A system for GDPR-compliant collection of social media data: from legal to software requirements

Fredrik Jonasson
Abstract

A system for GDPR-compliant collection of social media data: from legal to software requirements

Fredrik Jonasson

As of 2018 there is a new regulation regarding data protection in the European union. The legislation, often referred to as the General Data Protection Regulation (GDPR), has led to increased demands on organizations that processes personal data. This thesis has investigated the legal consequences of social media data collection with a certain focus on collection of tweets. The legal findings was then translated into possible enhancements of a tweet collecting software. The tweet collecting software was extended with a method for pseudonymization, however it turned out that our implementation had some serious performance issues. There where also work done on an implementation of a method providing automatic tweet posting with the purpose to repeatedly inform followers of a hashtag that a collection of tweets regarding that hashtag is taking place. Lastly, some findings about possible future enhancements that can be done on the software was laid out.
7 Design of the transparency functionality
7.1 Access to information .................................. 27
7.2 Sending of information to the data subject .......... 28
  7.2.1 Repeating publication of Tweet .................. 28
7.3 Complications handled during implementation ....... 28
  7.3.1 Duplicate content .................................. 28
  7.3.2 Collection of our own published beacons ....... 29

8 Conclusion
8.1 Results and discussion ................................ 29
  8.1.1 Performance problems .......................... 29
  8.1.2 Reflections on performance efforts .......... 30
8.2 Lessons learned ......................................... 31
8.3 Future work ............................................. 32
1 Introduction

The data produced in social media networks are interesting for doing different types of social research. To be able to conduct research the data needs to be fetched and processed. Which data to fetch differs with the research. On the 25th of May 2018 the Regulation (EU) 2016/679 of the European Parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) took effect, hereafter shortened GDPR. The purpose of the legislation is to protect natural persons and the processing of their personal data. GDPR as a legislation acts as a clear statement from the European Union that most of the data being shared on social media should enjoy protection. The most clear examples of data we want to protect are personal data, consisting of names or home addresses. However, one should know that even more subtle pieces of data can pose as personal data and be used to, as an example, find someone’s political or sexual orientation.

This thesis focus is on a tool used to fetch data from the social network Twitter, the legal consequences of using the tool and how the tool can be modified to make it easier to be compliant with GDPR when using it for research. With GDPR applicable in the European Union (EU) the regulation is directly applicable in every country. However, during the work with the legal part we have used secondary sources regarding interpretations and clarifications. We have, as an example of a secondary source, turned to the Information Commissioner’s Office (ICO) who have developed guides for GDPR compliance in the United Kingdom. These sources are all adhering to the GDPR legislation but it is important to know that there can be some minor different interpretations among different countries, and guidelines, in the EU.

With this in mind, the main objective is to examine GDPR:s relation to social media data collection software and to design extensions to an existing tool that will simplify compliance with the legislation during collection of data.

2 Background

Twitter share data, that is tweets and related information, with third parties. The sharing is done by offering different possibilities to connect to Twitter and fetch, as an example, data concerning a hashtag. Sharing is done by Twitter offering application programming interfaces (API):s to the public.[11] In general terms, an API is an interface that makes information exchange between two applications possible. The API is the intermediary that takes a request and, perhaps, data from one application (for example Twitter) to another (for example another application that wants to count the occurrences of the character ‘F’ in a tweet.).[12]
2.1 Collection

When a third party is retrieving data it is commonly referred to as a collection. Collection is commonly done with different types of software and usually by using an API. This thesis is about a tool that uses the Twitter API for collection, storing and analysis of data. The tool is called *The Digital Methods Initiative Twitter Capture and Analysis Toolset* (DMI-Tcat), hereafter referred to as the tool.[13]

2.1.1 A tweet object

The Twitter API returns various kinds of data. For the scope of this thesis the tweet object is the most interesting, since this contains all the typical data that researchers collect when studying social media on Twitter. When fetching a tweet from the Twitter API it is returned in JSON-format manner that is called a Tweet object.[21]

A tweet object consists of what one would call a tweet i.e the text. The object also consists of related information, as an example the userID, an unique number that is linked to a Twitter user. Since every user has an unique userID it can be used to identify a user on Twitter. There is also a unique number attached to every tweet object making it uniquely identifiable.

2.2 The Digital Methods Initiative Twitter Capture and Analysis Toolset

DMI-Tcat is an open source and freely available tool that aims to provide both capture and analysis of data.[10] The tool is freely available and open source.[13] The software being open source permits access and modification to the source code. After modification, usage and distribution of the modified version is permitted.[9] For this thesis that means a permit for us to take the source code of the software, extend it with new features, and then use and distribute it freely.

2.2.1 Using DMI-Tcat

DMI-Tcat supports both capturing and analysis of Twitter data. When it comes to capturing it is important to note that the software uses the Twitter API and is thereby bound to the limitations of that API. We can only capture the data that Twitter is offering to their API users. The analysis part of the tool offer a wide range of options for visualizing and exporting data. These options can be used to get different insights during research. The program is able to export to different common file formats as CSV and GEXF. CSV is an acronym for Comma-Separated-Values and it is used for tabular files. The most common usage of CSV files are probably in spreadsheet programs such as MS Excel or Google Spreadsheets. The GEXF, an acronym for Graph Exchange XML Format, is a common format for describing advanced networks and their structures. The format is supported by multiple software programs such as as
2.2.2 DMI-Tcat on an architectural level

DMI-Tcat is consists of a capturing and analysis part. These parts work independently from each other and only have the database in common. An important detail about the different parts are that you need to be a granted and logged in admin user to use the capturing part of the software. Hence only an admin user can collect data. This has the consequence that a "regular" user will have to get settled with only having access to the analysis part of the software while the admin can use both the capture and analysis parts.

Figure 1: The architecture of DMI-Tcat with the capture part on the left, the database in the middle and the sub-sampling and analysis part on the right.

2.3 Regulation

Due to the nature of social media, there tends to be a lot of personal data shared between different users of the social media platform. The most pressing, and for this thesis relevant, regulation regarding the collection and analysis of this data, which in many cases are personal, is the GDPR legislation.

