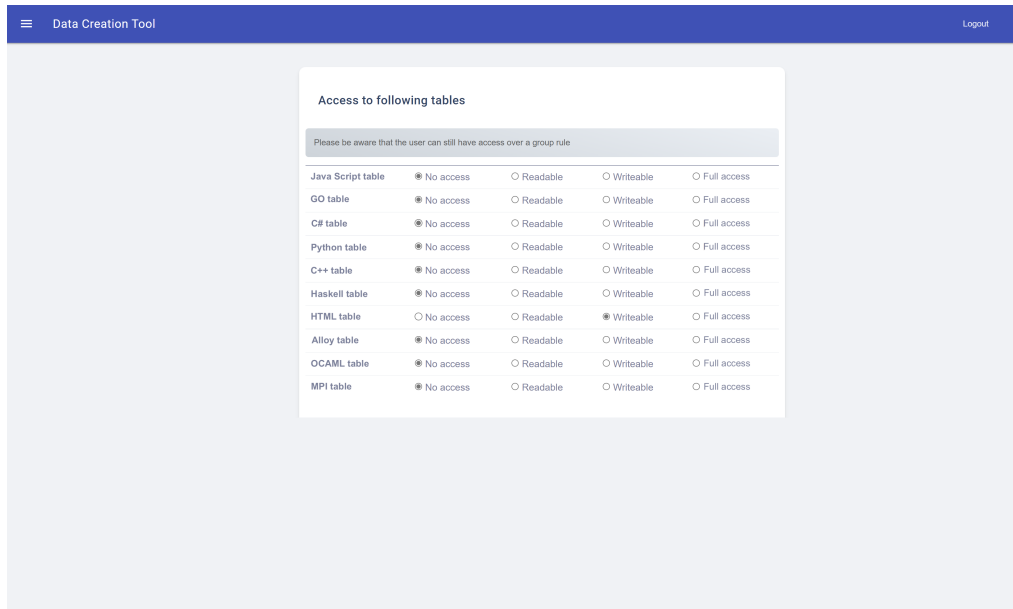


Figure 3.13: The 'user privileges' view of the Curation program

Here, each user can be assigned one out of 4 states: Readable, Writable, Full access and No Access. As the corresponding names say, Readable means that only read actions can be performed, and no changes can be performed. Especially also no submissions. Writable gives the user the right to change entries to submit them or revoke them, but she or he cannot add or delete entries. With full access, the user can basically do the same as an admin on the particular dataset. Moreover, with No Access, the user cannot access the dataset, the name or the size.

