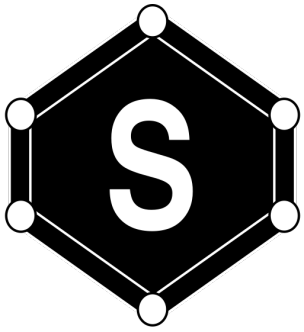
SOFTWARE PROJECT

Bc. Július Flimmel, Bc. Jaroslav Knotek, Bc. Lukáš Kolek, Bc. Petra Vysušilová

Scent of Current Network Overview



Socneto

Prague 2020

Project name: Scent of Current Network Overview

Authors: Bc. Július Flimmel, Bc. Jaroslav Knotek, Bc. Lukáš Kolek, Bc. Petra Vysušilová

Supervisor: doc. RNDr. Irena Holubová, Ph.D., Department of Software Engineering

Abstract: The aim of the project is to develop a framework for analysis of social network data. It provides users with functionality to analyse sentiment of large amounts of data and visualize the results. Data are acquired via social networks public APIs or custom modules.

We would first like to thank to our advisor, Doc. RNDr. Irena Holubová, Ph.D. She devoted a lot of time to to our project and provided us valuable advices and support. We would also like to thank to RNDr. Jakub Yaghob, Ph.D. for his willing help with school clusters. We are grateful Datamole provided its space for our meetings and coding sessions, which helped us to move a lot. Last but not least our thanks belong to Bc. Jiří Suchomel, who suggested real use cases for Socneto and also connected us with other people, which can use Socneto for their use cases.

Contents

Chapter 1

Introduction

The document covers the development phase of the software project. Documentation starts with a comparison of Socneto and other social network analysis tools, then there are described real-world use cases, followed by architecture and its components, topic modeling and sentiment analysis development description, platform extensibility and the final discussion.

1.1 Team

Bc. Jaroslav Knotek

- Subject of study: Software and Data Engineering
- Socneto components: Acquirers, Job Management service, Analysers, Architecture and Use Cases

Bc. Julius Flimmel

- Subject of study: Computer Graphics and Game Development
- Socneto components: Frontend, Backend and Deployment

Bc. Petra Vysušilová

- Subject of study: Artificial Intelligence
- Socneto components: Sentiment analysis, Topic modeling, LaTeX master

Bc. Lukáš Kolek

- Subject of study: Software and Data Engineering
- Socneto components: Storage, Monitoring and Architecture

Chapter 2

Project Overview

Socneto is a framework for analysing data from social networks for a given topic. Socneto can be extended with custom data acquirers and custom analyser. Socneto supports two major social networks: Twitter[1] and Reddit[2]; and two data analysers of key topics and sentiment supported in the current version of Socneto.

2.1 Introduction

This chapter opens with Section 2.2 which introduces the problem that Socneto tackles. The following Section 2.3 then compares Socneto with products focusing on either social networks or data processing. Section 2.4 describes achievements of the Socneto in the real world. At the end of this chapter, Section 2.5 traces the project development process.

2.2 Problem Analysis

This section introduces what solution Socneto offers to the major problems concerning social networks and data analysis tools. Each of the problem and its solution is discussed separately. The problems are the following:

- A content personalization
- Abundance of data
- Social networks API limits
- Limited customization

[1] `www.twitter.com`
[2] `www.reddit.com`

2.2.1 A Content Personalization

Social networks, such as Twitter and Reddit, allows users to easily subscribe to a content producer or to a group interested in some topic. This personalization may result in the user being enclosed in a bubble without being confronted with the opposite view. User is not exposed to anything that does not fall into user's field interest. It makes it easy for the content producers to influence their subscribers (hence the word influencers).

Socneto solves this problem by inverting the approach to the content searching. As stated above, users usually follow influencers or groups that publish posts related to some topic. The opposite approach, which is implemented in Socneto, is to search for a topic and find the content creators and other related topics.

2.2.2 Abundance of Data

The other problem social networks posses is the large amount of data that is not possible to read through them all and make one's opinion. Social networks do not typically offer a tool helping the user to improve orientation in the vast amount of data with comprehensive statistics of related topics, sentiment and such.

All data downloaded are accessible to the users to visualize their properties with various charts. The user can see a number of posts related to the given topic on the timeline, sentiment development in the time, top related topics.

2.2.3 API Limits

Although social networks have their data publicly accessible, it is not an easy task to access it. Access to the data via API is limited in two ways. One way is a limit imposed on a number of posts that can be downloaded per a given time period (typically 15 minutes or an hour). The other limit is imposed on the latest post which can be downloaded. For example the latest accessible post is only a week old which makes it impossible to analyse post of the past. (The limits are discussed in Chapter Implementation, Section 3.6)

Socneto overcomes the API limits by downloading the data continuously over a long period of time utilizing the limits to its maximum. It does not help users who want to analyse the history of the topic, but it helps people who know the topic in advance. When such user submits a job and the data starts flowing, the results can be seen immediately - Socneto is designed to calculate the analyses on the fly so the user always has the data up-to-date.

2.2.4 Limited Customization

There are various tools that focus on either downloading or analysing (or both) data from social networks (for deeper analysis refer to Section 2.3). These tools are either full fledged services that are not customizable at all or, on the other hand, the tool is an open source script that is customizable but typically is focused only on analysis or acquisition therefore it lacks functionality.

Socneto finds the middle ground by allowing users to add support for social network data acquisition and analysis out of the box as long as the components follow basic component

life cycle and interfaces(for more details refer to Chapter 5). When the users submit a job they can select multiple acquirers and analysers and also customize result visualizations.

2.3 Related Works

Socneto combines three major fields of interest: social networks, data processing and data analysis. Although there are plenty of tools and services that cover at least one of the fields, this section mentions only those that cover two or all three.

2.3.1 Field of Interest Classification

Social Networks

Social networks[3] are used by many types of users with various needs and requirements. The majority of the users are people that mostly consume the content and occasionally share something with friends or followers. The types of users that focus more on producing are companies or public bodies. While typical consuming users expect to have easy access to the content creators, the needs of the producers are more elaborate and can be summarized as follows:

- Intelligence - watching out for trends, consumer needs and profile appeal
- Influence Statistics - monitoring of posts influence
- Social Customer Care - swift reaction to posts or comments
- Social Media Management - publishing management and approval process and

Web monitoring and analysis

Data processing framework is expected to process and save large amounts from various sources of data of various types and shapes. The most notable properties of data processing framework are:

- Multisource - Monitoring of data from various sources.
- Multicontent - Ability to process textual, visual, audio and audiovisual material.
- Reporting - Comprehensive data presentation

In this context, data analysis is understood as a function of a tool to extract useful information from incoming data. Since Socneto analyses only textual data, the types of analyses that work with non-textual data are omitted in the following text. The tools falling into this category range from trivial functions such as word counters to very complex methods employing machine learning, such as sentiment analysis.

[3] YouTube and Reddit are also considered social networks in this text

Competitors

This section introduces four competitors falling into two categories according to their primary focus. The list of competitors is not exhaustive. The competitors merely are representing their category. The categories along with their representatives are the following:

- Social Network - Zoom Sphere[4], Social Bakers[5]
- Web Monitoring and Data Analysis - Monitora[6], Google Flu Trends[7]

ZoomSphere A tool focusing primarily on presentation and brand management is Zoom Sphere whose goal is to help companies or influencers to thrive on the supported social networks. Both products offer an extensive analysis of a current brand status focused on measuring the impact of created posts.

Social Bakers One of the best services in the market is Social Bakers offering a solution to the majority of problems stemming from managing social networks. According to its web page, it fulfills all the requirements discussed in social networks classification. Customers' needs are monitored on various levels ranging from conversational level – hot topic classification – to high-level brand sentiment.

Monitora Full-fledged media monitoring tools represented by Monitora that not only scrapes but also analyses and aggregates the scraped data. While the former tool is used by tech-savvy users who want only some data, the latter is a paid service that is claimed to employ a team of specialists to collect and interpret the results. This service is used by customers wishing to be informed public opinion.

Google Flu Trends Another service keeping track of web content is recently shutdown service Google Flu Trends which monitored Google Search activity related to searches of influenza. Such searches were used to predict the start of a "flu season" in more than 25 countries.

Comparison

In this section, Socneto is compared to the above-mentioned products. The main criteria are the following:

- Integration with social networks - Ability to connect to a social network and utilize its features.
- Searching capabilities - Ability to search continuously for the topic of interest across the web.

[4] `www.zoomsphere.com`
[5] `www.socialbakers.com`
[6] `Monitora`
[7] `https://www.google.org/flutrends/about/`

- Analysis and reporting - Ability to extract and present information from a vast amount of data.
- Extensibility - Ability to add user specific functionality

Integration with Social Networks The best integration with social networks is offered by Social Bakers which offers the best suite to utilize social network features. It connects content from all large social networks (such as Facebook, Twitter, LinkedIn[8]) and services that users use to consume and comment on the content such as YouTube[9]. Similarly to Social Bakers, ZoomSphere offers similar features. Both applications offer many charts that the user can easily set up and personalize and both require the user to explicitly connect the application with a profile.

Monitora does not have such requirements and offers similar reporting features related to social networks. Monitora is not restricted to social networks, it can scrape any important source of information on the internet. Lastly, Google Flu Service does not work with social networks at all.

Searching Capabilities The broadest searching capabilities are offered by Monitora whose data sources include Radio Broadcast, TV, press and online content (social networks included)[10]. On the top of that, Monitora archives the content and up to now offers content from the past 20 years. Therefore Monitora excels in the variety of data it covers and in its continuous acquisition. On the other hand, Google Flu trends utilize only Google Search input but it has access to the history of all Google searches.

The data sources of Zoom Sphere and Social Bakers are limited only to social networks. They archive all the data related to the given user profiles and also continuously monitor all related data.

Analysis and reporting Zoom Sphere[11] and Social Bakers[12] analytic tools focus primarily on metrics important to a social network user such as the impact of a post or the trend analysis so the user can create posts with the best possible influence. Both Zoom Sphere and Social Bakers offer a variety of reporting tools to visualize all the data. Monitora[13] offers similar types of analyses and reports without limiting itself to social networks only. Google Flu Trends is an analytic tool itself.

Extensibility Zoom Sphere does not explicitly offer any means of extensibility they are covering most of the required functionality of the social network domain out of the box. Social Bakers have a similar approach, yet the platform can be extended but the customer would have to participate in the partnership program[14]. Monitora is not an extensible product, but its creators offer custom services[15].

[8] www.linkedin.com
[9] www.youtube.com
[10] https://monitora.cz/monitoring-medii/
[11] https://www.zoomsphere.com/analytics
[12] https://www.socialbakers.com/solution/measurement-and-reporting
[13] https://monitora.cz/analyza-medii/
[14] https://www.socialbakers.com/integrations-and-partnerships
[15] https://monitora.cz/kontakt/#sluzby

Google Flu Trends offered their API for users to potentially implement custom analysis on the top of the google search data. Although the service was closed in August 2015 [16] and the API is no longer available, the datasets are available for academic purposes.

Comparison with Socneto In terms of social network capabilities, Socneto is not a competitor to Social Bakers or Zoom Sphere. Especially with the user-profile oriented capabilities. These two products focus on a user profile (or multiple profiles at once) and analyse only data related to its activity. Socneto, on the other hand, works the other way round. It searches for any information relevant to a given topic regardless the user profile which is the similar approach Monitora implements.

The searching capabilities of Socneto that are supported in the current version (further extensible) is limited to Reddit and Twitter only. The strong point is that these social networks are monitored continuously and data are also stored in an archive.

The supported analyses of the current version of Socneto correspond to the way Socneto obtains data. The analyses are applied to the data for the given topic producing public opinion through sentiment analysis or related topics through topic modeling (see Chapter 4). Given the fact that Socneto does not focus on a particular user or a profile, the post-specific analyses could not be implemented. But with the proper prediction model, Socneto can be easily extended to simulate Google Flu Trends, but applied to social networks.

The products that were discussed in this section lack the ability to easily extend their functionality in which Socneto stands out. Socneto has the potential to compete with the other through extensibility (see Chapter 5) in the following way:

- Zoom Sphere and Social Bakers - Both products excel in the integration with social networks and in the focus on a profile rather than a broad overview. Although Socneto already implements social network data acquisition, it can implement another one which does not search through data related to the given topic but focus on all data related to a single user.
- Monitora - The strongest side of Monitora is the broad scope of sources it can track. Socneto's collection of data sources does not have to be limited to social networks or textual data.
- Google Flu Trends - The flu prediction can be incorporated into Socneto Analysis and then applied on the data from various other sources.

