SOCNETO

Software project

supervisor: Doc. RNDr. Irena Holubová, Ph.D.

August 1, 2019

Contents

1 Introduction

For more than a decade already, there has been an enormous growth of social networks and their audiences. As people post about their life and experiences, comment on other people's posts and discuss all sorts of topics, they generate a tremendous amount of data that are stored in these networks. It is virtually impossible for a user to get a concise overview about any given topic.

Project Socneto offers a framework allowing the users to analyze data related to a chosen topic from given social networks.

1.1 High Level Description

Socneto is an extensible framework allowing a user to analyze the content across multiple social networks. It aims to give the user an ability to get a concise overview of a public opinion concerning a given topic in a user-friendly form based on data collected across social networks.

Social networks offer free access to data, although the amount is limited by number of records per minute and age of the post which restrict Socneto from downloading and analyzing large amounts of historical data. To adapt to these limitations, Socneto offers continuous analysis instead of one-off jobs. It continuously downloads data and updates the respective reports (see Figure 1).

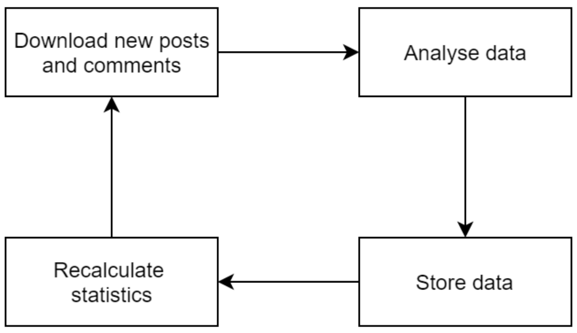


Figure 1: Endless pipeline

To prove the concept, Socneto supports selected types of data analyses, such as topic extraction and sentiment analysis supporting English and ~~Czech languages~~. However, additional types of analyses can be supplied by the user.

In terms of data acquisition, Socneto supports two main social networks: Twitter [1] and Reddit [2]. Both of them provide a limited free API used by default or an unlimited API for users who have paid accounts. Additional social network adapters can be also supplied by the user.

1.2 Use Case

Generally, a user specifies a topic of interest, selects type(s) of required analyses and social networks to be used as data sources. Socneto then starts collecting respective data, runs them through analyzers and stores the results. The user can then see the results either in a tabular version or visualized with customized charts.

A typical use case is studying sentiment about a public topic (e.g., traffic, medicine etc.) after an important press conference, tracking the opinion evolution about a new product on the market, or comparing stock market values and the general public sentiment peaks of a company of interest.

The framework runs *on premises* thus the user is responsible for handling security and connecting to storage compliant with GDPR rules, thus security of the system and a security of the data are both responsibilities of the user.

1.3 Architectural Overview

Socneto is designed to be a platform processing data from various sources focusing on data from social networks. Data themselves have a little value for the end user if they are not analyzed and visualized properly. To reflect these priorities, layers of the project architecture can be visualized as depicted in Figure 2.[1]

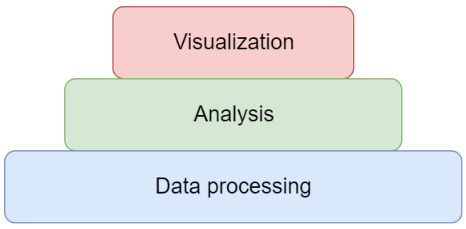


Figure 2: Conceptual separation of application responsibilities: to develop the data processing platform, to analyze the data, and to present them to the user

The backbone part of this project is the *data processing platform* responsible for data acquisition, data storage and cooperation among all components to successfully deliver any results to the user.

In order to interpret acquired data correctly an *analysis* is performed. There are many possible approaches how to analyze data from the Internet, thus this part has to be extensible by the user to fit his/her needs.

The analyzed data are then presented to the user in a concise form supported by *visualizations* and sample data.

The requirements stated above are in Socneto ensured by various cooperating modules. They are connected together forming a pipeline with modules dedicated to data acquisition, analysis, persistence and also a module managing the pipeline's behavior (see Figure 3).

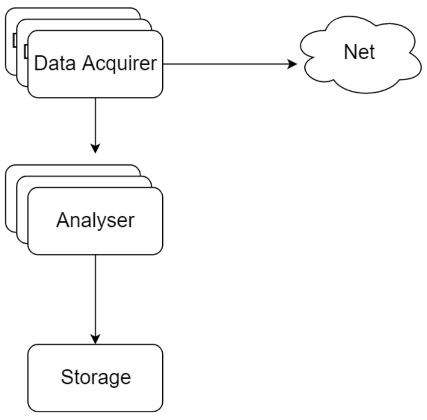


Figure 3: A simplified version of the pipeline processing data

[1]We are aware of the fact that the customer might think otherwise.

2 Planning

The following section offers and insight into the team composition, responsibilities of members and project milestones.

2.1 Team

The list of team members and their responsibilities is provided in Table 1.

Name	**Responsibilities**
Irena Holubová	Supervisor
~~Jan Pavlovský~~	~~Machine learning engineer, software engineer – builds the platform with the focus on machine learning integration~~
Petra Vysušilová	Machine learning, linguistic specialist – develops the sentiment analysis model
Jaroslav Knotek	Software engineer – designs and builds the platform
Lukáš Kolek	Data engineer – designs and develops the data storage
Július Flimmel	Web engineer – builds the web application and front end

Table 1: Team members and their responsibilities

2.2 Development Process

The development follows agile practices. In the beginning, the team meets every week to cooperate on an analysis and an application design.

Once the analysis turns into specification and is defended, the team is divided into two groups cooperating on separate parts focusing on a data platform and machine learning reflected in analyzers.

The best results are achieved when the team works together. The cooperation will be encouraged by several all-day-long workshops when the whole team meets at one place personally in order to progress.