### 3.1.12 The privileges of the groups

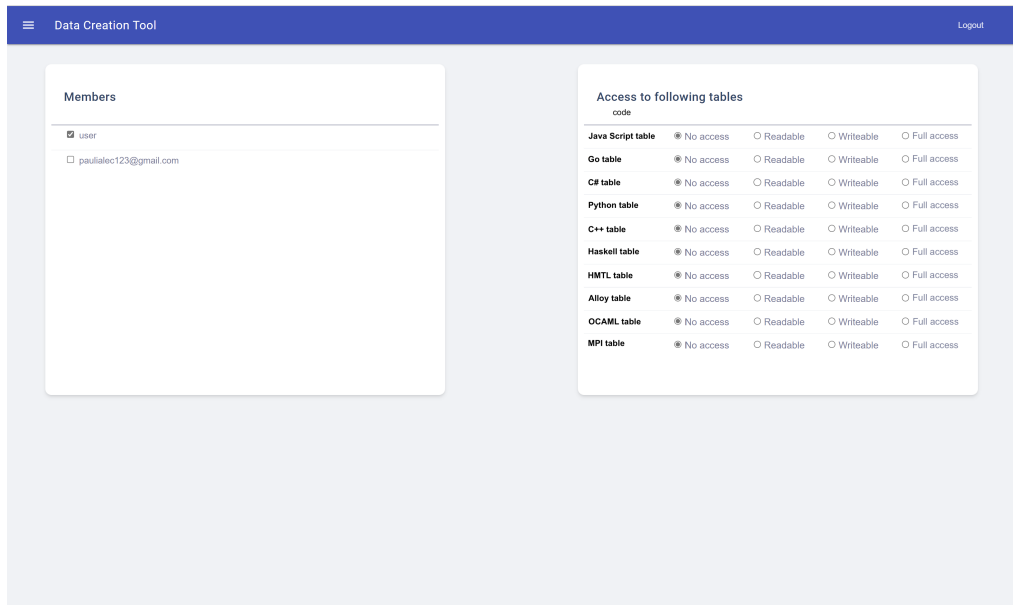
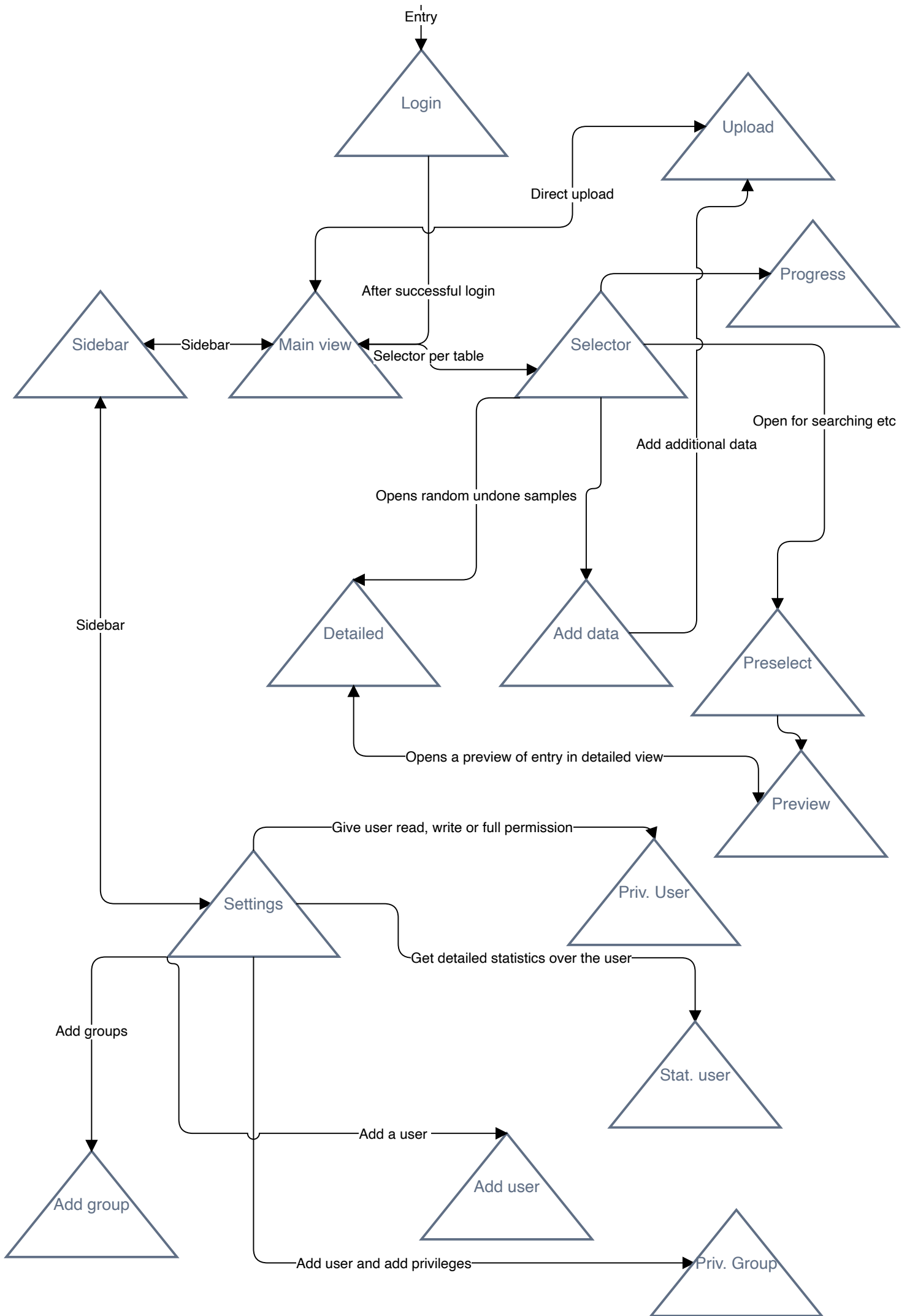


Figure 3.14: The 'group privileges' view of the Curation program

This view consists of two sub-parts. One has a complete list of the users on the system and lets us easily add or remove users from the selected group. Then on the other side, we can decide on which dataset which of the 4 (same as in privileges of the user) privileges is assigned.