2.3.1 Natural persons and processing of their personal data

GDPR declares that the protection of natural persons in relation to the processing of personal data is a fundamental right.

There are some central terms that are cornerstones of the legislation. These terms constitute who the legislation aims to protect and from what. These terms are explained below.

A natural person is a person in flesh. In social media the natural persons can be referred to as users. However, it is important to note that not all users on a social media platform are natural persons. As an example there exist a lot of companies who maintain accounts on the big social media platforms. Since
these companies are not natural persons the GDPR legislation does not apply to them. A common synonym of natural persons in GDPR are data subjects.

Processing of personal data needs to comply with certain rules. The term processing is defined in article 4 in GDPR as:

‘processing’ means any operation or set of operations which is performed on personal data or on sets of personal data, whether or not by automated means, such as collection, recording, organization, structuring, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available, alignment or combination, restriction, erasure or destruction.

The definition of personal data is: ”personal data’ means any information relating to an identified or identifiable natural person (‘data subject’); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person”.

With the understanding of these terms one can usually reel off a lot of areas where the processing of personal data is being done, especially when it comes to the digital applications and services that one uses every day.

2.3.2 Personal data processing on Twitter

Much of the data publicized on Twitter is personal data since it can be used to identify a natural person. Therefore this kind of data enjoys legal protection from GDPR against unlawful processing. An example of Twitter processing personal data is the storing of tweets that are containing information about where a natural person lives.

It should hereby be clear that Twitter is processing personal data regarding natural persons and therefore has to comply with the GDPR legislation.

2.3.3 GDPR appliance when capturing tweets from the Twitter API

Besides Twitter, also a third party who is collecting data from Twitter needs to comply with the GDPR legislation.

When creating a Twitter account one has to give consent to the terms and conditions of the service. This consent gives Twitter, and maybe(Thismoebayisexplainedmoreindetailinsection2.4.1) even third parties, some freedom in how they can use the personal data that a user are posting.

Generally, regulations and their appliance is a subject lacking precise answers. In every case there exist certain circumstances that can make a verdict differ a lot.

With GDPR being a new regulation there is a lacking of guidance from the European Union courts regarding how to interpret all the parts of the legislation. In an effort to mitigate this uncertainty a significant amount of research of the
regulation and its implementation has been done for this thesis. Besides providing technical discussion and solutions the thesis aims to give further insights on the legal matters as well.

2.3.4 Anonymization and pseudonymization

GDPR mentions anonymization and pseudonymization as means of an action that provides security for the data subject.

Anonymization of personal data is the act of stripping the personal information in such a way that an individual cannot be identified. As a data processor, the benefits that follows by anonymizing collected data can be read from GDPR recital 26 that says that "The principles of data protection should therefore not apply to anonymous information, namely information which does not relate to an identified or identifiable natural person or to personal data rendered anonymous in such a manner that the data subject is not or no longer identifiable". One should know that even the anonymization of data is processing according to GDPR, meaning that up to the point of the data being unable to be used for identification of a data subject the regulation needs to be followed.[6]

Pseudonymization is a process that results in the personal data being replaced with a reference number. To be able to depseudonymize and thereby tie the reference number back to a individual you need additional information. By implementing technical and organizational measures you can ensure that the mentioned additional information is kept separately, thereby adding a layer of protection. Personal data that is pseudonymized is still viewed as personal data by GDPR, however, since pseudonymization reduces the risk for data subjects GDPR consists of multiple incentives for processors to pseudonymize their stored personal data.[6]

One of these incentives resides in article 6.4(e). The article permits some remission on the obligations regarding processing for a purpose other than for the initial purpose to which the data subjects have given their consent. When ascertaining whether processing for another purpose than the initial still could be compatible with the initial one, and therefore being able to continue to process without further measures, there are some factors being taken into account. One of these factors are the existence of appropriate safeguards, where pseudonymization is explicitly mentioned as an example of a possible safeguard.

Another incentive resides in article 89 where purpose limitation can be excepted from while the processing is done for "archiving purposes in the public interest, scientific or historical research purposes or statistical purposes". To use this exception the one responsible for capturing the data needs to make sure that there exist safeguards. Again, in article 89(1) pseudonymization is explicitly named as a possible safeguard.

2.4 Purpose of the thesis

The main goal with this thesis is to enhance the DMI-Tcat tool in a way that makes it easier for a user to be GDPR-compliant when working with the tool.
An optimal work flow which utilizes the already developed functionality of the DMI-Tcat plus added functionality would act as in figure 2.

An fully GDPR-compliant tool should contain all the functionalities above. However, there exists technical limitations that makes the implementation of everything in the workflow too big of a task for this thesis. Thus, this thesis will try to satisfy some of the required functionality and to pave the way for implementation of the other parts.

2.4.1 A Twitter users legal matter

When signing up on Twitter, a data subject has to accept the terms of service[15], the privacy policy[16] and the use of cookies. The privacy policy explicitly informs the user about tweets being public, and being used by Twitter as publicly and freely available. An exception is made for Twitter’s non-public ways of communication. The intention can be seen in this excerpt from the terms "In addition to providing your public information to the world directly on Twitter, we also use technology like application programming interfaces (APIs) and embeds to make that information available to websites, apps, and others for their use - for example, displaying tweets on a news website or analyzing what people say on Twitter.” The terms also inform that some other information is going to be public. That information is for example profile information, timezone and
language.

For the scope of this thesis, we are only going to focus on collecting data that has been posted in the public part of Twitter.

Since the data subject by registering to Twitter accepts the services widespread use of posted information one can argue that the data subject gives Twitter a consent to the processing of personal data with the purpose of "sharing information publicly" in accordance with article 6.1(a) in GDPR. And that it therefore is free for Twitter to spread the information, even to third parts, to their liking.

Twitter’s developer terms, which you have to accept to use the API, gives the impression that you don’t need to get further consent for collecting data from the API. Although the terms are mentioning some parts where you have to get a consent from the data subject, collection of tweets are not amongst these.[1]

However, in our opinion it is doubtful how applicable the consent a data subject gives when accepting Twitter’s privacy policy is to a third party collector. Can a third party rely on this consent to use the, on Twitter published, tweets for the collector’s own purpose? If this is the case, it would be equal as saying that you as a data subject when you are posting something on Twitter accept that it might be spread everywhere and for any purposes for indefinite time. This is a very long reaching, and general, interpretation that does not work well with GDPR’s explicit purpose to protect the data subject.