The process of development is divided into three approximately equally long phases. The first one focuses on creating the proof-of-concept (PoC) making sure that project's assumptions are not false. In the second phase, the team focuses on setting up proper storage, implementing advanced features such as sentiment analysis or proper result visualization and, last but not least, implementing mocked parts properly. The last part consists of creating deployment guide, writing documentation and testing.

The described plan is visualized in Figure 4 which shows three phases with key milestones. Greyed parts represent the already finished milestones. The following sections will describe each phase in detail.

2.2.1 Proof-of-Concept

The application relies heavily upon asynchronous communication between all components. To prove that the idea is plausible, communication should be tested by mocked components configured to communicate with given components using Kafka [3] and random data. It runs on infrastructure which consist of several virtual machines provided by the Charles University.

This phase should prove that the idea is plausible. At the beginning, the test features only a testing data acquisition component and a testing analyzer. But as the development advances, they are replaced with a production version and other types components are connected as well.

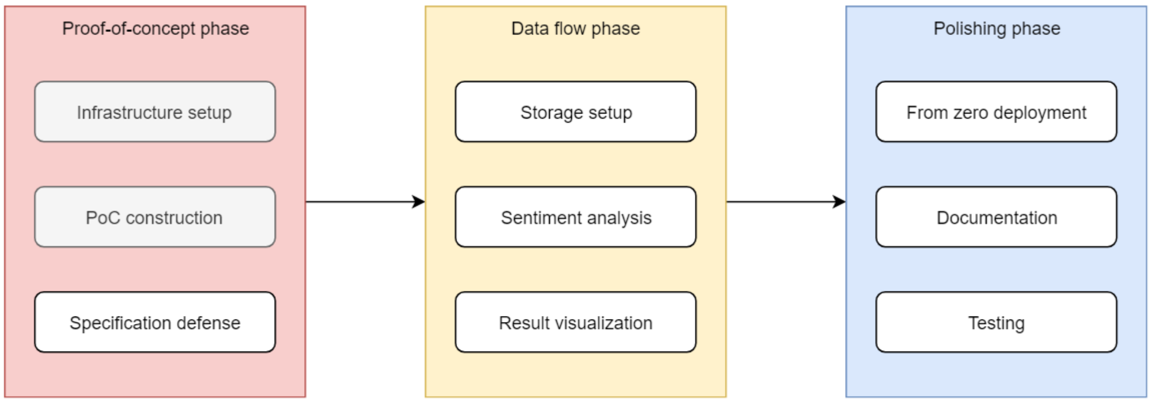


Figure 4: Project development phases

The product of this phase is a working simplified platform supporting a basic data flow and a final specification reflecting the gained experience. The platform is capable of coordinating all components which is necessary to support basic user interaction, e.g., submitting job and querying results.

This PoC was the responsibility of Jaroslav Knotek and Lukáš Kolek. At the same time, samples of front-end and analysers were developed by Július Flimmel and Petra Doubravová respectively. This phase is already finished. The next step is to implement proper data flow and implement all mocked components.

2.2.2 Data Flow

As the platform stabilizes, more focus is put to proper data acquisition, storage and querying. When the user submits a job, all components have to cooperate in order to deliver the expected results.

At this point, storage is built to store all data, e.g., data from social networks and application data needed for a proper job execution. It will be followed by proper implementation of the data acquisition component downloading data and feeding them to the analyzer. This will be the responsibility of Jaroslav Knotek and Lukáš Kolek.

The sentiment analyzer is expected to be the most complex and the most risky part of the whole project. It will require great deal of effort to develop, test and finally integrate it into the framework. This will be supervised by Petra Doubravová ~~and Jan Pavlovský~~.

At this point, the first results will start to emerge. We will start collecting data and start creating an overview based on real-world data. The result visualization will be developed by Július Flimmel.

2.2.3 Finishing and Polishing

The last phase focuses on extensibility and deployment, as well as improving precision of the supported analyzers. The application will be extended by the other data acquisition component connecting to Reddit and an analyzer covering simple topic extraction.

Once all the components are ready, the project will start to finalize with testing sessions that will focus on technical aspects of the framework and user interaction. At this point we will find a real-world partner that will supply us with his data in exchange for our services. ~~At this time we are communicating with a few community service associations such as “Hlídači státu” or “Milion chvilek pro demokracii” to whom we want to offer our services in order to test Socneto in real-world scenarios.~~

We will offer our services to Mff public relations department and we are seeking at least one other partner.

3 Supported Analyses

As a part of this project, two linguistic tasks will be implemented:

- Topic modeling and
- Sentiment analysis.

The intention of Socneto is not to implement a completely new linguistic model, but to re-use existing third-party packages and customize them. The used machine learning methods require labeled training and testing gold data. This is a problem especially for the Czech language . Particular available datasets are described below in the respective sections. We do not assume annotation of new data. For this reason and also due to the difficulty of topic modeling, we do not expect human-like results. As was said before, Socneto will be easily extensible with other types of analyses or their implementations if needed.

Both tasks involve the following steps (see Figure 5):

1. Preparation (gathering and pre-processing) of training and testing data,
2. Integration and customization of existing packages,
3. Training on train data
4. Measuring a performance, and
5. Changing meta-parameters of model for better accuracy.

Python library NLTK [4] is used for data pre-processing, such as tokenization and stemming.

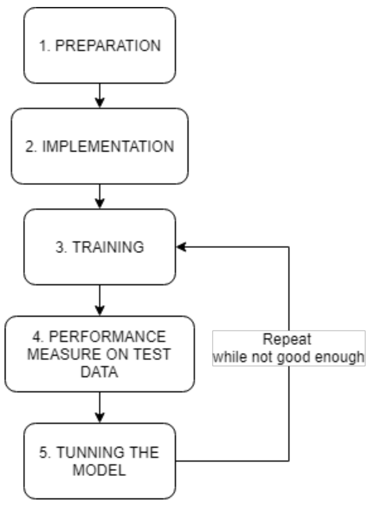

1. PREPARATION
2. IMPLEMENTATION
3. TRAINING
4. PERFORMANCE MEASURE ON TEST DATA
Repeat while not good enough
5. TUNNING THE MODEL

Figure 5: Machine learning workflow

3.1 Topic Modeling (TM)

This is generally a difficult problem for both supported languages (i.e., Czech and English), because it is not clear what is the required result. Also some language-specific rules are too complex to be solved aromatically yet. In addition, e.g., Twitter data are typically short, not consisting of whole sentences, involving a mixture of languages and emoticons, breaking grammatical rules, etc. So classical approaches to this task will probably not work very well.