In conclusion, Socneto has a great potential to compete on the market. In its current form, Socneto can be understood as a flexible framework that can be modified to serve multiple purposes.

2.4 Real-World Use Cases

Socneto helped to get a grasp of social network data in the following two cases:

- Doctors without Borders[17] - analysis of measles and snake bites.

[16] `https://www.google.org/flutrends/about/`
[17] `https://www.lekari-bez-hranic.cz/`

Language	value
en	46998
fr	1855
ar	1211
und	1010
tl	599
ja	482
fa	124

Figure 2.1: The graph show distribution of languages for a query `measles` in French, Arabic and English

- Datamole[18] - summarizing most discussed topics Consumer Electronic Show (CES)[19] on Twitter

2.4.1 Measles and Snake Bite Analysis

Doctors without Borders, represented by Jan Böhm whose responsibility, among others, is studying and monitoring social networks. Socneto was used for analysis of topics related to two given ones: measles and snakebites. The analysis consisted of running two separate jobs for each topic and gathering data for a period of 40 days.

As stated by Jan Böhm, the results showed several "interesting remarks that can be used in their work" and also revealed limits of the easily accessible Twitter tweets and their misleading nature. Although the query was given in three languages: English, French and Arabic, the results were mostly in English (as can be seen in Figure 2.1). According to Jan Böhm, "this supports a theory that Twitter is mainly used as a tool for reporting the news to the world and not for communication among locals whereas Facebook is claimed to be the platform for such purpose".

The next occasion where Socneto proved itself was an analysis of trending topics of a convention Consumer Electronic Show (CES)[20] that took place at the beginning of January 2020. Socneto helped to get a grasp of 120 thousand tweets during the three-day convention. The related topic analysis revealed that the most discussed topics were related to the television and car industry.

[18] `https://www.datamole.cz/`
[19] `www.ces.tech`
[20] `https://www.ces.tech/`

2.5 Project Timeline

The diagram of the project timeline is shown in Figure 2.2. The following text follows its structure.

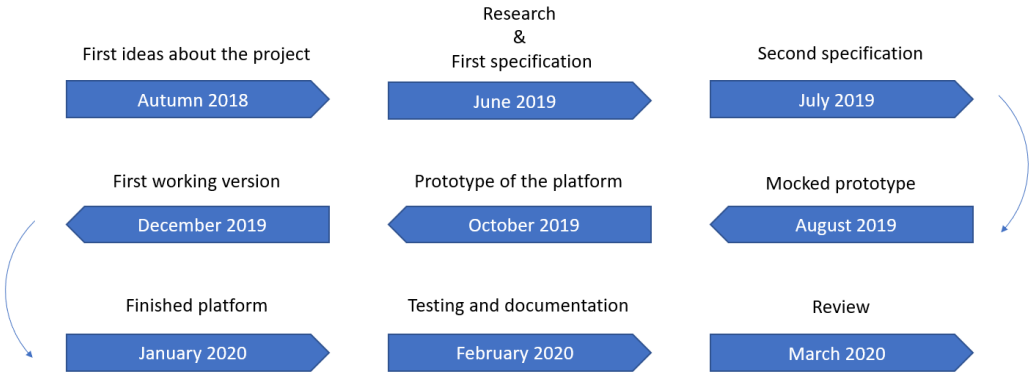


Figure 2.2: Socneto development timeline showing the progress since Autumn 2018

2.5.1 Autumn 2018 - The First Idea

The idea of creating a framework for data processing can be traced to Autumn 2018 and was motivated by the lack of practise with a big data framework. This idea was then consulted with the supervisor Doc. RNDr. Irena Holubová, Ph.D. who offered various feedback and helped us to find companies interested in cooperation. No cooperation was considered feasible though. Soon after, Jan Pavlovský (a former member of the team) came up with an idea to focus on social network analysis. The idea seems to be interested enough for both parties and the search for the member started. Each member joining the team came up with ideas and the project started to take more solid shape. After many discussions, we came up with the idea of Socneto in June 2019 and the official project could start.

2.5.2 June 2019 - Official Start

The software project work opened with writing the short specification with all the important ideas and features of the upcoming project. After it was accepted, the work on the second, full, specification could start. This document should contain not only a technical overview but also insight into project direction and management. We had an idea of separating the development into three stages:

- proof-of-concept - a project exploring the playing field,
- full application - first application with all working components

- testing and documentation - last stage of the development

2.5.3 July 2019 - Proof-of-Concept and Full Specification

The proof of concept was finished along with the full specification in July 2019. At the time we were quite confident that we will not encounter any blind alley that would prevent us from finishing the project. Although most of the smart components were mocked, we found out that an application featuring an asynchronous messaging system would work. All mocked components were able to communicate well and others can be plugged in easily.

2.5.4 August 2019 to December 2019 - Working Prototype

Most of our effort was put into replacing mocked components with working ones. Jaroslav Knotek finalized components downloading data from Twitter and Reddit. On top of that, we also added a component for the custom dataset which helped us with testing and at the end, it is a valuable addition to the offered "data acquirers" (for more information, see guide walking a user through the usage of custom dataset in Chapter 5).

Petra Vysušilová worked on sentiment analysis and topic modeling, replacing simple models with more elaborate ones. For better results, it was essential to get a more powerful infrastructure used for the training. For this purpose, the university GPU cluster was used to help train the model in a reasonable time.

The entity model and all storage capabilities of Socneto were developed by Lukáš Kolek who built up storage for posts, their analysis, and all metadata. His part of Socneto, the whole database layer, is used by most of the components and is crucial for its proper function. Lukáš also helped to design interfaces between components so the communication would be as smooth as possible.

The most visible part of the application was designed and implemented by Jůlius Flimmel who made it possible to visualize analyses and thus made them easily accessible. He was also responsible for designing the backend part of Socneto to ensure correct integration with the frontend.

This period was the time when Jan Pavlovský left the team after mutual agreement. Jan helped us the most at the beginning of the project when he proved to be a constant source of ideas. In the later phases, he has contributed to the architecture and a visual appearance. Unfortunately, his interest in the project was gradually diminishing after a discussion it was clear that did not intend to keep pace with the team. The project continued on with few changes. The support for the Czech language was omitted and all requirements concerning machine learning activities were lowered since Petra was the only remaining member capable of performing the task diligently.

2.5.5 January 2020

Because of a careful planning at the beginning, the implementation phase went smoothly without major hiccups. At the end of December, we had a working platform with all the necessary components. It was time to find real-world users. With the help of Jiři Suchomel from the PR Department of the Faculty of Mathematics and Physics, we met Jan Böhm for

whom we successfully analysed Twitter posts concerning measles and snake bites (for more details, see Section 2.4).

2.5.6 February 2020

At the beginning of 2020 with the platform finished, we could start proper testing and documenting. At that time, we wrote scripts that start only a selected component in a controlled environment and assert its behaviour (see Chapter 6). The documentation was written mostly during February. The 1st of March 2020 is the final deadline and according to the Software Project Committee customs we would be finished with defense by the end of the march.

Chapter 3

Development Documentation

3.1 Architecture

Socneto is a distributed application, the whole platform is robust and components are loosely coupled. Application is divided into services by their functionality. Everything is connected via Kafka messaging 3.1.3 or internal REST API.

3.1.1 Components

Socneto is divided into eight logical parts:

- Messaging (see Subsection 3.1.3)
- Frontend (see Section 3.3)
- Backend (see Section 3.4)
- Job managing service (see Section 3.5)
- Multiple data acquiring components (see Section 3.6)
- Multiple data analysers (see Section 3.7)
- Storage (see Section 3.8)
- Monitoring (see Section 3.9)

Socneto can be extended by adding custom data acquirers or data analysers (for more details, see Section 3.10).

3.1.2 Component Relationships

The communication among the components respects the data flow of the application. A user submits a job through frontend to backend which notifies job management service (JMS). JMS then notifies all selected data acquirers and data analysers. The data analysers start to produce posts that start to flow into data analysers and to the storage. The analysers consume posts and produce analyses that are also sent to the storage. The storage consumes the posts and analyses and offers them to the user's queries through backend query endpoints. These relationships can be seen in Figure 3.1.

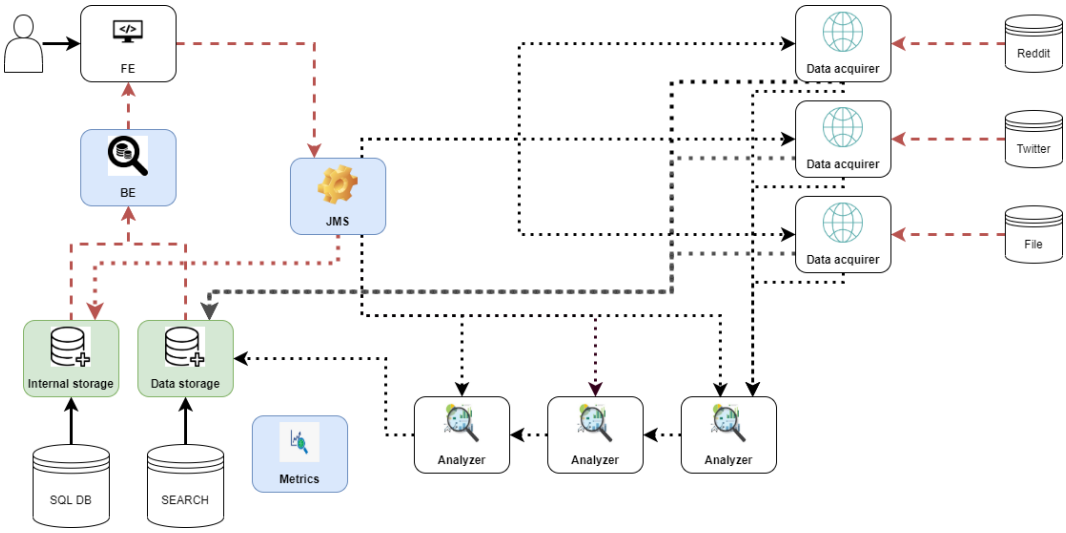


Figure 3.1: A diagram of the Socneto architecture showing relation ships among all components. This communication is asynchronous

3.1.3 Messaging

Publish & Subscribe Principle *"Publish/subscribe messaging, or pub/sub messaging, is a form of asynchronous service-to-service communication used in serverless and microservices architectures. In a pub/sub model, any message published to a topic is immediately received by all of the subscribers to the topic. Pub/sub messaging can be used to enable event-driven architectures, or to decouple applications in order to increase performance, reliability and scalability."*[1]

Publish/Subscribe in our Use-Case Posts are acquired and asynchronously sent to analysers and storage. Analysers subscribe to their assigned queues and after each analysis, they also send results to storage via messaging. Using this approach we achieved better

[1] `https://aws.amazon.com/pub-sub-messaging/`

extensibility of Socneto, high-throughput and loose coupling between modules. Defined message structures are described in Appendix A.1. Acquirers and analysers get assigned topics during a registration and update processes.

Implementation In Socneto we have chosen Apache Kafka 2.4.0[2] for messaging, which is an open-source project from Hadoop family[3] developed by Linkedin and the platform meets our requirements. Also, all common languages have a library with a connector to Kafka.

3.2 Project Structure

The project has the following directories

- `acquisition` - Data acquirer code
- `analysers` - Data analyser code
- `backend` - Backend code
- `docs` - Documentation
- `frontend` - Frontend code
- `job-management` - Job Management Service code
- `storage` - Storage related code
- `tests` - Tests

3.3 Frontend

Frontend enables the user to use the platform via a modern graphical interface. It is a web-based application, so it can be run in any environment with a web browser, however it is aimed for the environments with larger screens (i.e. computers, not phones). Using frontend, the user can:

- **create jobs** - start acquiring and analyzing posts,
- **see the results** of jobs, including:
  - *analyses*,
  - optionally filtered acquired *posts*

3.3.1 Dependencies

Frontend requires *backend* component to be running in order to function properly. This is caused by all the data it displays is being acquired via backend.

[2] `https://kafka.apache.org`
[3] `https://hadoop.apache.org`

3.3.2 Implementation

The frontend is primarily written in `AngularDart` library [4] with minority of the code being written in pure `JavaScript` and `CSS`. The `AngularDart` heavily uses `angular_components` library [5]. It provides multiple reusable components which are based on Material design [6].