An important part of a consent according to GDPR is the principle of the purpose to which you give your consent. There needs to be a specific, explicit and legitimate purpose to which the data subject gives it’s consent.[3]

There is a possibility that the consent given by a data subject when publishing on Twitter ends with usage on Twitter’s platform and by Twitter, and one can argue that it is not possible for Twitter to collect a general consent that gives the right to "public spreading" since that purpose might be too general. When a user removes his or her published information from Twitter, the earlier consent to which Twitter can use the personal data is also removed. This makes it doubtful if a third party freely can continue to use the, from Twitter removed, data.

2.4.2 Responsibilities towards the data subject as a third party collector

When a collection of data is done from the Twitter API, the collector inherits the responsibility towards the data subject from Twitter. This means, amongst other things, that the collector needs to comply with the demands stated in article 5 GDPR for the legality of the processing. If the processing is done in a way that complies with article 89(1) it is easier to to comply with article 5.1 (b) and (c) due to the demands being more relaxed.[5]

Further legal discussion will now be paired with suggestions of new features that can be added to the software to make it easier to comply with the legislation. A compact version of what we would suggest the system to do and why it should do it can be viewed in Table 1.
3 System requirements

A tool alone, no matter how GDPR-enhanced, is not enough to comply with the GDPR legislation. There need to be an organized approach to the data processing and clear purposes before one even starts to capture any data. With that being said, since a user of DMI-Tcat most likely will have to comply with the legislation, the ambition of the thesis is to come up with suggestions and in some cases implementations of technical features that facilitate compliance. As shown in Table 1 we have aimed on specifically four rights that we think a third party collector should offer the data subject. On Table 1 these rights are mapped against the corresponding system requirements.
Table 1: The relation between a data subject privilege, corresponding legislation and a possible technical feature that would assist in compliance with these demands from the legislator

<table>
<thead>
<tr>
<th>Data subject privilege</th>
<th>GDPR - legislation</th>
<th>Corresponding System Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudonymization article 6.(4)e, 25, 32.(1)a and 89(1)</td>
<td>From the GDPR interface, the collector should have the choice, before data collection begins, to pseudonymize the data. The tool should then in an automatic way collect the data and then pseudonymize the data by for example replacing screen name and userID with a random code. After the collection is finished the collector shall be given a CSV file containing information that enables a twitteruser to undo the pseudonymization and translate the data again and make it identifiable who sent what.</td>
<td></td>
</tr>
<tr>
<td>Subjects right to access article 15</td>
<td>The system shall be able to collect data concerned to a twitteruser. The tool shall be able to compile this data into a format that is portable, for example a CSV file or a PDF. It shall then be possible to export this data and retrieve it in some commonly used file format. The purpose is to be able to grant a twitteruser a sample of his or her data.</td>
<td></td>
</tr>
<tr>
<td>Subjects right to rectification article 16</td>
<td>The tool shall be able to collect data concerned to a twitteruser. The collection shall be done in a way that is similar to the methods for collection when it comes to “right to be forgotten” and “access to data”. The tool shall be able to access certain fields in this data set and make changes to all of these fields and thus update and rectify the data set.</td>
<td></td>
</tr>
<tr>
<td>Subjects right to be forgotten if it is not damaging scientific research done in accordance with 89(1), article 17(1) and article 17(3)d.</td>
<td>The tool shall be able to collect data concerned to a twitteruser. The data shall be compiled in such a way that it is possible to export in for example a CSV file. It should also be reachable by the system in such a way that it is possible to enter a twitteruser id, get the data concerned to a twitteruser and then make a choice so that the tool deletes that data.</td>
<td></td>
</tr>
<tr>
<td>Fair and transparent processing recital 60 and the principle of fair and transparent processing</td>
<td>The tool shall be able to inform twitterusers that their tweets are being collected. It should also provide the users with information how the users can express their rights.</td>
<td></td>
</tr>
</tbody>
</table>
3.1 Some words about transparent processing

Three of the privileges listed in Table 1 require fulfillment of the GDPR principle of *fair and transparent processing* to be of practical use. This is expressed in practical terms in recital 60 which states that "The principles of fair and transparent processing require that the data subject be informed of the existence of the processing operation and its purposes. The controller should provide the data subject with any further information necessary to ensure fair and transparent processing".

Thus there is an expectation that the data processor in most cases shall contact the data subject to inform about the collection of the data-subjects data and information regarding purpose and so forth. If a data subject is unaware that his or her data are being processed by a third party, the mere possibility to have a right to access, rectification or deletion is not enough to make a difference from the data subjects perspective. As a third party collecting data from Twitter it can be complicated to fulfill the principle of transparency. There is an obstacle that is dependent on Twitter’s own rules regarding spam. If a third party has collected thousands of tweets and wants to inform the relevant users, via Twitter, that their data is being processed there is a risk of being banned by Twitter for spamming when sending out messages to all relevant users.

This problem has no clear solution, even though there are some ways to at least try to achieve transparency, for example posting a single Tweet periodically informing that you are processing tweets regarding a certain hashtag, a method we have investigated. This Tweet would then be shown to others searching for the actual hashtag, and then create awareness.[20] This solution comes with the risk that the Tweet just disappears in the mass of other tweets being published.

Aware of the problem, and the lack of a fully satisfying solution, we still chose to focus on the privileges listed in the table.

3.2 Contact with the community

Since DMI-Tcat are managed by a community we reached out to them on their official Github repository where the design was discussed.[8] Our ambition was that the developed features would be merged to a patch that would be added as a part of the software’s main stream.

4 Design

4.1 Presenting the design of pseudonymization for the community

To make sure that our work was beneficial to the open-source community behind the tool we made them a proposal that consisted of three enhancements:

- Pseudonymization.
• The right to see the data that is stored.
• The right to get one’s data deleted.

The response was positive and a more detailed description wished for.