Possible approaches together with their disadvantages are as follows:

- Word frequency + dictionary – an easy approach, but it cannot handle, e.g., pronoun references
- Entity linking – a knowledge base is needed
- Entity modeling – a knowledge base is needed
- Probabilistic models (Latent Dirichlet Allocation = LDA, or Latent Semantic Analysis = LSA)

In the case of the English language it will be implemented using package Gensim [5] created by Natural Language Processing Centre (NLP Centre) of the Masaryk University. LSA, LDA and frequency analysis will be performed and the better one or the combination of them will be selected.

Since there is not so much data for training this task in the Czech language, so, as mentioned above, in this case we do not expect expert results. Topic modelling dataset extracted from Wikipedia articles is actually communicated with already mentioned NLP Centre [6] as well. For the English language there exist much more suitable datasets, such as the Kaggle 'A Million News Headlines' dataset [7].

In our particular case, the topics for hierarchical structures like posts + their comments must be found for posts and related comments together. The result of this kind of analysis is a set of key words related to the post/post+comments structure.

3.2 Sentiment Analysis (SA)

Sentiment analysis is a quite subjective task. The output of this analysis is in the first place polarity – negative, positive, neutral – and probability of each of the categories. Sometimes subjectivity analysis is viewed as a part of sentiment analysis. Subjectivity analysis output could be a measure of building opinion on facts or personal feelings. The second possible approach directly finds emotions and opinions of the text author [8]. Subjectivity in the second sense of word will not be implemented as a part of Socneto. The first one in the form of the number on a selected scale can be implemented for the English language, as it is a part of existing libraries.

All models for this part are based on the Bidirectional Encoder Representations from Transformers (BERT) [9], and the last layer must be implemented according to the task and trained as specified in [10]. According to NLP-progress page [11], actual state-of-the-art accuracy of this task on English datasets in the case of binary classification is above 95%. For the Czech language it is above 70% [12].

There exists a number of datasets directly based on social network posts in English. For the Czech language there exist one dataset consisting of about 10,000 annotated Facebook posts [13], a corpus of CSFD reviews, and Mall reviews [14].

In general, for our particular purposes the current progress in this field is continuously consulted with experts from UFAL[2].

[2]`https://ufal.mff.cuni.cz/home-page`

4 Platform Architecture

The platform is separated into three different tiers:

- Data processing tier,
- Analyzers, and
- Front-end (visualization).

Each tier consists of several services responsible for a sub domain. The used service-oriented architecture improves separation of concern and also allows us to implement each service in different language that fits the problem the most. For example, data processing services are implemented in well-established enterprise-level languages such as Java or C#. On the other hand, analyses are implemented using less strict languages, such as Python, offering a number of libraries related to our case.

The *data processing tier* consists of services acquiring and storing data, cooperation of all components, system health and data processing pipeline management. *Analyzer tier* consist of various services responsible for analyzing data of a given format and are expected to produce data of a given format. The *front end* serves the user to understand the results preferable at first sight.

The backbone of the whole pipeline is to acquire data that the user requests by submitting a job definition containing a query, selected analyzers and selected social networks. The platform translates it to tailored configuration made uniquely for each component.

The design of the whole framework requires a lot of effort in order to avoid common pitfalls of school projects. The main possible weak spots would be communication among multiple services, customization and extensibility, and, last but not least, testing.

4.1 Acquiring Data

The API limits of social networks restrict the amount of data being downloaded in a given time interval. For example, Twitter standard API limits [15] allow connected application to look-up-by-query 450 posts or access 900 post by id in 15 minutes long interval. Reddit has less strict limits allowing 600 request per 10 minute interval. These limits must be reflected in each data acquirer.

Both APIs restrict free users from downloading large amounts of data and also it does not allow an access to data older than 7 days. Even though the limits are very restricting, continuous analysis saves the day – it downloads small portions of the up-to-date data, analyses them and immediately updates the results.

A request for data acquisition contains information about requested data in a form of a query and optionally credentials if the user does not want to use default one. Then the data acquirer starts downloading data until the user explicitly stops it.

The output of each data acquirer follows the system-wide format of unified post data (UPD) that features:

- text,
- creation or modification time,
- link to a related post (in case of a comment or re-tweet), and
- user id.

Socneto will support acquiring data from Twitter and Reddit with the use of 3rd party libraries LinqToTwitter [16] and Reddit.Net [17] respectively. Both of them will make it easier to comply with API limits and tackle respective communication (the limits must be reflected in each component itself). Each library has its own dedicated service and both of them will be written in C#.

4.2 Analyzers

Each type of analysis resides in its own service. The input format is the same as output of data acquisition services. The analyzers will process the data received from the acquirers. After analyzing each post they send the result to the storage. Since the results of the analyzers are the used by front end, the analyzers' output will need to be in a standardized form. As this output will be mainly processed by a computer, we decided to use the JSON format.

The structure of the JSON documents will need to be robust and simple enough, so that the user of front end may easily specify which data (s)he wants to visualize using JSONPath. The structure of the output is the following:

```
{
    "analyzer_name": {
        "analysis_1": { "type": "number", "value": 0.56 },
        "analysis_2": { "type": "string", "value": "rainbow" },
        "analysis_3": { "type": "[string]", "value":
                        ["friends", "games", "movies" ] }
    }
}
```

The whole analysis is packed in one object named after the analyzer. As the analyzer may compute multiple analyses at once, each one will be also represented by one object named after the analysis. The object representing the analysis has a strict format. It contains two attributes:

- *type*, specifying the type of the result. The supported types will be *number* (including integers and floating point values), *string*, and *lists* of these two. Lists of lists will *not* be supported,
- *value*, i.e., the actual result of the analysis.