The whole frontend codebase can be found in `/frontend` directory of the project repository.

Code - AngularDart

As a typical Angular application, it uses *component based architecture*. Each component consists of three parts:

- `Dart` code - implements the business logic of the component (it is later compiled to `JavaScript`, which is described in Section 3.3.2),
- `HTML` code - a special version of HTML which supports interpolations of `Dart` code,
- `CSS` code - used for styling the resulting `HTML` code.

Each component is translated into an HTML tag.

The components are complemented with *services*, which provide logic that is not bound to any component and *models* which represent data and their structure.

Services

The logic which is used across multiple components, such as communication with backend, is implemented in services. These can be found in directory `/lib/src/services/`. Frontend contains the following services:

- `HttpServiceBase` is an abstract wrapper of Dart's `http package`, which is implementation of HTTP protocol. This service adds an option to specify IP address and prefix of all HTTP requests. It also adds two functionalities to the http package:
  - checking whether the HTTP response was successful (i.e. returned status code indicating success), or else throws `HttpException`,
  - deserializes and returns the JSON body of the response in form of Dart's Map object,
- `HttpServiceBasicAuthBase` extends `HttpServiceBase` by adding `Authorization` header to all HTTP requests using the *basic authorization token*. This token can be specified directly, or by providing the service with username and password,
- `LocalStorageService` is a simple utility service which handles storing and loading of user credentials to the browser's local storage,

[4] `https://angulardart.dev/`
[5] `https://dart-lang.github.io/angular_components`
[6] `https://material.io/design/`

- `SocnetoDataService` extends `HttpServiceBasicAuthBase` with concrete HTTP requests to Socneto backend. All communication with backend is handled by this service,
- `SocnetoMockDataService` uses the same interface as `SocnetoDataService`, but instead of sending real requests to the backend, it returns mock data. This service should be used only during development and not on production!
- `SocnetoService` use `SocnetoDataService` via composition. It also uses `LocalStorageService` to store and load user's credentials in the browser's local storage,
- `PlatformStatusService` is used to get the status of core Socneto components and inform subscribed Angular components about its changes. If any Angular component is subscribed for the changes, the service polls backend using `SocnetoService` every 5 seconds and asks about the platform status. If a different status is returned then in the previous request, then all the subscribed components are informed about this change.

Components

Most of the frontend codebase consists of implementations of multiple different components. These components can be found in directory `lib/src/components`. They are typically implemented in three parts (Dart, HTML and CASS). The convention in this project is, that each part is implemented in a different file. All of these three files are then placed in the same directory, have the same name and differ only in their extension.

In this project, instead of using CSS, we use SCSS. Thanks to its more extensive syntax, SCSS is more user-friendly. However, it needs to be compiled later into CSS, because browsers do not support it. This process is described in Section 3.3.2.

Some components are designed for reuse in multiple other components. Their code is located in directory `lib/src/components/shared`. These components are:

- `ComponentSelect` - gets a list of Socneto components on input and displays to the user the list of these components. The user can select any of these components, and after each selection change, this component triggers an event about it,
- `Paginator` is a simple visualisation of pagination. It displays a list of available pages and triggers an event on each page change,
- `PlatformStartupInfo` displays a warning for each core Socneto component which is not running. It subscribes itself to the `PlatformStatusService` to get notified on each component status changed. This component is used only during startup! After all components are running, the component unsubscribes itself from the `PlatformStatusService`, and will no longer show warnings if a component stops working,
- `SocnetoComponentStatus` - gets a `ComponentStatus` as an input and displays it in a form of a colored icon,
- `Verification` is a modal dialogue, which is given a verification message on input, displays the message and triggers an event when the user presses the `yes` or `no` button.

The root component of the application, `App`, is the only direct child of HTML's <body> tag:

- `App` is the root component of the application. Its only functionality is to check whether there are valid credentials in the browser's local storage and login the user automatically if there are,
- `AppLayout` is the only child of `App` component. It creates the main layout of the page:
  - *header bar* at the top of the page,
  - *hideable drawer* on the left,
  - *content* below the header. It can be either `Login` or `Workspace` component. Which of them is displayed is based on current page's URL[7].

Description of other important components follows:

- `Login` displays inputs for login and password. It also disables the inputs during the platform startup, while not all of the core Socneto components are not running,
- `Workspace` displays user's workspace when logged in. It is composed of two other components: `JobsList` on the left side and the second component on the right. The second component is chosen by current page's URL[8]. It can be either `QuickGuide` or `JobDetail` component,
- `JobsList` displays paginated list of all user created jobs along with basic info about them. It is implemented using material list component[9]. The data for this component is retrieved from backend via `SocnetoService`,
- `CreateJobModal` contains two-step material list component[10]. The first step contains inputs for each required field in a job definition and the second optional step contains credentials for the job. Only if a correct job definition is defined by the user, the job can be submitted to backend,
- `QuickGuide` displays a simple guide for the user. The guide is implemented using material stepper component [11],
- `JobDetail` displays detail of the currently selected job in the `JobsList` component. It is a material list component with three tabs. The first tab contains `JobStats` component, the second contains `ChartsBoard` component and the third contains `PostsList` component,
- `JobStats` gets a `Job` on the input and displays some basic info about it and a few frequency charts (posts, language and author frequency),

[7] `https://angulardart.dev/tutorial/toh-pt5`
[8] `https://angulardart.dev/tutorial/toh-pt5`
[9] `https://dart-lang.github.io/angular_components/#/material_list`
[10] `https://dart-lang.github.io/angular_components/#/material_list`
[11] `https://dart-lang.github.io/angular_components/#/material_stepper`

- `ChartsBoard` gets a `jobId` on the input and displays one `Chart` component for each chart of that job. At the bottom of the list, it displays `CreateChartButton` component, which opens `CreateChartModal` when clicked on,
- `CreateChartModal` display inputs required for definition of a chart. Only a correct chart definition can be then submitted via backend,
- `Chart` get a `ChartDefinition` on input and queries chart data from backend according to it. It then displays the data in a chart. The chart's type is also selected from the chart definition,
- `PostsList` get `jobId` on input and displays *paginated* list of posts acquired by the given job. The list is retrieved from backend. It also displays optional filters, which can be applied to the list. After each change in the filter, it queries backend for the posts again. It also contains a button to export all the (filtered) posts to CSV. The button's redirect URL is retrieved from backend.

Routes

Some of the components use angular routing[12] to select which component to display. All of the available routes are defined in `/lib/src/routes.dart`, along with their route parameters and mappings of concrete routes to concrete components.

Styles

Instead of using plain CSS, we use SCSS for its better and more extensive syntax.

The frontend uses two color palettes in each component - *primary* and *background*. In order to easily change these in the whole application, theming is implemented. Each component has its own styles wrapped in a SCSS mixin[13], which has a *theme* parameter. This parameter contains both palettes, and every color used in the *mixin* is picked from them. All these component mixins are then included in the `apply-theme` mixin which passes the *theme* parameter to each of them. We then implement multiple CSS classes which include this `apply-theme` mixin with different predefined theme objects, which can be found in `lib/src/style/theming/themes.scss`.

Different color palettes are defined in `/lib/src/style/theming/palettes.scss`, which were created according to material color palette rules[14] using Palette Tool[15]. The SCSS functions for retrieving specific colors from color palettes are implemented in `/lib/src/style/theming/theming.scss`.

Models

Directory `/lib/src/models/` contains all of the models used throughout the application. It contains models for objects sent and received by backend as well as models used exclusively

[12] `https://angulardart.dev/tutorial/toh-pt5`
[13] `https://sass-lang.com/documentation/at-rules/mixin`
[14] `https://material.io/design/color/the-color-system.html#color-usage-palettes`
[15] `http://mcg.mbitson.com/#!?mcgpalette0=%233f51b5`

only on frontend.

Models representing requests contain function `toMap()` which serializes them to Dart's generic `Map` object. This can be then easily serialized to JSON string by `HttpServiceBase` and sent to backend.

Models representing responses from backend contain function `fromMap(Map data)`, which deserializes the `data` parameter into the model.

Interoperability

Dart SDK provides means to communicate with native JavaScript code via `js` package[16]. This interoperability is required, when we want to use a `JavaScript` library, which was not rewritten to Dart.

We use this functionality for two `JavaScript` libraries. One of them is `toastr`[17], which is used to display notifications. The second one is Socneto's own JavaScript library for displaying charts. To be able to use these libraries, we implemented interfaces between Dart and them, using the mentioned `js` package. These interfaces can be found in directory `/src/lib/interop/`.

Even with these interfaces, we still need to import these `JavaScript` libraries from the resulting HTML code. This is done in `/web/index.html`. The `toastr` library requires `jQuery` library to function, so that one is also imported.

Code - JavaScript

To draw different types of charts, we use `d3js library`[18]. This library was not rewritten to Dart. It also has an extensive API, so it would take considerable effort to create a Dart interface for it using Dart's `js` library as mentioned in Section 3.3.2. For this reason, we write chart components in pure JavaScript. From this code we then make a `JavaScript` library.

`Socneto` object implemented in `/lib/src/charts/charts.js` exposes multiple static functions for creating charts with data as their parameters. Implementations of the charts are then contained in separate JavaScript classes and CSS files located in the same directory.

Static

All the static image, which are used by the frontend are contained in `/lib/static` directory.

Configuration

`SocnetoDataService`'s backend address is read from environment variable `backendAddress`. It is read from the environment by utility class `Config`. If the variable is not specified, it defaults to `localhost:6010`.

The port where the frontend will be running is specified during build process (see Section 3.3.2).

[16] `https://api.dart.dev/stable/2.7.1/dart-js/dart-js-library.html`
[17] `https://codeseven.github.io/toastr/`
[18] `https://d3js.org/`

Build and Run

The frontend can be built using two different methods. It can be built either by using Docker or without Docker.

Build and Run - Without Docker

To build the frontend without Docker, the following packages are required:

- **Dart SDK** version 2.5.2 or higher,
- **NodeJs** version 8 or higher.

The following steps are required to build frontend:

- **Gulp** - for better performance, we do not want to import from HTML all of our JavaScript and CSS files individually. To merge all of them into one JavaScript and one CSS file, we use *gulp* tool. This tool can be installed only via `npm`, using command `npm install gulp && npm install gulp -g && npm install gulp-concat`. The gulp is configured via `/gulpfile.js` to do the merging as a default task. This means, we can run `gulp` command with no parameters or arguments and it will create the resulting two files, and put them to `/lib/static` directory.
- **Pub** - next, we need to download all the dependencies for our Dart application, which are specified in `/pubspec.yaml`. This is handled by running command `pub get`,
- **Webdev** - the frontend server is started using `webdev` tool. This tool was already installed in previous step by `pub`. To be able to use the tool from command line, we need to activate it using command `pub global activate webdev 2.5.1`. Now we can start the server using command `webdev serve`

Note: Since web browsers support only `CSS`, all of our `SCSS` files are compiled to `CSS` during the build. The compilation is done by `angular_components` library, and it is specified in build configuration file located at path `/build.yaml`.

The resulting commands sequence is then following (run from `/frontend` directory of the Socneto repository):

```
npm install gulp
npm install gulp -g
npm install gulp-concat
gulp
pub get
pub global activate webdev 2.5.1
webdev serve
```

Using `webdev serve` without any parameters, we start the server on address `localhost:8080` and compile Dart code to JavaScript using `dartdevc` compiler, which is more suitable for development then production. If we want to change the port, on which the server runs, we need to start it using command `webdev serve web:<PORT_NUMBER>` (replace `<PORT_NUMBER>`,

with the required port number). If we are building the Dart web application on production, it is more suitable to use its `dart2js` Dart to JavaScript compiler. This compiler next to other features also minifies the resulting JavaScript, increasing the performance of our application. This can be achieved by running the above commands with option `--release`.

Build and Run - Using Docker

To build the frontend using Docker, we need to have installed following pakcages:

- **Docker** version 18.0.9.7 or higher.

The Socneto repository contains prepared functional `Dockerfile` located in directory `/frontend`. This `Dockerfile` builds the frontend using `dart2js` compiler (see Section 3.3.2), and starts the server on port `8080`, accessible from `localhost` on host machine. To build and run the image, execute the following commands:

```
docker build -t "frontend" .
docker run -p "8080:8080" "frontend"
```

After these commands the frontend can be accessed via a web browser on address `localhost:8080`.