4.2 Choosing pseudonymization

It was decided to start working with the pseudonymization due to this feature being considered giving most benefit to the users regarded the amount of work necessary to implement it. Hence this implementation was the first, and as it would turn out, the only one where we presented and discussed the design in detail with the community.

4.3 Planning and discussion

The initial proposal consisted of modifying the collecting part of the software. This would place the greater part of the code on the capturing side of the software as seen in figure 3.

The reason for placing the pseudonymization before saving the data is legal. Since capturing, storing and analysis constitutes processing one would want to come in use of the relief from some of the GDPR obligations that follow processing as soon as possible. In that way some of the later processing after the pseudonymization is completed would not have as heavy obligations from GDPR.

The first step to implement pseudonymization functionality was to get familiar with the existing codebase to get a good understanding of where to put our modifications.

4.3.1 Source code of the capture part

When the software captures data from Twitter the tweets and related information are processed by mainly two PHP-classes; Tweet and Tweetqueue. Raw data from the API are put in an array named capturebucket. The capturebucket consists of data requested by a certain query.

From the capturebucket a Tweet object is created by assigning all attributes of the actual Tweet to the created object. After a Tweet object is created one can use syntax, who should be familiar to users of Objective oriented languages, like the ones in listing 1 to get information regarding a Tweet.

```plaintext
1  echo $Tweetobject -> from_user_name;
2  echo $Tweetobject -> from_user_id;
```

Listing 1: Illustration of the Tweet object, its attributes and how the attributes are accessed.

Before being uploaded to the database, every Tweet object is stored in an array inside the Tweetqueue object. The Tweetqueue class then operates on the
array with methods and attributes who as an example are making sure that all
the tweets belonging to a query are saved in the table containing all the other
tweets from the same query.\[4\]

With the Tweetqueue class consisting of methods for working with the inser-
tion of tweets into the Database, it is the last class the data passes before being
saved to the database. Our initial plan for implementing pseudonymization was
to do the extensions in the Tweetqueue class making it pseudonymize the tweets
before inserting them into the database.

When introducing this planned approach to the community it became clear
that they where doubtful about making changes in the code responsible for the
capture and database handling since they were of the opinion that it would
complicate things.

4.3.2 Feedback and changed plans

The input from the community on this issue clearly expressed that they wished
to, instead of pseudonymizing the data before inserting it to the database,
pseudonymize it at the point where it is extracted from the database to be
analyzed. A consequence of their suggestion is that the data will be stored, and
thereby processed, depseudonymized in the database providing some less safety
measures to the data subjects.

Their idea was our second choice from the legal point of view. However, even
with a design according to their suggestion our opinion was, and still is, that it
would still be a useful feature that would help us reaching our goal of making
it easier to be GDPR-compliant as a user.

The implementation plan was changed to make the pseudonymization take
place upon extraction on the analysis part of the program instead.

4.4 Implementation on Architecture level

As one can see in figure 1 the software can be divided in two parts, one that
fulfills the purpose of capturing data to the database and one that makes an
analysis on that data possible.
Efficiency and modularity was the ambition while planning for the pseudonymization functionality. By storing most of the pseudonymization code in one file these ambitions were somewhat achieved. The plan of attack was to force the original flow of the data by a file called `pseudonymization.php` where both pseudonymization and building and maintaining of a table consisting of the data needed for depseudonymization is done. By simply replacing the original data with pseudonymized data and then returning the pseudonymized data to the original flow of the program we would make sure that as little of the original code as possible was tampered with.

See figure 3 for an illustration of how the flow of the program will be forced to take the way by one of the pseudonymization modules.

4.5 User stories to implement

The user stories for the pseudonymization are the following:

- From the capturing interface the administrator should have the choice, before data collection begins, to pseudonymize the data. The tool should then mark the actual collection for pseudonymization. The tool should also create a pseudonymization table if it doesn’t exist already. The pseudonymization table holds the pseudonymized value and its corresponding reference value. The pseudonymization table is the measure one needs to depseudonymize data.

- Whenever a user wants to do an analysis of the pseudonymized data, he or she can export the data as usual.

- During export of the data, the tool pseudonymizes the data by replacing the original identifiable data with a reference number. The original data is then stored in the pseudonymization table where the original data, the
corresponding reference number and the type of original data is stored. The pseudonymized data is then given to the user.

- As an administrator, there exists a possibility to export the pseudonymization data from the analysis and export-interface. The tool delivers a CSV file to the user. The file contains all the information that was stored in the earlier step. With the help of the pseudonymization table the administrator can translate pseudonymized values and original data.

### 4.6 Specifications for the implementation

To be able to fulfill these user stories, the following ideas and plans for the implementation were laid down.

First, add a check box to the capture interface where you can choose if you want to pseudonymize the data you are about to capture.

Then, make sure to have a way to be able to check which collections that are pseudonymized or not. After suggestions from the community this was achieved by adding another column, pseudonymized, to the table tcat_query_bins.

Tcat_query_bins lists all the collections in the database and in our newly added column one can see if a collection are to be pseudonymized or not.

While a collection is started, there is already now a function that checks that all the necessary tables in the database exists. If any table are missing the function creates one. By extending this function so it also checks and, if necessary, creates a pseudonymization table there will always exist a table when we need it. The table will contain the information that gives the possibility to depseudonymize the data residing in the query bin.

The pseudonymization table will have three columns:

- **Reference value**, also referred to as Pseudonymize value. This value will replace the original data in the pseudonymized table. This will be the primary key of the table. Therefore every value in this column needs to be unique.

- **Original data** is the data that we are replacing with a pseudonymized value. The original data is data that we have chosen to hide and instead show the corresponding pseudonymize value.

- **Data type**. This value describes what kind of data we are pseudonymizing. As an example, if we are pseudonymizing a screen name the data type field will contain the string ‘screen name’. Since a lot of the values that we are pseudonymizing consist of numbers or more or less strange names that can appear confusing this column make the interpretation of the pseudonymization table easier.
5 Pseudonymization data types, storage and representation

The programming has mainly been done in the PHP language. The language only have one data type, the array. The array can be one dimensional or multidimensional with keys and values. When searching for a key in a multidimensional array it can be compared with a hash map when it comes to speed.