There may be multiple analyzers in our framework, so all their outputs are inserted into one analysis object. We do not use arrays of objects but named objects instead, so that the user may easily specify an analyzer and an analysis in JSONPath when creating a new chart in the front end.

Here we provide an example of a post's complete analysis. It contains analyses from two analyzers: *keywords* and *topic*. Keywords analyzer returns one analysis called *top-10* where the values are the found keywords in the post. The topic analyzer returns one analysis, *accuracy*, specifying to what extent the post corresponds to the given topic.

```
{
    "keywords": {
        "top-10": {
            "type": "[string]",
            "value": [
                "Friends", "Games", "Movies", ...
            ]
        }
    },
    "topic": {
        "accuracy": {
            "type": "number",
            "value": 0.86
            }
        },
}
```

Implementation of the types of analyses requires different amount of effort but integrating them will cost the same. Both of these services will be written in Python and will require an additional effort to integrate them into the system.

4.3 Communication

Each service runs in a separate docker container. Packing services into docker containers makes deployment easier since all required dependencies are present inside the container. Containers (except for the front end and respective back end) communicate using a message broker Kafka which allows for high throughput and multi-producer multi-consumer communication.

In our case, the data can be acquired from multiple data sources at the same time and sent to multiple analysis modules. Data processing is difficult to cover by the request/response model. More elegant and simpler solution is to use the publish/subscribe model which Kafka supports.

The services responsible for acquiring the data (*producers* in Kafka terminology) produce data to a given channel that are delivered to all services that are listening on this channel (*consumers*). It also keeps a track of which message was delivered and which was not delivered yet, so if any component temporarily disconnects, it can continue at work once the connection is up again.

Another benefit of message broker is that particular services are not aware of a sender of its input data and of receiver of its output respectively. It makes it easy to configure the data flow.

4.4 Cooperation

The main pitfall of asynchronous communication is the lack of feedback from a receiver. More specifically the sender is not sure whether anyone is actually listening. In order to tackle this problem, the Job Management Service (JMS) was introduced.

Job management is responsible for proper cooperation of all components. JMS is the component that each component must register to before the system starts working. When the user submits a job, JMS transforms it to multiple tailor-made requests for all involved components. Each request contains a configuration that influences which data belonging to a given job will be redirected to each output.

For example, if the user submits a job A that requires sentiment analysis of data from Twitter and Reddit with respective paid account credentials, the required types of analyses are topic modeling and sentiment analysis. JMS delivers job configuration with Twitter credentials only to Twitter data acquiring service, Reddit credentials to Reddit data acquiring service, and directs the output of data related to the given job to both analyzer services. The situation might get complicated when another job B is submitted requiring only data from Twitter and a topic modelling. JMS will configure the selected components to handle data of the given job differently without affecting the already running job. (The data routing of the parallel jobs A and B is visualized in Figure 6.)

To make sure that components are up an running, a system monitoring will be implemented (see Section 5).

4.5 Data

The data store is designed for running on a different machine without a public network connection for better security, different technical requirements for machines, and possible scalability.

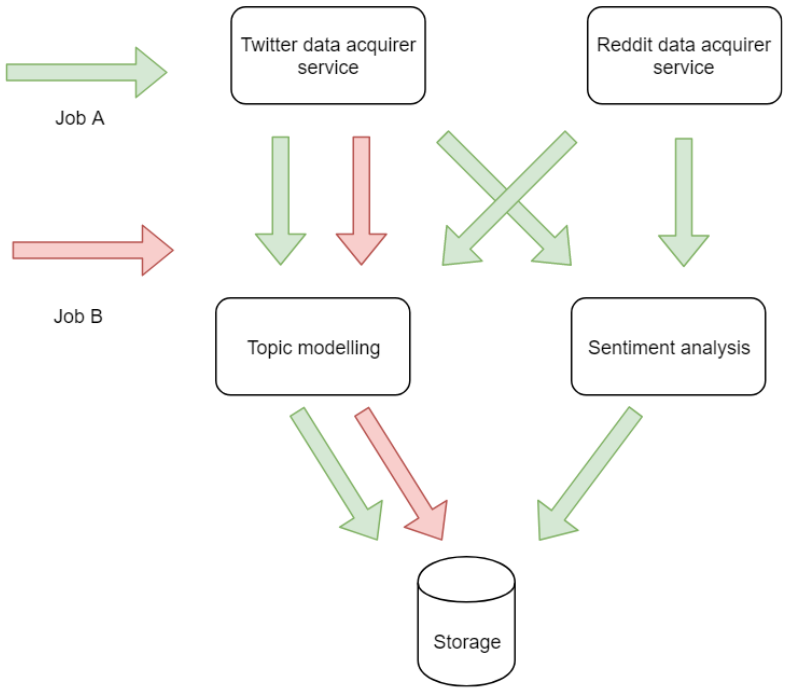


Figure 6: Two jobs running in parallel

Behind a storage interface there are several databases for different purposes and the interface is created for a transparent communication with all databases. More databases were chosen for better scalability and partitioning data by their usage (internal application data vs analyzed posts).