3.3.3 Communication

Frontend communicates with Socneto via backend's HTTP JSON REST API.

3.4 Backend

Backend serves as a middle component between *frontend* and the rest of the platform. It provides means for the users to retrieve different data from the platform and create new jobs. It also authenticates users using it and check whether the users are authorized to get the requested data.

3.4.1 Requirements and Dependencies

Backend requires two other components to be running in order to function properly:

- **Job Management Service** - to create and stop jobs,
- **Storage** - to retrieve multiple different data.

It also needs Kafka for event tracking.

3.4.2 Implementation

Backend is a web application running on ASP .NET Core 2.2 which implements standard Model-View-Controller principle. The solution follows data-driven-development which requires the code base to be split into the following projects:

- `Api` - entry point of the application, contains configuration `appsettings.json` and controllers handling HTTP requests,
- `Domain` - business logic, platform independent,
- `Infrastructure` - platform dependent implementation.

The application uses dependency injection that is configured in the project `Api` in the file `Startup.cs`.

Code

The API's controllers are split according to which entities they operate with. Their role is to define concrete HTTP routes the API exposes, which HTTP methods they allow, and how the bodies of the requests look like. They also verify, whether the user querying the API is authenticated and authorized to do such requests. The authentication is implemented via HTTP's `Authorization` header, using *Basic token*.

The following controllers are implemented:

- `SocnetoController` is a base class for each controller. After handling each API request, it checks whether `ServiceUnavailableException` was thrown, and if it was, its sets the response's HTTP status code to 503 (Service unavailable),
- `ChartsController` defines API for management of charts. It defines requests for getting list of all charts defined for a job, creating a chart for a job and removing a chart from a job,
- `ComponentsController` defines API for listing components, which are connected to Socneto platform. It defines one route for getting all currently connected analysers and one for currently connected data acquirers,
- `JobController` defines API for retrieving job-specific data. It defines routes for listing all user's jobs, creating and stopping a user's job and getting status of a user's job. It also defines a route for getting posts acquired within a given job. The request is paginated and returns posts only from a given page and also support multiple filters. To get all the posts, it exposes another route, which returns all the posts in CSV formatted file. The controller also defines routes for retrieving aggregation and array type analyses for a given job, as well as basic analyses for each job, which are *language frequency*, *author frequency* and *posts frequency*,
- `ManagementController` defines API which informs whether the backend is running, and which other Socneto core components are running,
- `UserController` defines single API route, which is for logging in. It verifies, whether the user which is trying to login exists, and provided credentials are correct.

The `Domain` project implements business logic of the backend. It is split into multiple services. While API project works only with interfaces of these services, their implementation are in this project. Just like the API project, Domain also defines multiple models,

which define objects, that are transfered between backend and other Socneto components, or between API and Domain projects.

The following services are implemented:

- `AuthorizationService` implements the logic of verifying, whether a given user is able to interact with a given job in any way,
- `HttpService` is a wrapper over C#'s `HttpClient`[19]. It implements utility functions for `Get`, `Post` and `Put` HTTP requests, which automatically check, whether the HTTP response was successful (HTTP status code which indicates success was returned), and deserializes the body of the response to a type specified by method's generic type. It also throws `ServiceUnavailableException` if the requested route is unavailable,
- `StorageService` implements Socneto's storage component REST API interface. All communication initiated by backend with storage component is handled by this service,
- `JobManagementService` implements Socneto's job management service's component REST API interface. All communication initiated by backend with JMS is handled by this service. It also sets default values for twitter and reddit credentials when submitting a job,
- `ChartsService` implements the logic behind creating, listing and removing charts for a given job. It queries storage component to retrieve and store the required data,
- `CsvService` implements creation of a CSV formatted file from a list of objects of the same serializable type. Serializable type in this context is a type, which uses `Newtonsoft.Json` library attributes, to mark, how it should be serialized into JSON. This service, however, uses these attributes to serialize the objects into CSV,
- `GetAnalysisService` implements creation and execution of correct queries to storage, to retrieve analysis of acquired posts with given parameters. A query creation is based on:
  - *Analyser's output format*: the query must ask for the field of the analysis, which is being populated,
  - *Type of the analysis*: the analysis can be an aggregation of analyses' field or an array of values of analyses' field,
  - *Requested result type*: the requested result may contain multiple analyses (e.g. line chart can contain multiple lines), in which case multiple queries to storage must be executed, and their results must be correctly merged together,
- `JobService` implements retrieving jobs statuses and jobs' acquired posts from storage,
- `UserService` implements authenticating a user from given credentials. It queries storage whether the given user exists.

[19] `https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=netcore-2.2`

Event tracking is implemented by class `EventTracker`, which has public method for tracking messages on different levels (error, warn, info, ...). Any tracked message is enqueued to `EventQueue`. `EventSendingHostedService` then tries to dequeue a message from the `EventQueue` every 5 seconds and sends it via `IMessageBrokerProducer` to the platform.

The `Infrastructure` project implements functionality required by the chosen infrastructure of the platform. It contains implementation of `IMessageBrokerProducer` interface using Kafka.

Configuration

The configuration of backend is stored in project `API` in file `appsettings.json`. The configuration is distributed via a dependency injection container. The most notable configuration objects are:

- `KafkaOptions` - contains location of Kafka,
- `StorageOptions` - contains location of storage component's API,
- `JobManagementServiceOptions` - contains location of job management service component's API,
- `DefaultAcquirersCredentials` - contains default credentials for Twitter and Reddit acquirers, which are used if no credentials are specified via frontend when creating a job.

Build and Run

Backend can be built and run with two methods: with or without Docker.

If using docker, the required Docker version is *18.0.9.7* or higher. The Dockerfile can be found in `/backend/` directory. From there, it can be built and run using the following commands:

```
docker build -t "backend" .
docker run -p "6010:6010" "backend"
```

Then the backend's API can be found on `localhost:6010`.

If not using Docker, the project can be built and run using *.NET Command Line Tools* (version 2.2.401 or higher). The following commands will build and start backend when executed from `/backend/Api`:

```
dotnet restore # download all third party libraries
dotnet build # build the projects
dotnet run # run the project
```

After executing these commands, backend's API can be found on address `localhost:5000`.

3.4.3 Communication

Backend communicates with *Storage* (Section 3.8) and *JMS* (Section 3.5) components. It also exposes its own HTTP interface.

HTTP interface

Backend exposes its own *HTTP JSON REST API* for *Frontend* to communicate with it. The API consists of multiple routes, which are documented in `/docs/api/be-api.pdf`.

Outgoing communication

To communicate with *JMS* and *Storage* backend uses their *HTTP APIa* (see Sections 3.5.5 and 3.8.4).

Most of the requests on backend simply redirect them to one of the other two components with minimal changes and some input validation.

3.5 Job management service

Job management service (JMS) responsibilities are:

- Keeping track of all registered components
- Accepting submitted jobs
- Distributing jobs among all components involved

3.5.1 Requirements and Dependencies

Job management service requires the following tools for it's proper function:

- Kafka 2.4.0
- .Net Core 3.1.100

It communicates with the following service:

- Storage service (see Section 3.8)

3.5.2 Code

JMS is web application running on ASP .NET Core 3.1 which implement standard Model-View-Controller principle. The solution follows domain-drive-principle[20] which requires the code base into the following projects:

- `Api` - entry point of the application, contains configuration `appsettings.json` and controllers handling http requests
- `Domain` - business logic, platform independent
- `Infrastructure` - platform dependent implementation

[20] `https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice`

- `Tests` - unit tests and integration tests

Application uses dependency injection that is configured in the project `Api` in the file `JobManagementServiceBuilder.cs`.

JMS main domain is to manage jobs invoked via HTTP interface. Each job is performed by component. Since the components are not known in advance, each component needs to registered itself via Kafka interface. JMS is stateless, all the data is stored in storage service. The major components of JMS can bee seen in the Figure 3.2.

Major Classes

- `JobController` - The controller responsible for processing job related requests
- `SubscribedComponentManager` - Contains the vital logic of JMS implemented by the method `StartJobAsync`. Also used for registering components
- `RegistrationRequestListener` - Continuously listens to the topic `job_management.registration.request`. It is used by all other components to register themselves (for more details refer to Section 3.5.5).
- `RegistrationRequestProcessor` - Parses registration requests and stores it in the database
- `StorageService` - Used as a proxy to storage service (see Section 3.8)

3.5.3 Configuration

The configuration of JMS is stored in the project `Api` in the file `appsettings.json`. It does not require any user's input. The configuration is distributed via a dependency injection container. Each option of the configuration needs to be bound to the file. This binding is in the project `Api` in the file `JobManagementServiceBuilder.cs`.

The most notable configuration objects:

- `KafkaOptions` - contains URI of Kafka. In case of starting JMS differently then advised, this element would need to be changed. (for `localhost` for example).
- `JobStorageOptions`, `StorageServiceHealtcheckOptions`, `ComponentStorageOptions` - these objects contain endpoint routes for the storage service.
- `StorageChannelNames` - names of Kafka topics used to propagate posts and analyses to the storage.

3.5.4 Build + Run

Navigate to the source code, specifically to the `job-management/Api` type in the following commands:

- `dotnet restore` - downloads all third party libraries
- `dotnet build` - build the projects
- `dotnet run` - run the projects

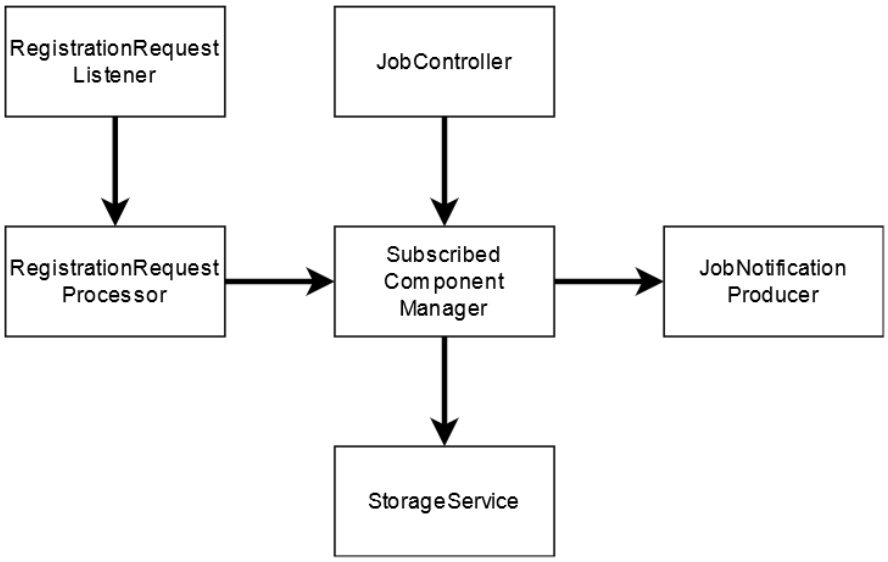


Figure 3.2: The diagram shows relationships among major classes of JMS

3.5.5 Communication

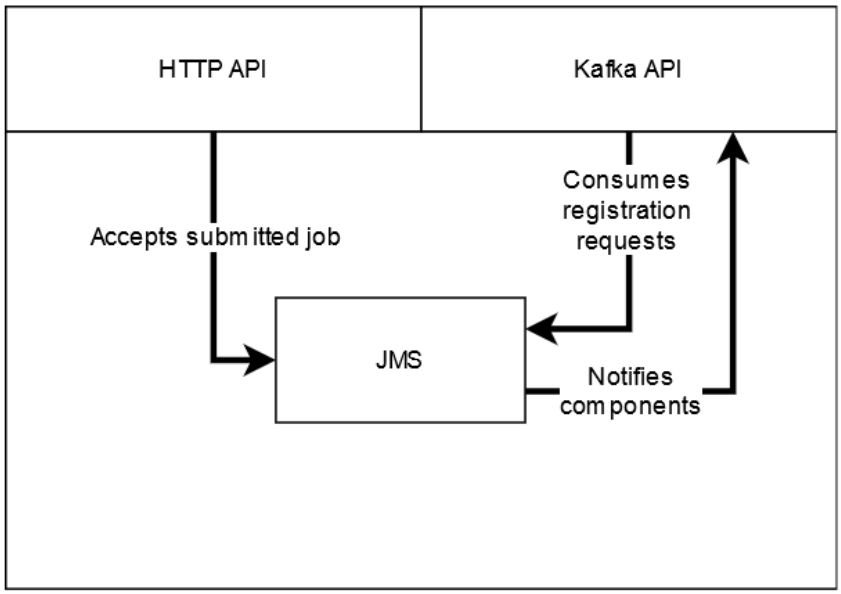


Figure 3.3: HTTP interface is used to start or stop jobs. Kafka interface is used to listen to registration requests

JMS uses Kafka and HTTP for communication depicted in Figure 3.3.