## 3.2 State diagram

On the following page, I created a state diagram of the application. Some trivial parts were not added in the interest of overview ability. The given diagram should help to get a basic overview.



# Technology Stack used

---

## 4.1 General Setup

I used a stack of PostgreSQL, Node.js and multiple front-end packages such as Material Design Lite [10], Ace [11], Bootstrap[12],Quill [13], Bulma[7] and Axios [14]. For the development process, I used Visual Studio Code and Postman. The web page is tested in Edge (Chromium-based), Google Chrome and Firefox. With that, most of the major browsers are tested. For PostgreSQL, I used Version 14.3. For Edge, I tested it with Version 103.0.1264.44 . For Chrome, I tested it with Version 102.0 . For Postman, I used Version 9.23.0 .

## 4.2 Concrete Setup

### 4.2.1 Setup in Visual Studio Code

Moreover, as soon as the project is imported into Visual Studio Code and all defined dependencies of the project are loaded, the project can be started by inputting 'npm start' (created shortcut for nodemon index.js) into the console. Afterwards, it can be accessed via localhost:5000 in any web browser.

### 4.2.2 Get started with adding functionality

When starting to do modifications, please first see the README.md and the corresponding linked commands in the files. There I have added a programmatic-specific explanation to get started modifying the code. But generally, as a short introduction, it can be said the following: As an entry point, the index.js file is used. In it, the corresponding API calls are handled with the help of supporting classes. Then all data is written and accessed via the postgres.js class. The front-end primarily consists of an HTML, JavaScript and a CSS file.

### 4.2.3 Setup of PostgreSQL

A user 'me' needs to be generated for setting up the back-end database. The default password is 'antenne'. The default host is 'localhost'. The default port is 1919, and the default database is 'web'. This 'default' can also be changed on demand in postgres.js, where they are accessible right at the top. Further, the table users, upload, groups, useraccess, usingroup, grouptable, actionstab, and stattab have to be created. Pre-written queries for that are listed in testsql.txt

## 4.3 Concrete setup of the database

The webpage was extended to support file sizes up to 5 GB. This extension is done in the index.js file and can easily be increased by increasing the corresponding parameter. All data sets larger than the corresponding upper limit must be uploaded in multiple uploads. This can be done via the add data functionality. The program uses the following tables in the back end:

### 4.3.1 users

This table consists of the attribute's id, email, password, role, and parent. Where email is the username that is most likely the email, as the email is often the chosen username; therefore, it was called email. Id is a unique id for this user. Then we have the password. The password is stored in a sha256 hash format to ensure security. By the way, it is also only transmitted in this format. The role is either that of an admin or a user and last but not least, the parent is the user who created the user.

id	email	password	role	parent
37	example@ethz.ch	497a.....65	admin	none
38	example2@ethz.ch	497a.....65	noadmin	example@ethz.ch

### 4.3.2 upload

This table stores each upload that is done. It has the following table entries: id, which is, as usual, just a unique id. Name is a uniquely created internal name that is not used in the front end. It facilitates the join operation via easier identification of corresponding entries and dismisses the problem of the same names as public names. Next, an entry key is added, containing a JSON object of the fields titles and their corresponding type. This JSON can be used to determine the type quickly and, in some instances, only query this sub-view of the data. Then we have the entry users, which hold the name of the creating user. In the following field, we have the publicname, which is the name that the

user gave the table. And then, we have also a count that holds the number of entries in the dataset.

id	name	keys	users	public name	count
1	u.dat....211264	{program:code}	example@ethz.ch	Java	23000
2	u.dat ...211223	{image:image}	example@ethz.ch	Java	55000