4.5.1 Storage Components

The storage architecture (see Figure 7) consists of the following components:

- Database interface
  - This component forms an abstraction over an implemented polyglot persistence for the rest of the platform. It consists of a *public* API for communication with the web application back end and Kafka client for receiving analyzed posts from analyzers and internal clients for databases.
- Relational database
  - Usage: internal data such as users, jobs, logs, configurations etc.
  - Requirements: Storage of relational data with possible JSON fields
  - Used implementation: PostgreSQL[3]
- NoSQL database
  - Usage: internal data such as users, jobs, logs, configurations etc.
  - Requirements: NoSQL storage
  - Used implementation: MongoDB[4]

[3]`https://www.postgresql.org`
[4]`https://www.mongodb.com`

- Search engine
  - Usage: searching in anayzed posts
  - Requirements: full-text search
  - Used implementation: Elasticsearch[5]

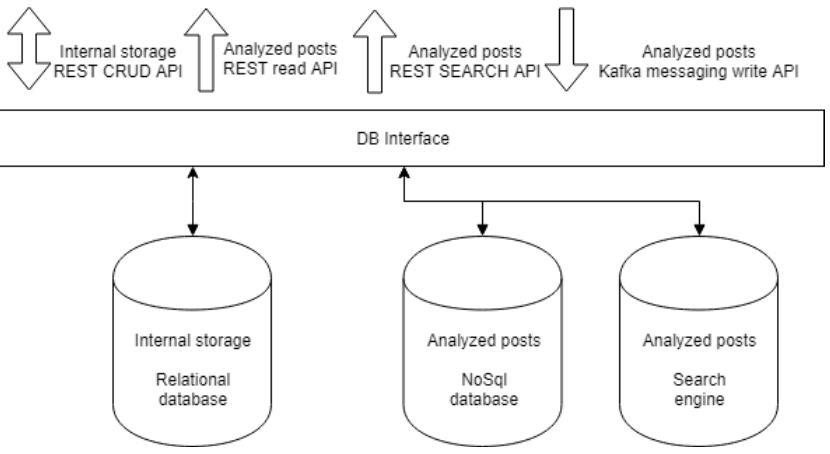

Internal storage REST CRUD API
Analyzed posts REST read API
Analyzed posts REST SEARCH API
Analyzed posts Kafka messaging write API
DB Interface
Internal storage Relational database
Analyzed posts NoSql database
Analyzed posts Search engine

Figure 7: Storage architecture

Note that neither any of the DBMSs nor the search engine will be hard-coded and thus it will be possible to replace any of them with a different component. Only the client interface needs to be implemented.

Also note that it is more common not to use Elasticsearch as a primary database. In the latest versions Elastic improved and removed some berries for this approach. But we decided to implement the storage in the "traditional" way and duplicate data in Elasticsearch.

4.5.2 API

- REST CRUD API for internal storage
- REST Read API for analyzed posts
- REST Search API for analyzed posts
- Write Kafka message listeners

4.5.3 Entities

The main expected entities with mandatory fields (depicted in UML in Figure 8) are as follows:

- **User:** userId, password
- **Job:** id, userId, job config, timestamp
- **Log:** componentId, timestamp, status, message, additional data
- **Post:** id, jobId, original post with additional info, list of analysis

[5]`https://www.elastic.co/products/elasticsearch`

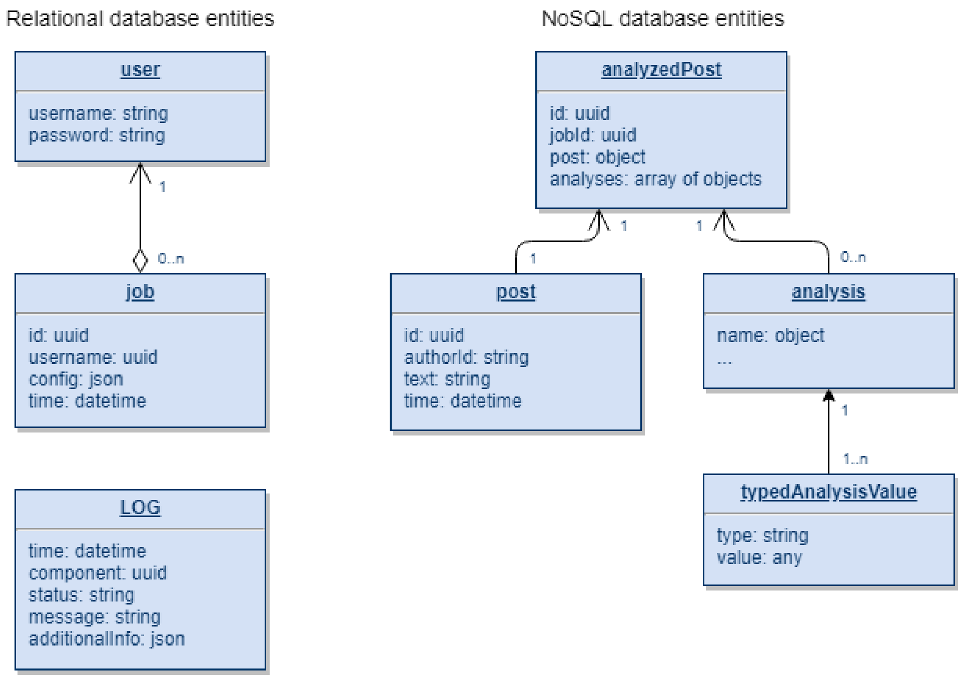


Figure 8: Database schema

5 System Management and Interface

The application architecture is loosely coupled, which brings a lot of possible orchestration problems. The platform should contain a component, which is responsible for collecting logs from analyzers, acquirers and possibly data storage. As the platform is already using Elasticsearch, an ELK stack[6] is efficient for this propose. Logstash[7] is responsible for collecting logs from the application, for communication the already mentioned Kafka is used. Elasticsearch is needed for storing logs and Kibana[8] for visualization. Logs will be also persisted in the relational database.

It is expected that only a technical user – admin – will have permissions for Kibana dashboard with logs. On the other hand, basic statuses of running jobs will be also displayed for a user in Socneto UI.

5.1 API

The back end API works using the HTTP protocol and the REST architecture. It works as a (protection/access) layer between the front end and the storage modules of Socneto. It provides an access for the authorized users to their jobs, their results and visualization definitions.