Kafka Interface

Kafka is used to listen for registration request of the JSON format on the topic `job_management.registration.` from all components.

The registration request (see Appendix A.1.1) of a component must contain the following fields:

- `componentType` - either `DATA_ACQUIER` or `DATA_ANALYSER` depending on the type of the component.
- `componentId` - unique id of the component
- `inputChannelName` - name of the channel unique to the component. The component will use it for ingesting the data (in case of acquirer, this field is `null`.
- `updateChannelName` - name of the channel that the component uses for accepting job notifications.
- `attributes` - an object contains any other attributes

HTTP Interface

JMS offers two endpoints accepting POST requests:

- `/api/job/submit` - starts jobs and notifies all components
- `/api/job/stop/<jobId>` - stops the jobs on all the components. Has no payload.

Submitting a job requires a body with the JSON payload with the following fields:

- `selectedDataAnalysers` - component ids of all selected analysers. If any analyser has not been registered yet, the request fails.
- `selectedDataAcquirers` - same but with acquirers.
- `topicQuery` - topic of interest.
- `jobName` - human readable name.
- `attributes` - custom attributes of selected components. If user wants the component to receive some attributes, the JSON object with the attributes must be wrapped with an element with the component id.

Outgoing Communication

JMS sends job notifications to the components via Kafka. Each registered component has a topic on which it listens to job configuration. When the job (see Section 3.5.5) is submitted, all components involved are sent a job notification (see Appendix A.1.3). The components that re-register themselves receive all active job configurations. This prevents allow crashing components to continue on unfinished jobs.

3.6 Data Acquirer

Data acquirer is a component downloading data from a social network (or any other component). Some social networks with free API impose limits on the amount of data downloaded (e.g., for Twitter limits refer to [21]). These limits are tackled by downloading data continuously not in one batch.

The posts may come in various languages but the analyses can usually work only with English. For this reason, analysers can optionally translate all acquired posts from any language to English.

3.6.1 Requirements and Dependencies

Data acquirers require the following tools for its proper function:

- Kafka 2.4.0
- .Net Core 3.1.100

[21] https://developer.twitter.com/en/docs/basics/rate-limits

- Minio 6.0.8

They communicate with the following services:

- Storage service (see Section 3.8)
- Job management service (see Section 3.5)

And, the following libraries are referenced:

- LinqToTwitter 5.0.0
- Reddit 1.3.4
- MinioClient 3.1.8

3.6.2 Code

Data acquirers are web applications running on ASP .NET Core 3.1 which implement standard Model-View-Controller principle. The solution follows domain-driven-development which requires the code based into the following projects:

- `WebApi.*` - For each data acquirer there is separete entrypoint web project with its own configuraiton file `appsettings.json`
- `Domain` - business logic, platform independent
- `Infrastructure` - platform dependent implementation. Contains specific implementation of each data acquirer's logic
- `Tests` - unit tests and integration tests

The majority of code base is shared among the three out-of-the box working acquirers: Twitter, Reddit and custom dataset loader. The differences are only in the entrypoint project `WebApi.*` which differently sets up dependency injection container. Otherwise all the code is the same.

Data acquirer code base consists of the following classes (their relationships are depicted in Figure 3.4):

- `RegistrationService` - Service sending registration on startup
- `JobConfigurationUpdateListener` - Listener that listens for job notification.
- `TranslationService` - Service that calls the Azure Text Translator API via HTTP.
- `JobManager` - Service that makes sure that the job is running and the posts are being correctly produced.
- Twitter:
  - `TwitterBatchLoader` - Service that loads the data from the API.
  - `TwitterContextProvider` - Creates context of the whole acquirer.

  - `TwitterDataAcquirer` - Class responsible for orchestrating twitter batches and creating twitter context.

- Reddit: `RedditDataAcquirer` - Service that loads data from the Reddit api.
- CustomDataset: `CustomDataAcquirer` - Service that connects to the Minio server and downloads given datasets.

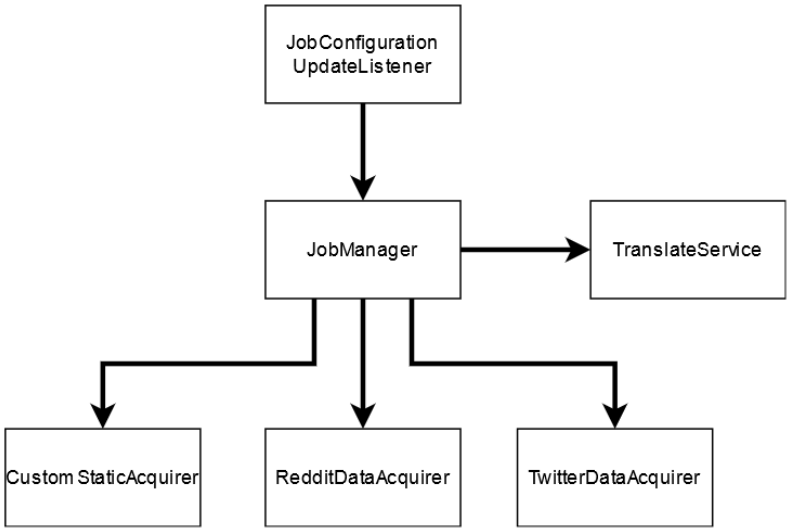


Figure 3.4: Relationships among major classes of Data acquirer

3.6.3 Configuration

The configuration of an acquirer is stored in the project `WebApi.*` in the file `appsettings.json`. It does not require any user input. The configuration is distributed via a dependency injection container. Each option of the configuration needs to be bound to the file. This binding is in the project `Application` in the file `DataAcquisitionServiceWebApiBuilder.cs`. The most notable configuration objects are:

- `ComponentOptions` - This object contains identity of the acquirer and Kafka topics which it uses.
- `TranslatorOptions` - This object contains settings of the translator service. Endpoint and subscription key can be found in a Azure Text Translator resource.
- `MinioOptions` - URL and credentials of the Minio server.

3.6.4 Build + Run

Navigate to the source code, specifically to the `acquisition/DataAcquirer/Web.*` and type in the following commands:

- `dotnet restore` - downloads all third party libraries
- `dotnet build` - build the projects
- `dotnet run` - run the projects

3.6.5 Communication

JMS uses Kafka for the communication depicted in Figure 3.5

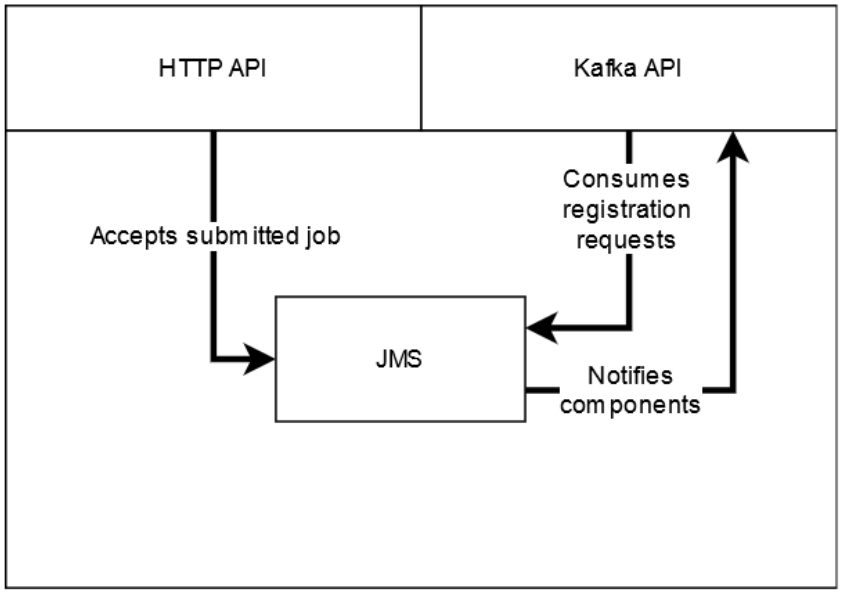


Figure 3.5: Kafka interface is used to listen to job notification, produce posts and register the component

Kafka Interface

Data acquirers listens for two types of job notification (see Appendix A.1.3): start job and stop job.

The start job notification has the following fields:

- `jobId` - Id of the job
- `command` - In case of start job, it is always `START`
- `attributes` - attributes of the acquirer. It is discussed in more detail below.

- `outputChannelNames` - An array of topics that the acquirer produce posts into.

NOTE: The start notification is the same for analysers too.

Each acquirer requires different attributes:

Twitter Data Acquirer:

- `TopicQuery` - name of the topic
- `Translate` - is `"true"` when the post should be translated
- `ApiKey` - Api key used to identify application to the Twitter API.
- `ApiSecretKey` - Api secret used to authenticate the application
- `AccessToken` - Access token issued by twitter application.
- `AccessTokenSecret` - Acess token secret used for authentication.

Example:

```
{
  ...
  "attributes": {
    "TopicQuery":"weather pocasi",
    "Translate":"true",
    "ApiKey":"123erbs",
    "ApiSecret":"sdsfabnakaler"
    "AccessToken":"assadfsa23234dafadsfn1234123",
    "AccessTokenSecret":"ghfgdr4350234daf"
    }
}
```

The credentials showed do not represent valid credentials.

Reddit Data Acquirer

- `TopicQuery` - name of the topic
- `Translate` - is `"true"` when the post should be translated
- `appId` - Id of the application registered with twitter
- `appSecret` - Secret used to authorize the application.
- `refreshToken` - Token which is used to not let the session expire.

Example:

```
{
  ...
  "attributes": {
    "TopicQuery":"weather;pocasi",
    "Translate":"true",
    "appId":"mffSocnetoApp",
    "appSecret":"adf289e8lmfals",
    "refreshToken":"klfoiowen1231"
    }
}
```

The credentials showed do not represent valid credentials.

Custom Static Data Acquirer:

- `Translate` - is `"true"` when the post should be translated
- `bucketName` - Name of the bucket in which is the custom data stored.
- `objectName` - Name of the dataset file
- `mappingName` - Name of the file with mapping

Example:

```
{
  ...
  "attributes": {
    "Translate":"true",
    "bucketName":"example-dataset",
    "objectName":"myData.json",
    "mappingName":"myData.mapping"
    }
}
```

The stop job notification has the following fields:

- `jobId` - Id of the job
- `command` - In case of stop job, it is always `Stop`

Outgoing Communication

Registration Acquirer needs to register itself on startup. The registration request was already discussed in the section JMS subsection Outgoing communication (see Section 3.5.5).

Posts Data acquirer produces posts (see Appendix A.1.4) with the following fields:

- `id` - Guid of the post
- `originalId` - Original id of the post issued by the respective social network.
- `jobId` - Id of the job in which context this post was produced
- `text` - Text of the post.
- `originalText` - If the translation is activated and the post was not originally in English, this field contains the original text and the field `text` contains the translation.
- `authorId` - Id of the author. In case of anonymous author, value "0" is used.
- `language` - Language of the post in ISO 639-1 format.
- `datetime` - Datetime when to post was created conforming to format `YYYY-mm-DDTHH:MM:SS`.

The posts are typically produced to two channels: one that is consumed by the storage and the other consumed by all analysers since analysers produce only analyses, not the posts. These two entities are joined in the storage. Names of the channels are configurable and are discussed in the configuration part of Data acquirer (Section 3.6.3).

3.7 Data Analysers

A data analyser is a component that consumes posts and produces per-post analysis. Socneto comes with two major analysis components: Topic modelling and Sentiment analysis. The implementation is common for both components. The functionality that is different is discussed in Chapter 4.

3.7.1 Requirements and Dependencies

Data acquirers require the following tools for its proper function:

- Kafka 2.4.0
- python 3.8
- Topic modelling trained model: en_core_web_lg-2.2.5[22] downloaded automatically upon building
- Trained sentiment model, also downloaded upon building.