5.1 Storage of fetched data

The tool creates a database and maintains some tables in the database. Some of the tables are the same for all of the collections, as an example the table that keeps track of all collections, with related information, of tweets.

Some tables are created exclusively for every collection of tweets that is initiated. Examples of these tables are the table containing all the tweets in a collection or the table containing all the hashtags in a collection. To illustrate the table structure there is an excerpt of the database tables used by the tool in listing 2. There one can see tables that are shared among all collections of tweets but also tables that are unique for the collection named ”small”.

Listing 2: Tables in a DMI-Tcat when there is one stored collection of tweets named *small*.

Listing 2 shows different tables containing different types of data. All the tables which names are starting with *tcat_* are shared amongst all collections. An excerpt of the table *small_tweets* and it’s structure is shown in listing 3.
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>bigint (20)</td>
</tr>
<tr>
<td>created_at</td>
<td>datetime</td>
</tr>
<tr>
<td>from_user_name</td>
<td>varchar (255)</td>
</tr>
<tr>
<td>from_user_id</td>
<td>bigint (20)</td>
</tr>
<tr>
<td>from_user_lang</td>
<td>varchar (16)</td>
</tr>
<tr>
<td>from_user_Tweetcount</td>
<td>int (11)</td>
</tr>
<tr>
<td>from_user_followercount</td>
<td>int (11)</td>
</tr>
<tr>
<td>from_user_friendcount</td>
<td>int (11)</td>
</tr>
<tr>
<td>from_user_listed</td>
<td>int (11)</td>
</tr>
<tr>
<td>from_user_realname</td>
<td>varchar (255)</td>
</tr>
<tr>
<td>from_user_Tweetcount</td>
<td>int (11)</td>
</tr>
<tr>
<td>from_user_Tweetcount</td>
<td>varchar (255)</td>
</tr>
<tr>
<td>from_user_timezone</td>
<td>varchar (255)</td>
</tr>
<tr>
<td>from_user_description</td>
<td>varchar (255)</td>
</tr>
<tr>
<td>from_user_url</td>
<td>varchar (2048)</td>
</tr>
<tr>
<td>from_user_verified</td>
<td>tinyint (1)</td>
</tr>
<tr>
<td>from_user_profile_image_url</td>
<td>varchar (400)</td>
</tr>
<tr>
<td>from_user_created_at</td>
<td>datetime</td>
</tr>
<tr>
<td>from_user_withheld_scope</td>
<td>varchar (32)</td>
</tr>
<tr>
<td>from_user_favourites_count</td>
<td>int (11)</td>
</tr>
<tr>
<td>source</td>
<td>varchar (512)</td>
</tr>
<tr>
<td>location</td>
<td>varchar (64)</td>
</tr>
<tr>
<td>geo_lat</td>
<td>float (10,6)</td>
</tr>
<tr>
<td>geo_lng</td>
<td>float (10,6)</td>
</tr>
<tr>
<td>text</td>
<td>text</td>
</tr>
<tr>
<td>reTweet_id</td>
<td>bigint (20)</td>
</tr>
<tr>
<td>reTweet_count</td>
<td>int (11)</td>
</tr>
<tr>
<td>favorite_count</td>
<td>int (11)</td>
</tr>
<tr>
<td>to_user_id</td>
<td>bigint (20)</td>
</tr>
<tr>
<td>to_user_name</td>
<td>varchar (255)</td>
</tr>
<tr>
<td>in_reply_to_status_id</td>
<td>bigint (20)</td>
</tr>
<tr>
<td>filter_level</td>
<td>varchar (6)</td>
</tr>
<tr>
<td>lang</td>
<td>varchar (16)</td>
</tr>
<tr>
<td>possibly_sensitive</td>
<td>tinyint (1)</td>
</tr>
<tr>
<td>quoted_status_id</td>
<td>bigint (20)</td>
</tr>
<tr>
<td>withheld_copyright</td>
<td>tinyint (1)</td>
</tr>
<tr>
<td>withheld_scope</td>
<td>varchar (32)</td>
</tr>
</tbody>
</table>

Listing 3: Columns containing information about a Tweet stored in the `small_tweets` table.
5.1.1 Export and analysis

To illustrate how the export and analysis page with its related functions interact, see Figure 4.

Figure 4: An illustration of how an export without pseudonymization works on an architectural level.

As seen in the illustration, if a user wishes to export data, he or she has multiple choices of what to export. For example, a full export, using `mod.export_tweets.php` or just an export of hashtags using `mod.export_hashtag.php`.

When making a choice on the analysis page a request is sent to one of the analysis or export files tasked with retrieving the searched information and present it in form of an exportable CSV file.
while ($data = $rec->fetch(PDO::FETCH_ASSOC)) {
    $CSV->newrow();
    if (preg_match("/urls/", $sql) || preg_match("/media/", $sql) || preg_match("/mentions/", $sql))
        $id = $data['Tweet_id'];
    else
        $id = $data['id'];
    $CSV->addfield($id);
    $fields = array('created_at', 'from_user_name', 'text', 'filter_level', 'possibly_sensitive',
                    'withheld_copyright', 'withheld_scope', 'truncated', 'reTweet_count', 'favorite_count',
                    'lang', 'to_user_name', 'in_reply_to_status_id', 'quoted_status_id', 'source', 'location', 'geo_lat', 'geo_lng',
                    'from_user_id', 'from_user_realname', 'from_user_verified', 'from_user_description',
                    'from_user_url', 'from_user_profile_image_url',
                    'from_user_utcoffset', 'from_user_timezone', 'from_user_lang', 'from_user_Tweetcount',
                    'from_user_followercount', 'from_user_friendcount',
                    'from_user_favourites_count', 'from_user_listed',
                    'from_user_withheldscope', 'from_user_created_at');
    foreach ($fields as $f) {
        $CSV->addfield(isset($data[$f]) ? $data[$f] : '');
    }
}

Listing 4: An extract from the file mod.export_tweets.php which is called when an export of all the tweets and their related data is chosen.