5.1.1 Authentication and Authorization

To provide protection of the data, almost each API call requires the user to be authenticated and authorized for their usage. These calls expect *Authorization* HTTP header to be filled. As perfect data protection is beyond the scope of this work, we use only *Basic authorization* for simplicity. After authenticating the user, the back end also verifies whether the user is authorized to read the data he is requesting. If not, a message with HTTP status code *401 (Unauthorized)* is returned. If the user is authorized, the requested data is returned.

For front end to be able to implement login, the back end contains login call. This call expects username and password in body and returns the user object which corresponds to the given credentials. If there is no such user, the status code *400 (Bad Request)* is returned.

5.1.2 Jobs

The API provides calls to request the list of user's jobs, their details and their results. Only authorized users are able to see the data. The user is able to access only those jobs (including their details and results) (s)he has submitted.

The API also contains an endpoint for submitting a new job. This endpoint expects correct job definition, otherwise it returns the HTTP status code *400 (Bad Request)*. The correct definition contains title, topic, non-empty list of data acquirer components to be used, non-empty list of data analyzer components to be used, and the number of posts to fetch (if not unlimited). *JobId* is returned if the job was successfully created.

5.1.3 Visualization Definitions

The user needs to be able to store and load the definitions of his visualizations (see Section 5.2.6). The back end provides API to list already defined visualizations of user's job and to define a new one.

[6] `https://www.elastic.co/products/elastic-stack`

[7] `https://www.elastic.co/products/logstash`

[8] `https://www.elastic.co/products/kibana`

5.2 Front End

The primary purpose of the front end is to provide the user with a tool to configure and look at multiple different visualizations of the analyzed data. The application will also allow the user to specify and submit new jobs to the platform and will inform him/her about their progress. The last functionality allows administrators to manage and configure individual components of the whole framework.

5.2.1 DartAngular + Material

We chose to implement the front end as a web application to develop a cross-platform software which is user-friendly and easily available. We chose to use modern and widely used style guidelines / library *Material Design* to quickly build a nice and professional looking product. We stick with *DartAngular*, because its library provides us with angular components[9] which already use Material Design.

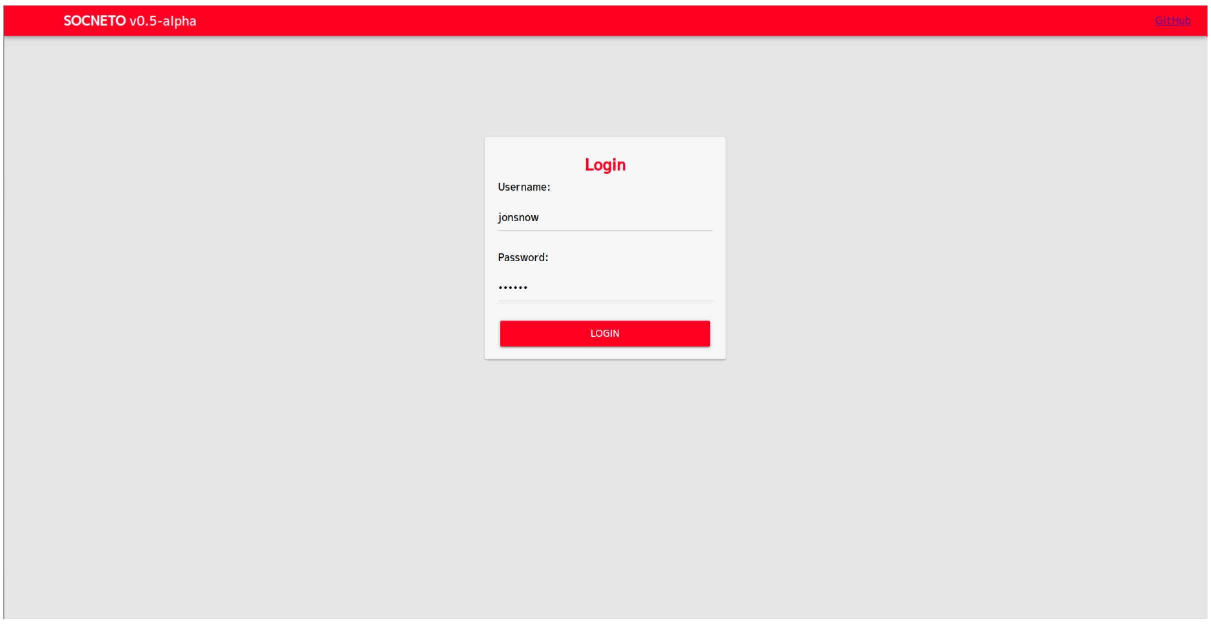
5.2.2 Components

Angular uses a component-based architecture, so each page is composed of multiple components. In this section we provide a description of the views the user can encounter and the components they are made from.

5.2.3 Login

As mentioned in the API section, the back end requires the user to be authorized for most requests. Therefore the user needs to login (see Figure 9) before (s)he starts using the application. After signing in with correct combination of username and password, the user is redirected to his/her dashboard. The credentials are also stored in the local storage, so the user does not need to insert them on each page reload.

If a user tries to access the content to which (s)he is not authorized (i.e., receives the HTTP result with status code *401*), (s)he is immediately redirected to a special page, where (s)he is informed about it. From there, (s)he can try visit different content or sign in again.