They communicate with the following services:

- Storage service (see Section 3.8)
- Job management service (see Section 3.5)

[22] `https://github.com/explosion/spacy-models/releases/download/en_core_web_lg-2.2.5/en_core_web_lg-2.2.5.tar.gz`

3.7.2 Code

The code wrapping libraries discussed in Chapter 4 can be found in the `main.py` file. There are three notable functions:

- `register_itself` - sends registration requests (see Section A.1.1) whose attributes contains element `outputFormat` with the structure of format of the output analysis.
- `process_acquired_data` - starts consuming topic where posts appear.
- `analyse` - invokes analysis.

3.7.3 Build + Run

Topic modelling:

- `pip install -r requirements.txt` - installs requirements
- `wget -O en_core_web_lg-2.2.5.tar.gz https://github.com/explosion/spacy-models/releases/d` - downloads the topic modelling trained model.
- `pip install en_core_web_lg-2.2.5.tar.gz` - installs the model
- `python main.py`

3.7.4 Configuration

The script `main.py` accepts the following parameters:

- `--server_address` - address of the Kafka server
- `--input_topic` - topic from which the analysers consume the posts.
- `--output_topic` - topic to which the analysers produce analysis for the post (not the post itself). Set to the database input topic.
- `--registration_topic` - topic to which the registration request is sent.

3.7.5 Communication

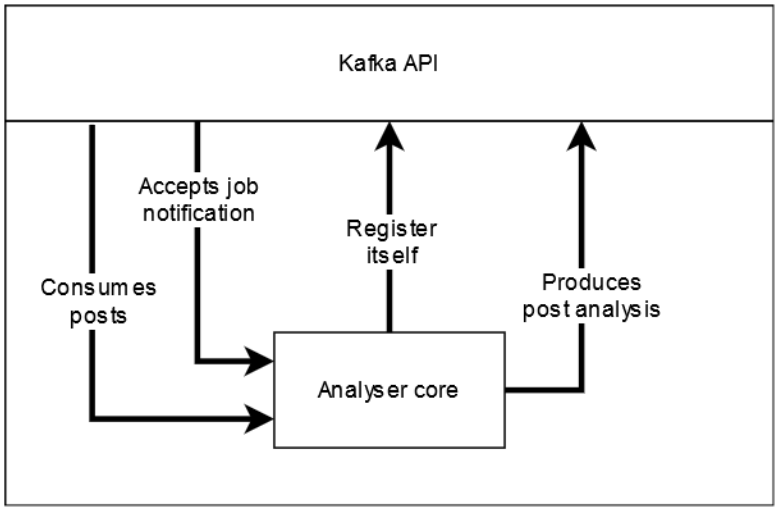


Figure 3.6: Kafka interface is used to listen to job notification, produce analysis and register the component

JMS uses Kafka for the communication depicted in Figure 3.6

Kafka Interface

The simplicity of the analysers allows them to ignore job notifications. The only entity they consume is a post (see Appendix A.1.4) which has been already discussed in detail in Section 3.6.5.

Outgoing Communication

Registration An analyser needs to register itself on startup. The registration request was already discussed in Section 3.5.5. The only addition is that they in element `outputFormat`.

Output format Every analyser can produce any number of results, but the result value has to have one of the six possible formats:

- `textValue` - a single string value
- `numberValue` - a single number value
- `textListValue` - a list of string values
- `numberListValue` - a list of number values

- `textMapValue` - a map of a string key and a string value
- `numberMapValue` - a map of a string key and a string value

During registration each analyser has to pass a map with result name and its value type. This information is used during a chart definition on FE. The result has to be always in this format (see 3.7.5 Analysis paragraph).

Analysis Analysers produce analysis entity (see Appendix A.1.5) with the following fields:

- `postId` - Guid of the related post
- `jobId` - Id of the job in which context this post was produced
- `componentId` - Analysers component Id
- `results` - Actual result of the analysis enclosed in an element with the analysis name.

Example - sentiment analysis:

```
{
  "postId": "1234dsfgdsf",
  "jobId": "4df8165f-9461-4cdc-b6bc-b5fa0414319b",
  "componentId": "s_analyser_1",
  "results": {
    "polarity": {
        "numberValue": 1,
        "textValue": null,
        "numberListValue": null,
        "textListValue": null,
        "numberMapValue": null,
        "textMapValue": null
    }
  }
}
```

Example - topic modelling:

```
{
  "postId": "1234dsfgdsf",
  "jobId": "4df8165f-9461-4cdc-b6bc-b5fa0414319b",
  "componentId": "topic_analyser_1",
  "results": {
    "polarity": {
        "numberValue": null,
        "textValue": null,
        "numberListValue": null,
        "textListValue": ['food','bread',...],
        "numberMapValue": null,
        "textMapValue": null
    }
  }
}
```

3.8 Storage

Storage component is used for storing internal platform data, acquired posts and analyses. This module provides an abstraction of physical databases for the rest of the components. Every other component can communicate with the internal part of the storage via REST API and all internal entities support a custom subset of CRUD operations. Posts and analyses can be stored via messaging, but they can be queried only via REST API. This approach should help to increase the performance of the whole platform and loose coupling between components.

3.8.1 Requirements and Dependencies

Storage module needs a relational database for storing internal platform data, but posts and analyses must be stored in a storage, where it is easier to query those data, so the second storage dependency is a search engine.

- Run: Java[23]
  - Minimal version of Java: 11
  - Minimal version of Maven: 3.3.9[24]
- Relation database: PostgreSQL[25]
  - Recommended version of PostgreSQL: 9.6

[23] https://www.java.com/en/
[24] https://maven.apache.org
[25] https://www.postgresql.org

- Search engine: Elasticsearch[26]
  - Recommended version of Elasticsearch: 7.4.2
- Kafka broker
  - Platform component
- Messaging
  - Platform component

3.8.2 Code

For implementation Java 11 language with Spring Boot framework[27] is used. Spring Boot framework was chosen due to easier and faster development, because it is built on the Inversion of Control principle[28] and also it provides an abstraction of databases and other dependencies. For building of the whole application multi-module Maven is used.

During the design phase, we included MongoDB[29] as NoSQL database for posts and analyses. After some time we have not found any reasonable real-world use case for duplication of data. (Synchronization performance was not great and the storage was using more physical disc space than needed.) Also, Elasticsearch has made internal improvements, so it is more often used as primary storage.

User authentication and authorization is not implemented in the current version of Socneto, because it is beyond its primary scope. If there is some identity provider in the future, it will be simple to improve security with Spring Security[30]. As a temporal solution, users are imported into the database during the start of the application. This solution should only simulate user authentication. The correct solution is more complex, because it has to be reused in many components of the whole platform.

Storage component is divided into four modules. Every module has a different usage: storage of internal data, storage of posts and analyses, Kafka consumer and REST controllers. The main pattern besides IOC is used MVC with more layers: controllers, validation on data transfer objects, domain layer, persistence.

Module internal-storage This module can be used for CRUD operations with internal data (components, jobs, users, etc.). For storing data PostgreSQL is used and communication are done via Spring framework. The framework adds an abstraction of storage for this module.

- Major classes:
  - `ComponentDto` - description of Analyzer or Acquirer component
  - `ComponentDtoService` - operations for `ComponentDto`

[26] `www.elastic.co/elasticsearch`
[27] `https://spring.io`
[28] `https://en.wikipedia.org/wiki/Inversionofcontrol`
[29] `https://www.mongodb.com`
[30] `https://spring.io/projects/spring-security`

  - `ComponentJobConfigDto` - configuration of components for a single job
  - `ComponentJobConfigDtoService` - operations for `ComponentJobConfigDto`
  - `ComponentJobMetadataDto` - object, where every component can store its metadata for a single job
  - `ComponentJobMetadataDtoService` - operations for `ComponentJobMetadataDto`
  - `JobViewDto` - view configuration of a single job
  - `JobViewDtoService` - operations for `JobViewDto`
  - `UserDto` - internal user and password
  - `UserDtoService` - operations for UserDto
- App properties config:
  - `spring.datasource.url` - JDBC string to database
  - `spring.datasource.username` - database user
  - `spring.datasource.password` - database password
- Spring concept:
  - Spring Boot Starter JDBC
- Dependencies:
  - PostgreSQL database

Module analysis-results

Module analysis-results uses Elasticsearch for storing posts and analyses. These objects can be queried in many ways:

Posts support search queries with pagination and aggregations by authors, language, etc.

Query `POST_AGGREGATION`:

- `COUNT_PER_TIME` - aggregation of posts over time
- `COUNT_PER_AUTHOR` - aggregation of posts per authors
- `COUNT_PER_LANGUAGE` - aggregation of posts per languages

Analyses can be queried in a more complex way. The basic list with results can be queried with pagination, but also aggregation over list and map results is supported.

Query `AGGREGATION`:

- `MAP_SUM` - sums occurences in `MapValue` result
- `LIST_COUNT` - sums occurences in `ListValue` result

Query `LIST`:

    - `LIST` - list of analyset result values
    - `LIST_WITH_TIME` - list of analyset result values with time

- Major classes:
  - `SearchPostDto` - an acquired post entity
  - `SearchPostDtoService` - operations for SearchAnalysisDtoService
  - `SearchAnalysisDto` - an analysis entity
  - `SearchAnalysisResultDto` - representation of analysis result value
  - `SearchAnalysisDtoService` - operations for SearchAnalysisDtoService
  - `ResultRequest` - a request for list values or agrregations
  - `ResultService` - an aggregation service for posts and analyses
  - `ListWithCount` - a result of list request with a total count filed
- App properties config:
  - `elasticsearch.host` - Elasticsearch host
  - `elasticsearch.port` - Elasticsearch port
- Spring concept:
  - Spring Data Elasticsearch
- Dependencies:
  - Elasticsearch

Module storage-messaging

This module is created for receiving posts and analysis messages from the Kafka message broker. Every message is parsed into an internal object and stored by the analysis-results module. The module consists of two configurable receivers for different topics.

- Major classes:
  - `AnalysisReceiver` - a consumer of analyses messages
  - `PostReceiver` - a consumer of posts messages
  - `KafkaConsumerConfig` - a consumer factory provider
- App properties config:
  - `spring.messaging.bootstrap-servers` - url to Kafka server
  - `app.topic.toDbRaw` - a topic for post messages
  - `app.topic.toDbAnalyzed` - a topic for analysis messages
- Spring concept:

  - Spring Kafka

- Dependencies:
  - `analysis-results` module
  - Kafka message broker

Module storage-web

Storage-web module contains all REST endpoints of the storage component. It uses CRUD services from Internal-storage module and search services from Analysis-storage module.

Also, this module is the only one which can be run. For execution Spring Boot Starter is used, which starts all modules together, loads configuration, connects to Kafka, database, etc.

Validation of input API objects is done via `javax.*` annotations. This approach helps with validation without a huge amount of code and it is built on Spring framework.

- Major classes:
  - `ComponentController` - REST API for components
  - `ComponentJobConfigController` - REST API for component job configs
  - `ComponentJobMetadataController` - REST API for component job metadata
  - `HealthCheckController` - health check endpoint
  - `JobController` - REST API for jobs
  - `JobViewController` - REST API for job view
  - `PostController` - REST API for posts
  - `ResultsController` - REST API for results
  - `UserController` - REST API for users
- App properties config:
  - `app.componentId` - ID of Storage component
  - `server.port` - Port of Storage component
- Spring concept:
  - Spring Boot Starter Web
- Dependencies:
  - `internal-storage`
  - `analysis-results`

3.8.3 Build + Run

- Build: `maven package -f pom.xml`
- Run: `java -jar storage-web-1.0.0-SNAPSHOT.jar`

3.8.4 Communication

The storage component provides two APIs. Kafka API is only for acquired posts and analyses. This API provides async communication for acquirers and analysers. For querying posts and analyses the REST API must be used. Both endpoints support pagination when it is needed. REST API is also used for CRUD operations of internal data. For better understanding, the flow is described in Diagram 3.7.