### 4.3.3 uploadeddata[unique id]

Because of performance reasons, it makes sense to encode a dataset directly into a PostgreSQL dataset. The problem that arises here is that a standard approach with setting parameterized values into a query does not work. However, this is the standard approach for PostgreSQL, especially with regard to the danger of SQL Injection. But the only approach to efficiently encode the database queries goes over dynamically on demand creating the query. Because otherwise, we could not leverage the very efficient implementation of the underlying database since the alternative would be to encode all data entries in one field. Therefore, I created a dynamic query that then gets later parsed. To prevent SQL insertion, I used the newly added and sadly still undocumented escaper functions of PostgreSQL. To test and evaluate the security, especially of these queries, I added later a section that evaluates the security of these queries. So to explain the effect of the given program code, here is a short example: Let us assume the fields that get uploaded are *Program code*, *what the code does* and *visualization of the code via code flow* then, the corresponding SQL table directly corresponds to these values. It would look like this (For reasons of clarity, the field 'id' is omitted here):

Program code	what the code does	visual. of the code via code flow
some program code 1	some explanation 1	some base64 value
some program code 2	some explanation 2	some base64 value

### 4.3.4 actionstab

In this table, all actions on the tables get recorded. We again record an id, a username that is the name of the user that initiated the action, a tablename that is the id of the table where the action is performed and the action and the time it was done. Again here is a short mock database to visualize the entries:

id	username	tablename	action	time
1	example@ethz.ch	3	submitted an entry	13.07.2022
2	example@ethz.ch	3	revoked an entry	13.07.2022

### 4.3.5 groups

This table contains all groups in the system. We have an id, then a name field which identifies the name of the group and a field show that determines whether the group is active or not. A short visualization is:

id	name	show
1	Java curation	true
2	Python curation	true

### 4.3.6 grouptable

Then we have a corresponding table for managing the access rights of the different groups. It is composed of an id as in each database, then the groupid that identifies the group, and then we have a tableid that identifies the corresponding table. Moreover, most importantly, we have the state that determines the group permissions. A short visualization is:

id	groupid	tableid	state
1	2	1	Writeable
2	2	2	False

### 4.3.7 useraccess

This database determines the access of a single user to the different datasets. It consists of the same basic blocks as the usingroup despite the fact that the information does not need to be joined before it can be accessed; it is like the resulting table after joins in the corresponding tables. It consists of the entries id, tableid, userid and state. They have the same functionality as above.

id	tableid	userid	state
1	23	13	True
2	2	2	Writeable

### 4.3.8 usingroup

And at least in terms of database design, the last piece of the puzzle of user access rights can be added with the database that stores the users that belong to a particular group. This database consists of an id and the groupid that identifies the group. As next, we have the unique userid that identifies the user, and in the last field, we have the state of the belonging.

id	groupid	userid	state
1	2	1	True
2	2	2	False

### 4.3.9 stattab

This table is for creating the statistical overview per user and consists of the following fields: id, username, tablename, action and time. Username is the user's id that created the actions, while the tablename is the unique id of the table the action was performed on. The action is a string that determines the action, and the time is the time of the action in milliseconds. And again, here is a short visualization:

id	username	tablename	action	time
1	23	13	submit	1655745416056
2	2	2	login	1655753725403



## 4.4 The JavaScript implementation of ordering in the details view

So the central part of this algorithm is located in the `order.js` class. The entry point is the `reorderbox` function that takes the arguments: `keys`, `order`, and `numofrender`. The first holds the names of the corresponding boxes previously created. The `order` is a transposition of the standard ordering (natural numbers from 1 to the corresponding length) that was inputted previously via the modal questionnaire and stored locally on the users' device. As next, the `gethandles` function gets pointers to the corresponding boxes, and the size of the view that holds the boxes on the current device is determined. The algorithm has gathered and preprocessed all information to be ready to loop over the boxes and reposition them. This reposition is done in the `position` function in the following way: There are three variables for the 'thirds x' values and three variables for the 'third y' values. Then we have the same again for the two respectively four half variables; thus, we initialize the following values: `thirdx`, `third1x`, `third3x`, `thirdy`, `third1y`, `third2y`, `halfx`, `half1x`, `halfy`, `half1y` and initialize them in the following way that is best explained in the following short drawing:

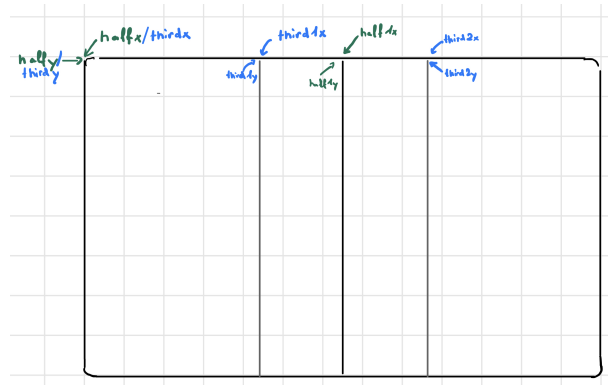


Figure 4.1: The initialization of the fields

Simply put, they mark the top left corner of the respecting size. The finest size here is one-third, and therefore it works only to take a third as the smallest unit. However, in general, if the respecting class should be generalized for even smaller sizes (which would optically have a hard time), then we could simply add more variables. We do not have a full variable in the function since the two half sizes can model it. That is not the case for the half through the thirds. Nevertheless, as a next step, we determine the size of the respective box that should be ordered. With its size, we determine whether it has the size of an entire row, half of a row or a third of a row and proceed accordingly. If, for example, it has the full size, we compute the maximum of all y values, thus the

maximum of (thirdy's and halfy's). As the last step, we can now simply add the border's size, and then we place the box at halfx and the previously computed y coordinates. Since all corresponding sub boxes are now filled, we can set all y coordinates of (thirdy's and halfy's) to the previously computed max plus the border plus the size of the current box. The cases for the half and the third sizes happen analogously. They just have to previously determine which of the two respective three positions is the minimal and then only add the y values at those positions they overlap.

## 4.5 SQL Injection and the testing

### 4.5.1 What is SQL Injection

SQL injection is the process of running unauthorized queries in a database. They can reach from getting information out of the database that the user is not allowed to obtain until the deletion of specific objects in the database. Its primary cause is the use of unchecked information directly obtained from the user in the underlying databases. It has been estimated that it is one of the most popular techniques[15] to gain access to a web page. That is because many web pages are still vulnerable to such attacks.

### 4.5.2 How do I find SQL Injection vulnerabilities ?

First, I assumed two models of attackers. The first one has no login privileges and wants to try to get access to the webpage via SQL Injection. Because most APIs require a login check before, the attack surface of the first user is quite limited. Also, as I did not need to build the query dynamically, it is protected by the standard SQL insertion protection of the PostgreSQL framework. Therefore, this attack should not be too interesting, and the tests should pass easily. The second model is as powerful as a single user on the system can get. It is an admin user that has 'full' access to his part of the system. Here I used also dynamic queries that are protected by the new escaper methods. Therefore, this is the primary reason to do these tests; if something fails, it is most likely in this test. As a side note, it is essential to note that at all login-protected APIs, the login protectionism was removed at the time of the testing. However, this should not change the result, as the login checks are not connected to a backend database at this stage of the program. Therefore, here also, no attack surface for SQL Injection exists.

### 4.5.3 Used tool

As an automatic injection discovery tool, I used JSQL, which can be found under the following link on GitHub: <https://github.com/ron190/jsql-injection>. It is also the SQL Injection tool used by Kali Linux, a Linux distribution primarily famous for its 'hacking' tools.

### 4.5.4 Results of Model 1

As expected, the respecting API directly exposed to the web is secure according to the following test result of JSQL. I tested it once for each parameter (injection in this parameter) and once with an injection on all parameters denoted with a \* for the respecting parameters in the tool.