Of certain interest in listing 4 is the array named $fields on row 9. As shown this array consists of multiple keys, these keys correspond to the columns shown in the excerpt from the database in listing 3. On row 1 one can see that as long as there are tweets left that we want to export, these are saved in a variable called $data. The $data variable can be thought of as one row from the table ending with "tweets" and therefore represents one Tweet. On line 2 a CSV object gets a new row and in the loop at row 10 the fields that we wish to show as columns in the exported CSV table are written to the CSV object. Then we go back to row 1 to check if there are any Tweet left that we want to export.
5.1.2 Access policy

For the pseudonymization to work effectively there is a need for a solution where a regular user just cannot access pseudonymized data. A regular user should thereby not be able choose whether the pseudonymization shall be activated or not. There has to be some kind of limitation, or an access policy where the pseudonymization is guaranteed and thereby forced by someone responsible for the data processing. Any other option would leave the pseudonymization completely useless since the point of protecting the data subject’s data would be lost.

5.1.3 Community concerns

During the discussion regarding this limitation for a user to freely choose whether to view the data as pseudonymized or not there were some concerns from the community regarding the autonomy given to the users of the tool.

Even if one can understand the consideration of the autonomy for the users an implementation of a pseudonymization that anyone freely can turn on and off would be useless. To still adhere to the purpose of this thesis the features we suggest and implement need to have a real and positive impact on the tool:s GDPR-compliance.

Therefore it was decided to only let administrators have access to the pseudonymization table, against the community concern about the user autonomy. However, we decided to follow the recommendations from the community regarding where in the architecture the pseudonymization functionality would be implemented.

As a good compromise between usability, autonomy and the legal reality the development began. In parallel with the development we have tried to explain our standpoint to the community and also asked them for suggestions that would both fulfill legal and usability requirements. They have not yet returned. The lack of response is understandable, the community are working with this on their spare time and to participate in a legal discussion on this level is something that requires one to read and investigate the legal matters quite a bit.

A careful expectation is that, even if the community does not think that these changes live up to their requirements, there will still be a bigger understanding regarding what a compliant tool requires thanks to this added features and act as a inspiration to come up with own suggestions.

6 Implementation

With the ambition of reusing as much code as possible and to keep changes at the same place a new file, pseudonymization.php, was created. The files purpose was to contain the code responsible for this feature. Figure 5 gives an idea of how different files and parts in the program are interacting with each other.
6.1 Cooperation with other files

Fortunately, the original design of DMI-Tcat is already written and designed in a very modular way. An example of this modularity can be seen in figure 5. This makes it fairly easy to implement the call to our module in the different original files.

Figure 5: An illustration of how an export with pseudonymization works on a per-file-level.
while ($data = $rec->fetch(PDO::FETCH_ASSOC)) {
    $last_index = pseudonymize($data, $pp);
    $CSV->newrow();
    if (preg_match("/urls/", $sql) || preg_match("/media/", $sql) || preg_match("/mentions/", $sql))
        $id = $data['Tweet_id'];
    else
        $id = $data['id'];
    $CSV->addfield($id);
    $CSV->addfield(strtotime($data['created_at']));
    $fields = array('created_at', 'from_user_name', 'text', 'filter_level', 'possibly_sensitive', 'withheld_copyright', 'withheld_scope', 'truncated', 'reTweet_count', 'favorite_count', 'lang', 'to_user_name', 'in_reply_to_status_id', 'quoted_status_id', 'source', 'location', 'geo_lat', 'geo_lng', 'from_user_id', 'from_user_realname', 'from_user_verified', 'from_user_description', 'from_user_url', 'from_user_profile_image_url', 'from_user_utcoffset', 'from_user_timezone', 'from_user_lang', 'from_user_Tweetcount', 'from_user_followercount', 'from_user_friendcount', 'from_user_favourites_count', 'from_user_listed', 'from_user_withheld_scope', 'from_user_created_at');
    foreach ($fields as $f) {
        $CSV->addfield(isset($data[$f]) ? $data[$f] : '');
    }
}

Listing 5: Excerpt from the same source code as in listing 4. Here with added pseudonymization functionality which is represented by the function call at line 3. Note that the function pseudonymize, which is residing in the file pseudonymization.php is handling both global variables and parameters sent
6.1.1 The pseudonymization table

Since pseudonymization, unlike anonymization, offers the possibility of depseudonymizing the data and making it possible to identify individuals there needs to be a data structure supporting depseudonymization. Without such a data structure one could not depseudonymize the data and the data would instead become anonymized. The structure is a table keeping track of what information that is masked by which reference number is needed. In this implementation, that table is the pseudonymization table visible in listing 6.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>pseudo_val</td>
<td>bigint (11)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>original_data</td>
<td>varchar (255)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>fieldtype</td>
<td>varchar (255)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

Listing 6: Structure of the pseudonymization table.

The different columns in listing 6 have the following functionality:

- **pseudo_val** is the reference value that will replace and thereby mask the original data in the table that is being pseudonymized. The column is set as primary key with auto incrementation, thus making sure that there is a unique replacement value for every unique original data that is masked.

- **original_data** consists of the data that we are masking. This column can consist of a screenname, a user id or any other data that we want to pseudonymize.

- **The field type** is a column with the purpose of explaining what kind of data that is masked. For a pseudonymized id value, this column would simply contain the string "id".

6.2 Writing of pseudonymization.php

To have the module pseudonymization.php acting as independent as possible there was a need for, in addition to direct pseudonymization functions, other supporting functionality.

6.2.1 Supporting functionality

Since the module shall be able to take the data, process it and return it to the regular program the following tasks need to be taken care of by support functionality:

- Fetch the existing pseudonymization table from the database and store it into an array thus making it accessible during the whole pseudonymization process.

- Check if a certain collection of tweets are marked for pseudonymization.
• Keep track of how many records that exists in the pseudonymization table and how many that are added during the pseudonymization process. Thus also keeping track, and making sure, of the pseudo_vals incrementation and uniqueness.

• Save the added values of the pseudonymization table back to the database.

6.2.2 Pseudonymization function

In the listing immediately below the procedure PSEUDONYMIZE used for pseudonymization is described with pseudocode.

1: procedure PSEUDONYMIZE(fields_for_pseudonymization, pseudonymization_table, Tweet, startindex)

2:   fields_for_pseudonymization = an array consisting of all the fields that we would like to pseudonymize in a Tweet.