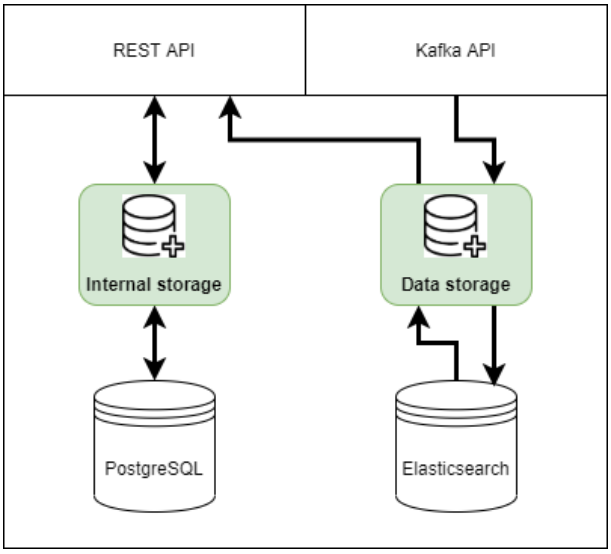


Figure 3.7: Storage architecture uses two databases forming polyglot storage

- REST
  - Swagger: `<storage-host>:<storage-port>/swagger-ui.html`
  - Exported swagger documentation: `docs/api/storage-api.pdf`
- Messaging topics
  - Posts: `job_management.component_data_input.storage_db`
  - Analyses: `job_management.component_data_analyzed_input.storage_db`

3.9 Monitoring

The platform contains many components and, hence, a unified logging is useful when an application is deployed. ELK Stack[31] is used for platform monitoring purposes. Every component can log its events over HTTP (port: 9999) or through Kafka messaging (topic: log_collector.system_metrics) into Logstash[32]. Only Log entities (see Appendix A.1.6) with

[31] `https://www.elastic.co/what-is/elk-stack`
[32] `https://www.elastic.co/logstash`

defined format must be used. When a log is received by Logstash, it is stored into Elasticsearch. (The platform uses the same instance for posts, analyses, and logs.) A user can open Kibana[33] dashboard in a browser and see logs from all components in one place. Architecture is described in Diagram 5.1.

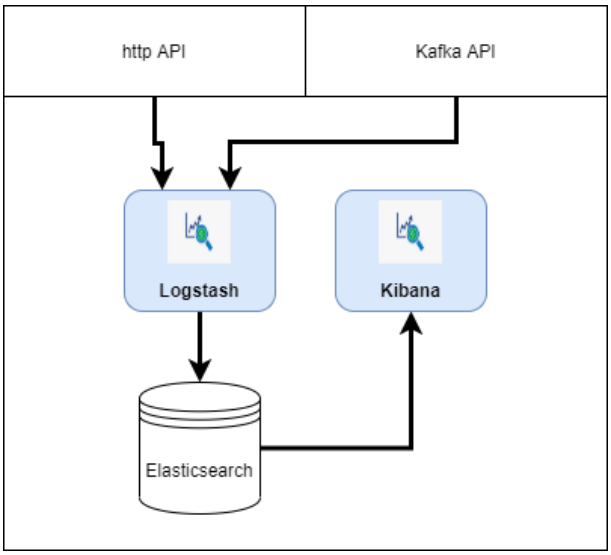


Figure 3.8: Diagram of monitoring interfaces. Logstash is connected to HTTP API and Kafka API and feed the data to database which is used by Kibana

3.9.1 Requirements and Dependencies

- Logstash
  - Minimal version of Logstash: 7.4.2
- Kibana
  - Recommended version of Kibana: 7.4.2
- Elasticsearch
  - Recommended version of Elasticsearch: 7.4.2
- Kafka messaging
  - Platform component

[33] `https://www.elastic.co/kibana`

3.9.2 Configuration

Logstash Configuration

```
input {
  http {
    id => "http_in"
    port => 9999
  }
  kafka {
    bootstrap_servers => "kafka:9092"
    topics => "log_collector.system_metrics"
    codec => "json"
  }
}

filter {
  if ![attributes] {
    mutate {
      remove_field => [ "attributes" ]
    }
  }
}

output {
  elasticsearch {
    hosts => [ "elasticsearch:9200" ]
  }
}
```

Kibana Dashboard

A user can specify his own dashboard with logs in Kibana. Possible fields are:

- `componentId` - a name of a component
- `eventType` - FATAL, ERROR, WARN, INFO, METRIC
- `eventName` - a custom event name
- `message` - a custom message
- `timestamp` - a time when the log was created
- `attributes` - any JSON object with additional info

3.10 Extensibility

Socneto can be extended with custom data acquirers or data analysers. The components must implement the following stages properly.

3.10.1 Contract

Registration To component sends registration request (see Section 3.5.5 to the topic `job_management.registration.request` consumed by JMS (see Section 3.5). This registration request makes the component discoverable. Without the registration, component could not be selected by the user.

In case that the component was already registered and crashed. Upon the re registration, the component receives all the job notifications of running jobs that was received during the the component lifetime. This behaviour makes it easier for component to recover since it does not have to persist the notification itself. This behavior implies that the registration is idempotent therefore multiple re registration will not cause problems.

Starting a Job After successful registration, the jobs notification (see Appendix A.1.3) starts flowing event to the channel specified in the
`updateChannelName` field in the registration request. The processing of those jobs has been already discussed in respective chapter of the analysing components (see Section 3.7.5) and data acquiring components (see Section 3.6.5).

Data acquirers and data analysers react differently to the job notification. While data acquirers are expected to start producing posts (see Appendix A.1.4) to all topics present in the `outputChannelNames` array the acquirers need to wait for some posts from `inputChannelName` to come first then analyse them and produce the analysis (see Appendix A.1.5). In case of data acquirers, the job notification element `outputChannelNames` contain at least two topics: one represent a topic which storage consume to store raw posts. The others are then for each selected data analyser. In case of data analyser, the `outputChannelNames` is usually only one — the storage input topic.

Stopping a Job The jobs can be stopped using also the job notification with a command element `command` set to `Stop`. This signals the component that can safely stop listening to the respective input topics. In case of analysers, stopping should be delayed for some time until the all of the messages are not consumed. Data acquirers should be shut down immediately.

Chapter 4

Text Analysis

4.1 Introduction

In this chapter, the social network posts text analyses originally featured in Socneto are described.

The solved problems are presented in the section 4.2 of this chapter. First, the motivation behind the choice of these problems and other possible types of analyses is discussed. Furthermore, a comparison of different input data types and their limitations is given. In the sections 4.3 and 4.4, the chosen problems are described precisely together with selected solutions. A quick comparison of possible methods, the theory behind them and implementation details are provided for each problem.

Section 4.4.5 of this chapter is dedicated to a discussion of results and a conclusion.

4.2 Natural Language Processing - Possibilities and Drawbacks

As main goal of Socneto is analysis of posts from social networks, this section provides an introduction to a natural language processing (NLP). In Deep Learning book [1], NLP is defined as *"...the use of human languages, such as English or French, by a computer."*.

4.2.1 What Is Possible

The main intention of Socneto is to present automatically analysed data from various social networks. Socneto is extensible in many directions including new social networks and new types of analyses, but some analysis types are provided by default. Chapter 3 of this documentation describe how to choose, obtain and store data from Reddit and Twitter. The question is, what to do with them.

NLP offers a variety of problems to solve. Some of them are beyond our focus, for instance, conversion between written and oral language. We could imagine using this to analyse short videos published on Twitter. This could give us more text for analyses, but we focus first on basic analysis of the text we have.

Another possible way of natural language processing is **syntax analysis** - examination of the formal structure of words, sentences and larger parts of the text. *Sentence breaking* or *word segmentation*, which divides the text into smaller, further analysed parts are included in the pipeline for text preprocessing. The task of *lemmatization*, also used in preprocessing, lies in finding a base form of a given words, meaning for example nominative of singular for nouns or infinitive for verbs. The analysis can then continue with *morphological segmentation*, which divides words into smaller parts, morphemes. To return to whole sentences, different types of trees are used to represent the structure of sentences and dependencies within a sentence (see Figure 4.1).

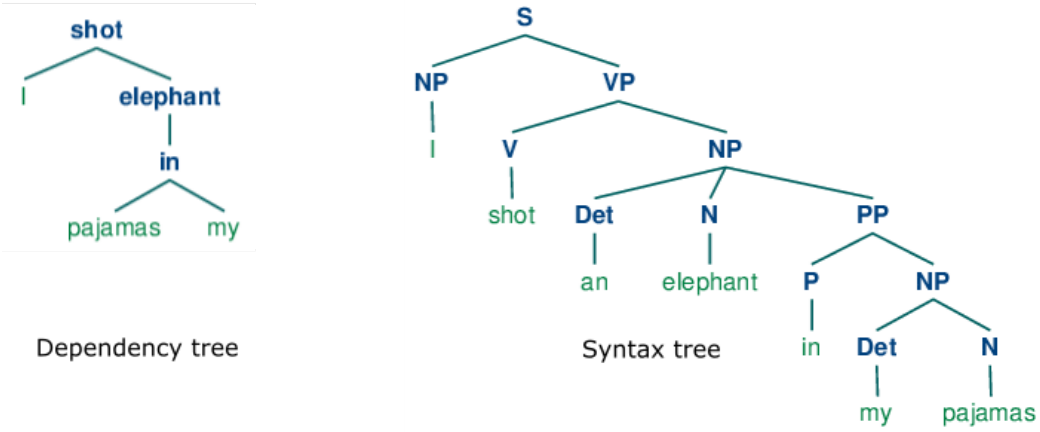


Figure 4.1: An example of the syntax and dependency tree. Dependency tree, as the name indicates, describes dependencies between words. Such dependencies are of various types, for example an elephant in the example is an direct object of the shooting action. A root of such tree is typically a predicate of the sentence. On the other hand, syntax tree represents syntactic structure of the sentence according to the grammar. The root of the tree is *sentence*, which is split to noun and verb phrase. These can be further divided into phrases compound from particular instances of parts of speech (e.g. nouns, adverbs, verbs, prepositions etc).
Source: [2]

All previously mentioned treatments and many others are available for syntax analysis and most of them are needed for whatever task is performed, but they do not provide much information about the meaning of the text, at least for a human reader.

More suitable for this purpose is the part of NLP dedicated to **semantic analysis**. Again, it contains various methods from natural language text generation to recognition of homonymy or polysemy of given words. It could solve sophisticated assignments as answering questions about the input text document or translation. Another possible task is to find in the text so-called named entities - like persons, months or cites - or linking these entities to some knowledge base. We can try to recognize the formality of the text, its sentiment, main

topics or even try to automatically modify the text to be clearer.

It is not necessary to be limited to text analysis only. In addition to speech recognition, it is also possible to engage in computer vision for extracting information from pictures. The importance of social networks data does not lie only in user contribution, but also in the metadata. Interesting results can be obtained via study of different user groups, their relationships to other groups, or different topics and opinions together with some demographic data.

4.2.2 The Nature of the Data

Social networks have become important communication medium in recent times. They have the power to influence many people whether it is shopping, politics, environmental responsibility or the newest trends. So there are many reasons for understanding them. Data stored in networks are growing every second and it is not in human power to consume all of them. The problem of obtaining this data is solved by the acquiring mechanism of our platform as described in other sections of the documentation (chapter 3).

Inputs to natural language processing can be data of many kinds. Both syntactic and semantic models can be trained on *treebanks*. An example of a treebank is in Figure 4.2). A treebank is a parsed corpus with various types of annotations. For machine translation documents with many language versions are appropriate.

Eurotra [4] is, for example, almost twenty years lasting project of the European Commission dedicated to machine translation started on the latest seventieths. EU administrative documents in French and English served as datasets.

Topic modeling is typically applied on a big set of medium length documents (e.g. scientific articles). In contrast, data from tweets and Reddit are different. Tweets are short snippets of text full of odd characters, newlines, and ends of lines. They contain pictures, emoji, a mixture of different languages, slang expressions, and grammatical errors. And they are very short, sometimes only one sentence long, few hashtags and a link or a picture. Texts from Reddit are larger, but there is a challenging aspect of both social sites. The data has tree structure: Tweet and retweets or comments in the case of Twitter and main thread post with replies in the case of Reddit. These replies or comments can be even shorter than their parent post and/or do not mention previously said things explicitly. For performing a reasonable work of analysis it is necessary to have all these pieces of information together.

Questions about the meaning of the text have another difficulty, which is, however, part of their attraction. They do not have always an unambiguous response. In the case of topic modeling, the goal is to identify general topics of a given text. These topics are not necessarily mentioned explicitly in the text and are then obviously hard to agree on them, especially if there is no given set of topics to choose from. Semantic analysis is a little bit easier. Evaluation of extremely positive or negative texts is mostly consistent among people. But of course, there is text somewhere in between, where classification is not so simple. Sentiment analysis must also deal with things like sarcasm or irony. It is not so complicated to learn a model when positive words like “good”, “nice”, “favorite” or “love” means positive sentiment of the classified text, but it is much harder to distinguish situations, where they mean the opposite.