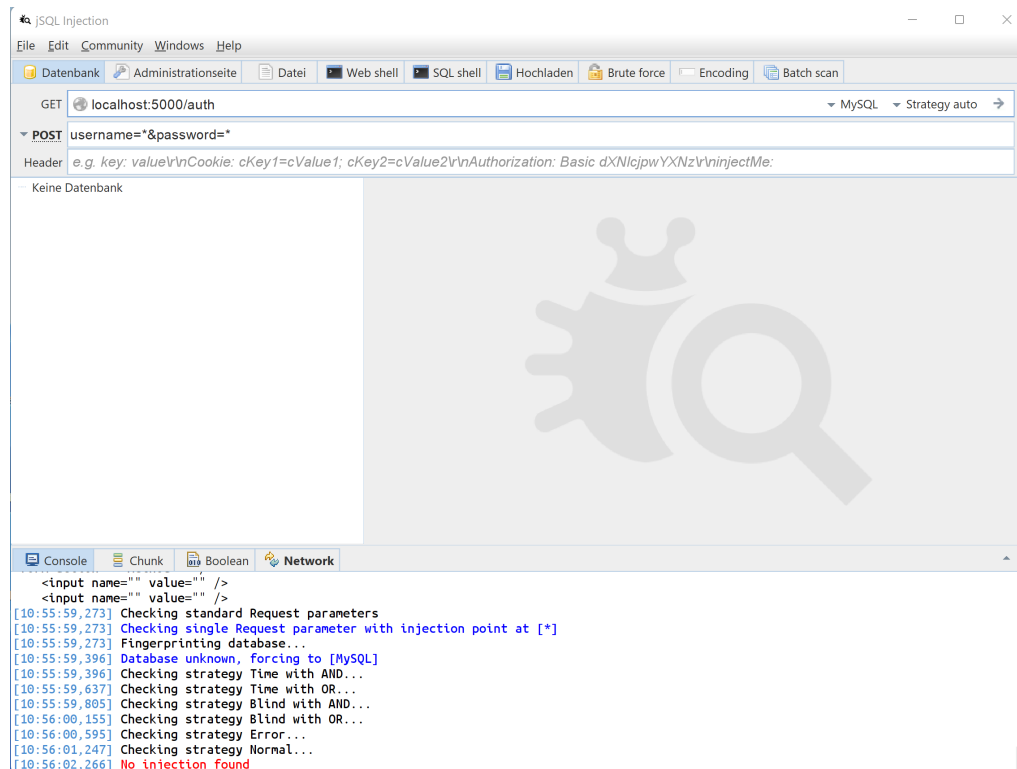


Figure 4.2: A screenshot of the test results for the auth API

### 4.5.5 Results of Model 2

Also, no SQL Insertion is possible after testing the API that in the back-end generates dynamic queries and uses the above-mentioned undocumented escaper

functionality for protection. This is a very pleasing result. It also shows that an attacker with access to the system cannot break in via SQL Insertion. In addition, it shows the usability of the new feature in PostgreSQL and its potential advantages over static queries.