3:   pseudonymization_table = a table where all pseudonymized original data are stored together with their pseudonymization value.

4:   Tweet = a Tweet and its fields of data.

5:   startindex = index of the last entry in the pseudonymization table.

6:   for each field f and its value in Tweet do

7:     if f exists in fields_for_pseudonymization then

8:       if value exists in pseudonymization_table then

9:         value = pseudonymization_table[value]

10:      else

11:         startindex = startindex + 1

12:         pseudonymization_table[startindex] = value

13:     value = startindex

14:

15:

There exists one special case where this approach will not suffice and that is when there is a mention of a user in a text. Since a big part of Twitter is to reTweet or in other ways refer to other users there are frequently appearing user names in text.

Since we want to pseudonymize only certain parts of the text and not the whole text we search the text with help of a regular expression that only matches the substrings containing mentioning of users that we want to replace.
In the pseudocode listing immediately below we see the procedure PSEUDONYMIZE walking through textfields searching for substrings with aid from a regular expression.

1: `procedure Pseudonymize(fields_for_pseudonymization, pseudonymization_table, Tweet, startIndex)`
2: \[
\text{if } f == 'text' \text{ then} \\
\]
3: \[
\text{search } f \text{ after any substrings starting with '@'} \\
\]
4: \[
\text{string } = \text{appendstringwithall findingsof substringsstartingwith'@'} \\
\]
5: \[
\text{for every substring } S \text{ in string do} \\
\]
6: \[
\text{if } S \text{ exists in pseudonymization_table then} \\
\]
7: \[
\text{value } = \text{pseudonymization_table[value]} \\
\]
8: \[
\text{replace occurences of substrings in string with } S. \\
\]
9: \[
\text{else} \\
\]
10: \[
\text{startIndex } = \text{startIndex + 1} \quad \text{pseudonymization_table[startIndex]} = \text{value} \\
\]
11: \[
\text{S } = \text{startIndex} \\
\]
12: \[
\text{replace occurences of substrings in string with } S. \\
\]

With the above functionality in place, a pseudonymization of all the values is achieved.

7 Design of the transparency functionality

Despite the challenges for a data subject presented in section 3.1 we found a possible implementation for the case where the capturing of tweets are done based on a certain hashtag. This gives the collector of data, where the basis of collection are dependent on a hashtag, to inform a potential data subject that a collection of tweets is occurring. The data subject will be informed while viewing tweets with a hashtag that are collected. This was made possible by the sending of a Tweet, containing the hashtag that we are collecting (hereafter called beacon), with the help of Twitter’s API.

Except providing information, the beacon provides any user the opportunity to enter a page where they can get information regarding if the collector have stored any of their tweets.

7.1 Access to information

To be able to provide the data subject with information regarding the data collected regarding him or her we designed a page where the data subject can use Twitter credentials to log in. This is made possible by Twitter offering a solution called Twitter OAuth.

By offering login with the Twitter account we make sure that only concerned persons can access our site. When they are using Twitter OAuth they are giving us their unique id for their twitter account, we then use this id to search our
database if we have data stored that belongs to the user of the account with the specific id.

When logged in the data subject is presented with the different collections in which he or she is represented, there is also a possibility to open each collection and there see exactly which Tweet we have stored.

7.2 Sending of information to the data subject

As discussed in section 3.1 it is hard to fully comply to the principles of transparency and information regarding informing a data subject on Twitter about his or her’s data being collected. However, when it comes to collecting data based on a certain hashtag or location we assume that a user who is contributing to a hashtag-given discussion will probably also follow what others write with that hashtag. If the collector then publishes a beacon containing that hashtag and informs about that a collection of tweets is done for that hashtag there is a probability that the data subject sees the message and hence knows about the collection and the fact that he or she probably is represented there.

7.2.1 Repeating publication of Tweet

To make sure that the beacon does not disappear too fast among all the other tweets containing the same hashtag we wanted a repeated publishing where we inform about the collection taking place.

As one can see in Figure 1 there is a Cron job who runs the capturing script once every minute. Cron is a job-sheduler in Ubuntu where one can schedule repetetive tasks.[17] In the case of DMI-Tcat, the repetitive task is calling a PHP file every minute to fetch data from the Twitter API and insert it to the database, this is taken care of by a Cron job.

To keep our impact on the architecture at a minimum we decided to take advantage of the existing Cron job as it was. Also, even if the Cron job run once per minute we only want to send one Tweet per day due to Twitter’s rules for the API.

To not send a Tweet everytime the Cron job gets called we used the fact that the collections we are sending beacons for have a stored date and time when they where created.

Every minute when the Cron job is called we check if it is the time on the day that the tracking of the collection begun. A hashtag that we started to track at 22:11 will therefore always have its beacon sent at 22:11 every 24 hours.

7.3 Complications handled during implementation

7.3.1 Duplicate content

When posting tweets with Twitters API one rule is that you can’t post identical tweets one after another.[2]

Since we wanted to repeatedly send our beacon Tweet this was an obstacle. We circumvented this issue by adding the time in days that had passed since
the collection begun to our beacon message. Since we wanted to send one Tweet per day, the number representing days since start of the collection would always increase by one day for every sending, hence the tweets would not be identical.

Our first beacon then mentioned one day, the second beacon mentioned two and so forth.

7.3.2 Collection of our own published beacons

A consequence of us posting beacons containing the same hashtag that we are tracking is that our posted beacons are added to our collections as well.

This poses a problem since our research material gets contaminated by our own posted beacons. To avoid this we added the id (hereafter id) of the account that we are using when posting the beacon. With the help of this id we added a condition to the functionality responsible of collecting tweets. The condition consists merely of an IF-case telling the software to not keep tweets that are sent from our own account with our id.

8 Conclusion

Despite having a good idea of what to implement on paper there appeared obstacles which will be tended to in the following. As a result there was a revised plan that resulted in work being done on functionality for pseudonymization and a possibility for a data subject to view his or hers data. Besides implementation, there have been some legal contributions. Even if the initial plan was to implement more of the functionality it is my hope that the work done can give inspiration, guidance and insights to those developing and working with tools who interact with Twitter.