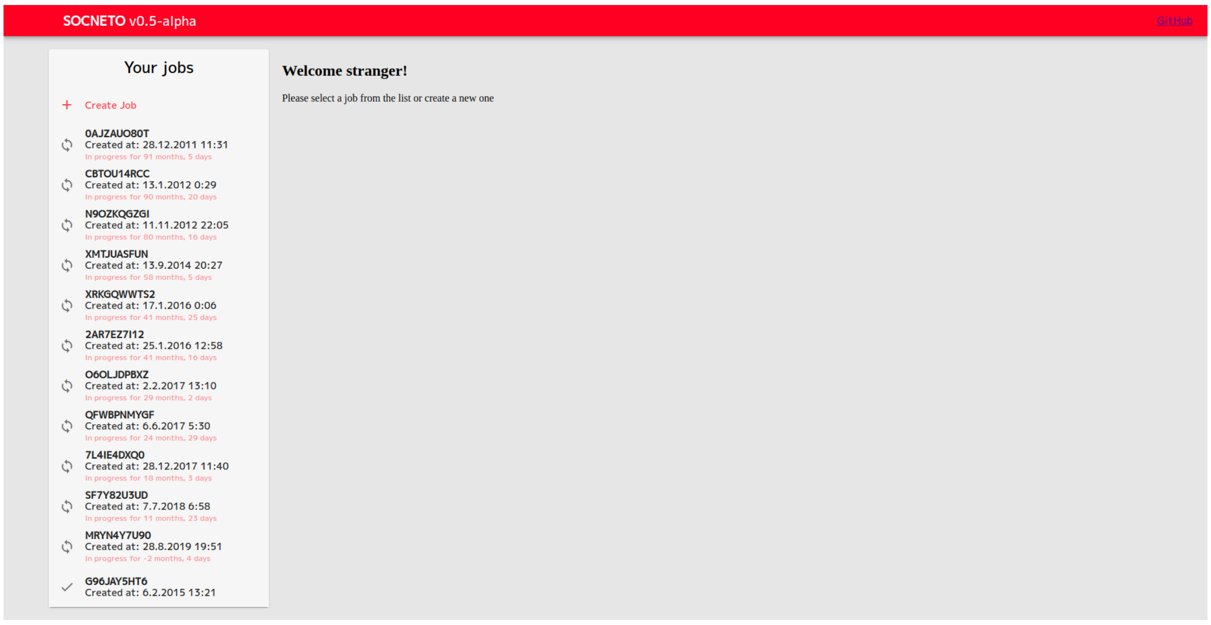


Username: jonsnow

Password: ······

Figure 9: Login page

[9]`https://dart-lang.github.io/angular_components/`



0AJZAUO80T
Created at: 28.12.2011 11:31
In progress for 91 months, 5 days

CBTOU14RCC
Created at: 13.1.2012 0:29
In progress for 90 months, 20 days

N9OZKQGZGI
Created at: 11.11.2012 22:05
In progress for 80 months, 16 days

XMTJUASFUN
Created at: 13.9.2014 20:27
In progress for 58 months, 5 days

XRKGQWWTS2
Created at: 17.1.2016 0:06
In progress for 41 months, 25 days

2AR7EZ7I12
Created at: 25.1.2016 12:58
In progress for 41 months, 16 days

O6OLJDPBXZ
Created at: 2.2.2017 13:10
In progress for 29 months, 2 days

QFWBPNMYGF
Created at: 6.6.2017 5:30
In progress for 24 months, 29 days

7L4IE4DXQ0
Created at: 28.12.2017 11:40
In progress for 18 months, 3 days

SF7Y82U3UD
Created at: 7.7.2018 6:58
In progress for 11 months, 23 days

MRYN4Y7U90
Created at: 28.8.2019 19:51
In progress for -2 months, 4 days

G96JAY5HT6
Created at: 6.2.2015 13:21

Welcome stranger!

Please select a job from the list or create a new one

Figure 10: Dashboard

5.2.4 Dashboard

The dashboard (see Figure 10) displays the history of all jobs the user ever submitted including a simple information about them (i.e., name, submission date, etc.). After selecting a specific job, a component with more details about the job and data visualizations is shown. The list of jobs also contains a button which shows a component for submitting a new job.

5.2.5 New Job

This component aims to an easy and user-friendly ability to submit a new job (see Figure 11). We use Material input components to provide the user with the best UX. The user can specify the *name* of the job, the *topic* to be searched for, which registered Socneto *components* to use (analyzers and data acquirers), and the number of posts to fetch, or unlimited.

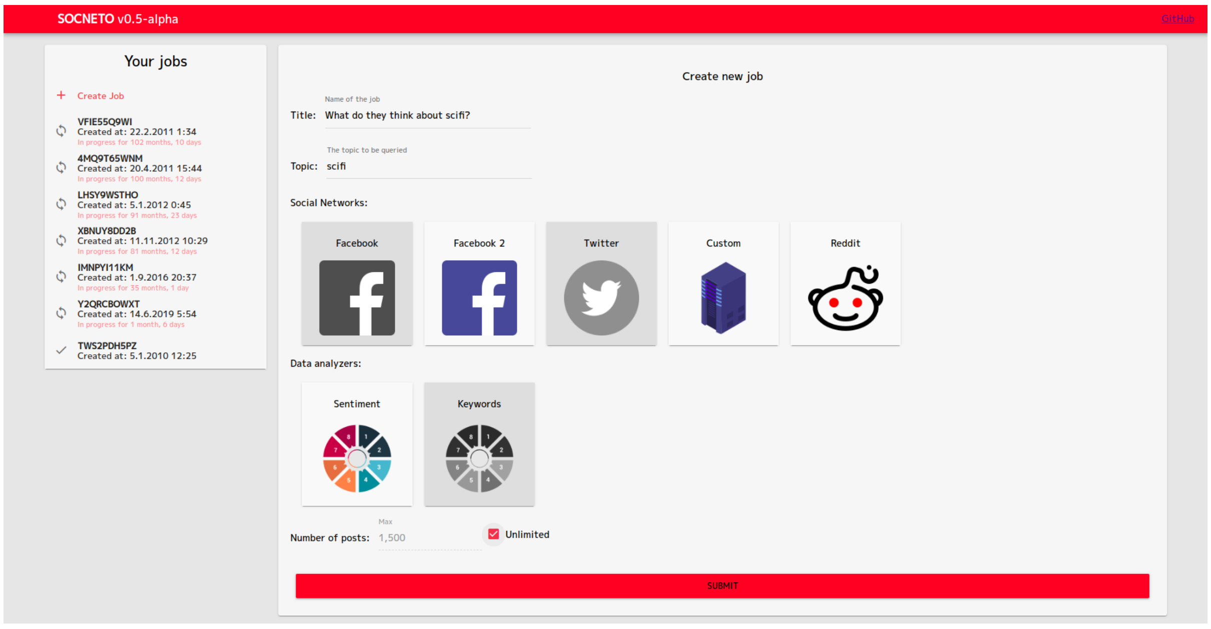
5.2.6 Job Detail

The job detail component contains the list of user-specified visualizations of the analyzed data. It also contains a paginated list of all acquired posts and their analyses.