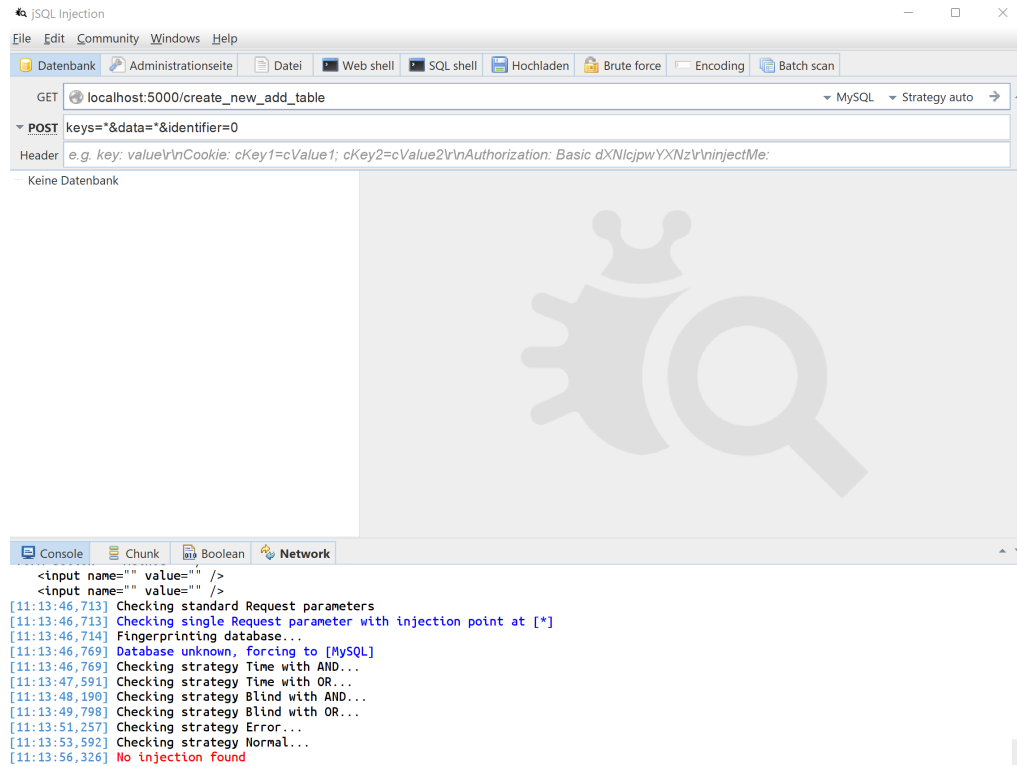


Figure 4.3: A screenshot of the test results for the 'create new add table' API

# Evaluation

---

## 5.1 Further Improvements

In my opinion, curating highly structured data is much easier and more efficient with the tool developed in this bachelor thesis. The situation with this webpage is now the following: There are tools for curating images language and with this tool also code. But, it turns out that it is usually most accessible and therefore most efficient to accomplish everything in one tool, and while the tool developed here is specialized in code, the curation of images is possible but not as mature as with the already existing tools. As a future development, it is undoubtedly advantageous to expand the functionality here. It is also possible to integrate certain machine learning functionalities directly into the tool and thus, for example, always use the data point for curation where the uncertainty is most significant. But, one must always be careful not to destroy the i.i.d. assumption of the data. However, here would certainly also be an area in which one could make the tool more intelligent.

## 5.2 Closing Words

In my opinion, the further development of humanity depends significantly on the progress of artificial intelligence. However, the basis for this is a well-founded and, above all, easy-to-create data collection. With my bachelor thesis, I was able to contribute to the fact that today's still human editors can cure this data quickly and efficiently.

# Bibliography

- [1] Hamel Husain Miltiadis Allamanis, “Code search net evaluating the state of semantic code search,” 2019.
- [2] Ziyu Yao Daniel Weld Weu-Peng Chen and Huan Sun, “Staqc: A systematically mined question-code dataset from stack overflow,” 2018.
- [3] “Conversion of an image to base64,” in <https://www.base64-image.de/>, 2022.
- [4] Stackoverflow, “C# code,” in <https://stackoverflow.com/questions/21325661/convert-an-image-selected-by-path-to-base64-string>, 2022.
- [5] appdividend, “Python code,” in <https://appdividend.com/2020/06/23/how-to-convert-image-to-base64-string-in-python/>, 2022.
- [6] I. Ward, “Json lines (jsonl),” in <https://jsonlines.org/>, 2022.
- [7] “Bulma,” in <https://bulma.io/>, 2022.
- [8] “Flexbox,” in <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>, 2022.
- [9] Mick P. Couper Reg Baker Joanne Mechling, “Placement and design of navigation buttons in web surveys,” 2011.
- [10] “Material design lite,” in <https://getmdl.io/>, 2022.
- [11] “Ace,” in <https://ace.c9.io/>, 2022.
- [12] “Bootstrap,” in <https://getbootstrap.com/>, 2022.
- [13] “Quill,” in <https://quilljs.com/>, 2022.
- [14] “Axios,” in <https://axios-http.com/>, 2022.
- [15] “Sql injection frequency,” in <https://outpost24.com/blog/SQL-injections-cyberattacks>, 2022.
- [16] “Data curation,” in <https://towardsdatascience.com/machine-learning-secret-source-curation-e8c3107dcc13>, 2022.
- [17] André Freitas and Edward Curry , “Big data curation,” in [https://link.springer.com/chapter/10.1007/978-3-319-21569-3\\_6](https://link.springer.com/chapter/10.1007/978-3-319-21569-3_6), 2022.