8.1 Results and discussion

From a GDPR perspective the pseudonymization function is somewhat working but there are some problems with performance. Due to the extra time it takes to use different analysis parts of the program on a pseudonymized collection of tweets there is a risk that users of the tool will be reluctant to use the pseudonymization functionality, which would be contrary to the purpose of this thesis.

8.1.1 Performance problems

Although a proof-of-concept variant of pseudonymization functionality was developed early in the development process a considerable amount of time has been spent on trying to mitigate performance problems.

Table 2 contains measures for the first implementation of the added features, here no work has been done to mitigate the performance issues.
<table>
<thead>
<tr>
<th>Number of tweets</th>
<th>Export time for pseudonymizing the data and using the data to build a pseudonymization table</th>
<th>Export time for pseudonymizing the data without building a pseudonymization table</th>
<th>Export time for pseudonymizing the data using an already created pseudonymization table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 243</td>
<td>48.8 seconds</td>
<td>0.15 seconds</td>
<td>10.37 seconds</td>
</tr>
<tr>
<td>20 705</td>
<td>Over 5 minutes</td>
<td>2.63 seconds</td>
<td>N/A due to interruption of export while building the table</td>
</tr>
<tr>
<td>108 071</td>
<td>Over 5 minutes</td>
<td>13.624 seconds</td>
<td>N/A due to interruption of export while building the table</td>
</tr>
</tbody>
</table>

Table 2: Time it takes for a full export, with varying variants of pseudonymization, of bins containing 1 243, 20 705 and 108 071 tweets. When recording time the maximum time registered is 5 minutes.

One can see that the time spent for exporting and at the same time building a table is increasing very fast when the number of tweets are increasing. This indicates performance issues when extending the array containing the already done implementations.

Even if there has been some progress on the performance, there is still a troublesome duration depending on the pseudonymization functionality.

### 8.1.2 Reflections on performance efforts

Since the performance problem was thought of just being a minor error the focus was on "fast" rewrites of the pseudonymization code. These rewrites consisted mostly of movements of data type declarations and function calls in the code as an effort to find a small and obvious error.

These rewritings did however not result in any greater improvement. Another consequence of all the rewrites was a loss of the bigger picture coming with every rewrite. The ambition tended to start drifting from writing good software to just write a pseudonymization function that was more usable. After a couple of rewrites the insight that the problem was being bigger than a simple coding error aroused. This marked the start of a more structured, and rewarding trouble shooting.

One disadvantage with PHP is the lack of good working tools for profiling. The tools that do exist tend to be too demanding to install and use which makes the gained benefit of using these tools very small in comparison with just scanning the code manually line by line. Since there was a clear bottleneck when working with arrays compared to just changing values without any storing some
research on PHP arrays where done. Since PHP's only data type is the array the space of solutions is narrow, one cannot simply just change to a faster data structure since arrays are the only ones we can work with. Our pseudonymization table consists of three columns and to keep track of all the information in the columns a multidimensional array is used. The multidimensional array could, if unwisely used, be a performance killer since there is a lot of accessing on different levels in the array. After more reading up on the performance of PHP multidimensional arrays and a more meticulous planning of the data structure another rewriting of the software was made.

PHP support associative arrays with a key that is pointing at a value. An associative array is working like a hash table, where lookups occur in $O(1)$ time instead of linear. By restructuring the multidimensional table in a way such that all the lookups done was done in a way similar to how a hash table lookup is made some performance was gained.

Besides optimizing the use of the data structures there where also changes in how parameters were used. We achieved the best results by passing parameters by reference.

When pseudonymizing a Tweet there are multiple fields in the array constituting the Tweet that we are referencing. Every value in every field is pushed to the pseudonymization array. Since a collection of tweets can consist of extremely many tweets this means that the array carrying all the pseudonymization data grows a lot faster than one record per Tweet.

Since all our profiling efforts have shown that a big part of our problem is the amount of data we tried introducing parameter passing by reference. In PHP referencing means that instead of copying the data structure every time that it is passed we are sending an address to the very same data structure. This gave us some performance gains.

8.2 Lessons learned

During this process scrum has been used. There have been planned sprints consisting of 25 hours per week. Due to inexperience with PHP and software engineering in general the difference between anticipated time and actual time for one task has been as high as 30 hours. A lesson for further situations where the time tends to disappear is to have a clear upper limit of how much time one should spend trying to fix a task before getting help from a more skilled person. In short, a fixed timeframe for how long one should be trying by oneself before asking for help. This would prevent digging far deep into a problem and offer a fresh perspective on an earlier stage, something that could help the one getting stuck.
8.3 Future work

<table>
<thead>
<tr>
<th>Data subjects privilege</th>
<th>GDPR - legislation</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudonymization</td>
<td>article 6.(4)e, 25, 32.(1) a and 89(1)</td>
<td>Begun and almost done. However, there is some amount of work left to be done to get a version that is of practical use.</td>
</tr>
<tr>
<td>Subjects right to access</td>
<td>article 15</td>
<td>Done</td>
</tr>
<tr>
<td>Subjects right to rectification</td>
<td>article 16</td>
<td>Not done.</td>
</tr>
<tr>
<td>Subjects right to be forgotten</td>
<td>if it is not damaging scientific research done in accordance with 89(1). article 17(1) and article 17(3)d.</td>
<td>Not done.</td>
</tr>
<tr>
<td>Fair and transparent processing</td>
<td>recital 60 and The principle of fair and transparent processing</td>
<td>Done but only when we are tracking certain hashtags.</td>
</tr>
</tbody>
</table>

Table 3: The privileges we have worked towards and which of them that we have built and which of them that are left.

Since implementation of the intended functions halted with the pseudonymization there is a lot of future work to be done, work that can get a head start by follow up on the insights this thesis have produced.

First, a big part of the future work is to fix the speed of the pseudonymization functionality. Some suggestions regarding this have been discussed, but the main theories we have is that one solution should be to work with a pseudonymization table for each collection of tweets. Now every collection in the database shares the same bin which makes the pseudonymization table grow really fast.

The possibility to login with your twitteraccount and view how you are represented in a collector database is implemented. However that feature should be expanded with the possibility of deleting your information for our database.
References