At first, the component contains no visualizations. The user has to specify which data from the analyses (s)he wants to visualize. This approach gives the user a degree of freedom instead of being presented by fixed charts. When creating a new chart, the user only has to select the type of chart (pie chart or line chart), and write a JSONPath to the attribute to be visualized. The definitions of charts are then stored in the storage, so the user does not need to create them each time.

5.3 System Health

A system that consist of multiple components is hard to maintain. In order to simplify investigation of failing services and to speed up recovery process, Socneto will store and manage all metrics using the ELK stack. Proper metric tracking requires definition of system wide guidelines of what values will be tracked and what alarms should be fired when components are malfunctioning. Integration of this system requires its installation on our infrastructure and integration in each service.


SOCNETO v0.5-alpha

GitHub

Your jobs

+ Create Job

VFIE55Q9WI
Created at: 22.2.2011 1:34
In progress for 102 months, 10 days

4MQ9T65WNM
Created at: 20.4.2011 15:44
In progress for 100 months, 12 days

LHSY9WSTHO
Created at: 5.1.2012 0:45
In progress for 91 months, 23 days

XBNUY8DD2B
Created at: 11.11.2012 10:29
In progress for 81 months, 12 days

IMNPYI11KM
Created at: 1.9.2016 20:37
In progress for 35 months, 1 day

Y2QRCBOWXT
Created at: 14.6.2019 5:54
In progress for 1 month, 6 days

✓ TWS2PDH5PZ
Created at: 5.1.2010 12:25

Create new job

Name of the job
Title: What do they think about scifi?

The topic to be queried
Topic: scifi

Social Networks:

Facebook · Facebook 2 · Twitter · Custom · Reddit

Data analyzers:

Sentiment · Keywords

Max
Number of posts: 1,500 ✓ Unlimited

SUBMIT

Figure 11: New job

6 Risk Analysis

In order to mitigate the impact of unforeseen situations, a PoC was build to prove that the project analysis is plausible. Although the PoC successfully works, several possible weak spots were not covered.

The most important part of the project is data which might be impossible to acquire in the future. Due to several incidents related to social network data analyses, public APIs were restricted. For example, it is harder to get historical or user specific data. At this point, the situation is manageable using workarounds such as continuous analysis but this situation might change at any point. We would have to focus on obtaining data differently (e.g., using available libraries of sample social network data) or use different social networks that has not limited the data yet.

Secured access to data is not the only possible concern, it is also the data quality. In order to accurately perform sentiment analysis, high quality data are required. Data from social networks may have several short-comings such as incorrect grammar or they may be very short affecting its accuracy. Insufficient data quality will require us to present to the user his/her results with a metric that will state reliability of the sentiment. Another possibility is to introduce data pre-processing which may negatively impact accuracy of the results.

The data itself is not the only crucial point. The project implements asynchronous communication used by Kafka which was tested on a very small amount of data that worked perfectly on a single node Kafka solution. If the amount of data increases the communication might start to reach limits of the throughput and the system might get overloaded. This can be solved by adding nodes to Kafka increasing the throughput and using different data format, e.g., to replace JSON with a binary format, such as Protocol buffers of Avro.

List of Figures

List of Tables

References

[1] "Twitter." [Online]. Available: https://www.twitter.com/

[2] "Reddit." [Online]. Available: https://reddit.com

[3] "Kafka." [Online]. Available: https://kafka.apache.org/

[4] "Natural language toolkit." [Online]. Available: https://www.nltk.org/

[5] "Gensim." [Online]. Available: https://radimrehurek.com/gensim/

[6] "Natural language processing centre of masaryk university." [Online]. Available: https://nlp.fi.muni.cz/cs/CentrumZpracovaniPrirozenehoJazyka

[7] "A million news headlines." [Online]. Available: https://www.kaggle.com/therohk/million-headlines/data

[8] A. Montoyo, P. Martinez-Barco, and A. Balahur, "Subjectivity and sentiment analysis: An overview of the current state of the area and envisaged developments." *DECISION SUPPORT SYSTEMS*, vol. 53, no. 4, pp. 675 – 679, n.d. [Online]. Available: https://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds

[9] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[10] "Sentiment analysis with bert." [Online]. Available: https://medium.com/southpigalle/how-to-perform-better-sentiment-analysis-with-bert-ba127081eda

[11] "Sota sentiment analysis." [Online]. Available: http://nlpprogress.com/english/sentiment_analysis.html

[12] K. Veselovská, *Sentiment analysis in Czech*, ser. Studies in Computational and Theoretical Linguistics. Praha, Czechia: ÚFAL, 2017, vol. 16.

[13] "Facebook data for sentiment analysis." [Online]. Available: https://lindat.mff.cuni.cz/repository/xmlui/handle/11858/00-097C-0000-0022-FE82-7

[14] "Sentiment analysis resourcees." [Online]. Available: http://liks.fav.zcu.cz/sentiment/

[15] "Rate limits." [Online]. Available: https://developer.twitter.com/en/docs/basics/rate-limits

[16] "Linq to twitter." [Online]. Available: https://github.com/JoeMayo/LinqToTwitter

[17] "Redit.net." [Online]. Available: https://github.com/sirkris/Reddit.NET