The second problem connected with data quantity is a human inability to read them all. For this purpose, out-of-the-box analysis is aggregated to get a better overview.

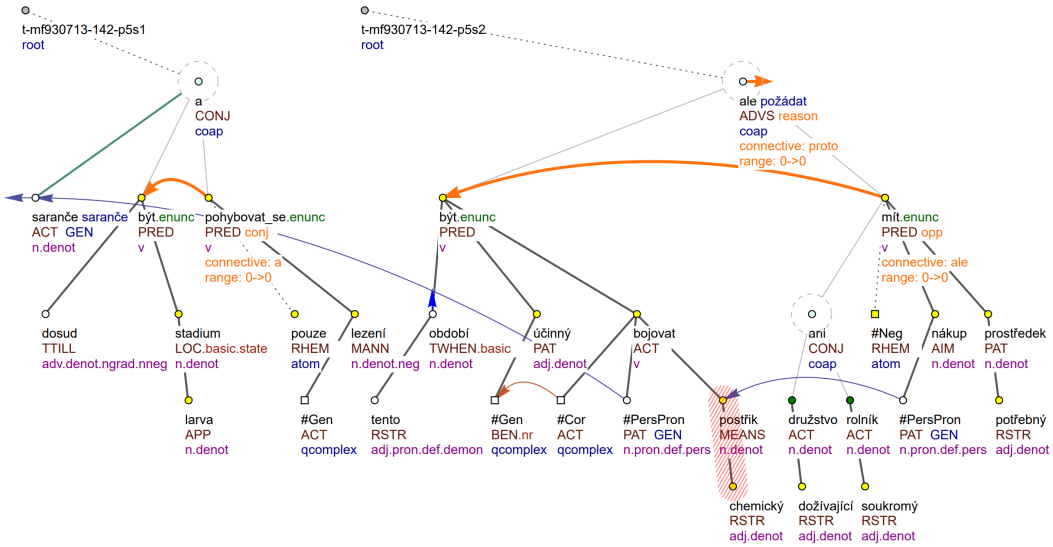


Figure 4.2: Prague dependency treebank example [3] for the sentences: *Sarančata jsou doposud ve stadiu larev a pohybují se pouze lezením. V tomto období je účinné bojovat proti nim chemickými postřiky, ale dožívající družstva ani soukromí rolníci nemají na jejich nákup potřebné prostředky.* This treebank contains dependency trees, but is is just one of many possibilities. This example is from Prague dependency treebank, which offers different layers of annotations. Red strips over words *chemický* and *postřik* marks multiword phrase, conjunction between *rolník* and *družstvo* is expressed as by one type of nodes, blue lines denotes coreference etc.

4.2.3 Choices for Socneto

For these reasons we decided to focus on semantic types of analysis of data and to present two types of semantic analysis - *sentiment analysis* and *topic modeling*. Both these analyses helps us with extracting a meaning and their results can be aggregated in a reasonable way. Topic modeling allows us to identify popular topics themselves or to find other often related topics to the given one. So we can find out, what are people on social sites talking about. The second provided analysis - *sentiment analysis* gives us an overview of their opinion about those topics. Both types of analysis are applied to a big bunch of data and return summary analysis, which is another useful feature. There are typically too many user posts to analyse, so inspecting each of them separately would produce an overwhelming amount of information and this does not solve the need for automatic analysis.

4.2.4 Machine Learning

As in almost every machine learning application, possible methods can be supervised or unsupervised. Supervised methods require a dataset (sometimes called gold data), where examples are problem instances together with correct answers. The learning algorithm then tries to find patterns or rules for classification and (iteratively) compares its answers with the correct ones. On the contrary, we need no correct answers for unsupervised learning. It is typically based on searching in the space of parameters to minimize some function (*loss function).* And it is also possible to solve tasks without machine learning at all by creating a set of hand-made rules. This approach is historically the first, but certainly not in the results.

4.3 Topic Modeling (TM)

Topic modeling task can be described by one of the following definitions:

1. A classification of a bunch of documents into a given number of topics together with learning typical keywords for each topic.

   - **Input:** A set of documents, desired number of topics (integer, greater than zero).
   - **Output:** Estimated probability distribution of topics for each document, estimated probability distribution of words for each topic.

2. Finding a list of words (=topics) associated with given data and their importance or distribution (which is what Socneto wants to do).

   - **Input:** Text for analysis (all tweets or posts together.)
   - **Output:** A set of words best representing input.

   As there exists solution for the problem A, there also exists solution for problem B. It is true, because problem B can be transferred to the problem A by setting a count of topics to 1 and searching only for keywords. This does not solve the problem of finding abstraction topics not mentioned in the text but gives a quite good overview of the input data.

3. A classification of texts into given topics.

   - **Input:** Texts for classification, topics to choose from
   - **Output:** One label from given topics for each text.

4.3.1 Methods

Regarding supervised methods, it is possible to use any known supervised classification algorithm. Among others support vector machines, logistic regression or neural networks belong here. The intended usage of Socneto is an application on data from many different areas, so we cannot define topics in advance, which is the reason, why our problem is not following definition C.

This determines us to use unsupervised methods. For TM defined as in definition A, two mainly used methods are Latent Semantic Analysis (LSA) [5] and Latent Dirichlet Allocation (LDA) [6]. They are both based on the same assumptions, but LDA is an improved version of LSA with better performance. Topic modeling as in definition B was chosen for implementation, so we use LDA for it. A detailed description of theory and implementation is available in the following section.

Because LDA used in the described way (i.e. following the definition B) can find only words explicitly expressed in the text, we wanted to enrich the analysis by some more abstract topics. It can be done via solving *entity linking* problem, which assigns hierarchical structure of categories to found entities/topics. The structure for classification is called the knowledge base and can be extracted automatically for example from Wikipedia. Or we can solve another problem - *Named Entity Recognition* (NER). Algorithms for NER are able to tag each named entity by its category. Categories vary depending on implementation and training data, but categories like city, proper name, date, language or month can occur. This is supervised learning, but with a limited quantity of target classes and allows us to search for topics in specific categories provided by NER.

4.3.2 NER

The task of finding named entities in the text and classifying them into the right category.

- **Input:** Unstructured text (= natural language), categories
- **Output:** Classification into right category if exists for all words (see Figure 4.3)

NER can be divided into two subtasks - identification of entities and their classification. As for every classification task, we can use supervised machine learning or rule-based system written by an expert. Another useful feature of this analysis is to recognize not only words but whole phrases (e.g. National Park). This analysis can served as another tool for exploring the data. It allows filtering topics by categories like *person* or *place*. It can also helps to improve topic searching, as named entities are typically important int the text and NER is able to find pronoun references to them. Without such references, the word can occur just for one time and could appear as unimportant to LDA.

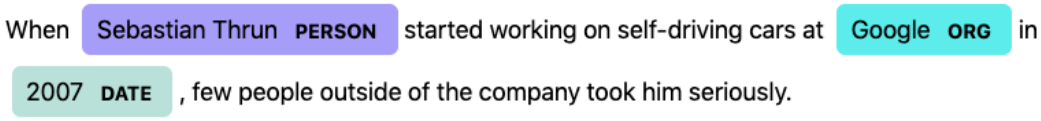


Figure 4.3: An output of the Named Entity Recognition. Example from the SpaCy library documentation

Class	Description
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.
CARDINAL	Numerals that do not fall under another type.

Table 4.1: Classes in NER (spaCy)

Our analysis provides classification into 18 classes (see Table 4.1) as we used third-party library SpaCy[1], where these classes are predefined.

The used library, SpaCy, has pre-trained models for tagging and entity recognition for different languages. Algorithms in this library are not exactly based on one article, that could be named. Based on the GitHub discussion [7], spaCy implementation of NER can be shortly described as classification using Convolutional Neural Networks (CNN) as in paper [8], but residual connections are used instead of dilatation and there are some other minor differences. Together with residual connections, there are two other important features - word embedding strategy using subword features and a transition-based approach. All three principles will be shortly described now.

Word Embeddings with Subwords

Word embeddings are numbers or numerical vectors representing some categorical features. Their interesting feature is that when the source categories are near to each other, their embeddings are as well. Each dimension of the vector represents a different property of the

[1] `www.spacy.io`

word. One famous example for all: Embedding for the word 'Queen' can be almost exactly computed by the following formula on respective embeddings $king - man + woman$. A limitation of the original paper on embeddings [9] concept was its inability to represent the morphological structure of the word. Embeddings were created based on the corpus and their occurrence in a similar context. It means, that no embedding exists for a newly seen word, even if it is morphologically very similar to a known one. Numeric representation of n-gram taken into account during the computation of an embedding is a solution to the problem.[10] After this step, all words are represented by their embeddings and the following processing is performed only on them.

CNN with Residual Connections

The convolutional neural network (see Chapter 9 in [1]) is a type of deep neural network with good ability to represent structures in the space. The input of this network is not a vector but a matrix (like a picture represented by values of its pixels). There is one single path from the input to the output of the network and every layer consists in the application of a filter or multiplication by a kernel on the sliding window.

In the case of residual connections, the network is split into smaller parts. To avoid vanishing of information during the flow through the network, new shortcut connections (= residual connections) are added between some nodes. This approach is widely used in image recognition but also spread to other fields (see Figure 4.4).

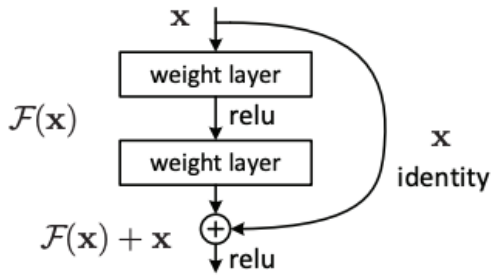


Figure 2. Residual learning: a building block.

Figure 4.4: An example of the network for image recognition with residual connections. Residual network is a chain of layers with connections between as in deep forward networks, but there is also added connection marked as x identity, because in this case the activation function on the residual is identity function, which creates shortcut for strengthening information flow through the network. *source: [11]*

Transition Based Approach

This approach consists of a sequence of small steps. In every step, only one word from the buffer is labeled or just one change to the configuration is applied [12].

4.3.3 LDA

As mentioned above, Socneto uses LDA in the following manner:

- **Input:** A list of words (extracted from the text)
- **Output:** Top n (in our case $n = 10$, but it can be changed) words describing input.

As the data are full of odd characters, newlines, and ends of lines, the first step is to clean the data. Our first trials have shown that adjectives are rarely useful, even indeed they make results messy and not informative at all. The same applies to pronouns, so words of both categories are removed. We can do this because the input of LDA is just a list of words, thus we can remove those words we don't want to be considered. We use the same pipeline described in the section 4.3.4 because it helps us with the filtration.

The previously mentioned LSA is an ancestor of LDA and it is better for showing basic ideas behind these two methods. LSA (same as LDA) is based on two assumptions:

- Every document contains a mixture of topics.
- Every topic is connected with different, but not necessarily disjoint, set of words.

LSA than creates a matrix of documents contra terms, where values are *tf_idf*-s, where *tf* stands for a term frequency

$$tf = \frac{term\_occurences}{number\_of\_words\_in\_document}$$

and this is count over the whole corpus of documents (do not forget, that LSA is primarily applied on the corpus of different documents), while *idf* is inverse document frequency

$$\frac{number\_of\_documents}{number\_of\_documents\_with\_term}.$$

Idf works as an evaluation of the importance of the word. The result is then:

$$tf\_idf = tf \cdot idf.$$

This matrix is then decomposed using *singular value decomposition* (SVD). Decomposition gives us three matrices - S, U and V. Matrix S is always diagonal and each element on the diagonal represents one topic. Matrices U and V are document-topic and term-topic matrices respectively (see Figure 4.5) and they contains probability distributions.

Selected k greatest elements of S, corresponding columns from U and rows from V gives us k most important topics. V than offers most important words for given topics.

LDA is based on the same assumptions. Unlike LSA, LDA assumes that topics and words follow Dirichlet's distribution. LDA tries to learn a model that best explains how data was generated.

4.3.4 Implementation

LDA model is created using package Gensim[2] built by NLP Centre of the Masaryk University [13] especially for topic modeling, which pratically the only possible solution. We considered two libraries for NER task - *NLTK*[3] and *spaCy*. As using spaCy is much more comfortable

[2] `www.radimrehurek.com/genisim`
[3] `www.nltk.org`

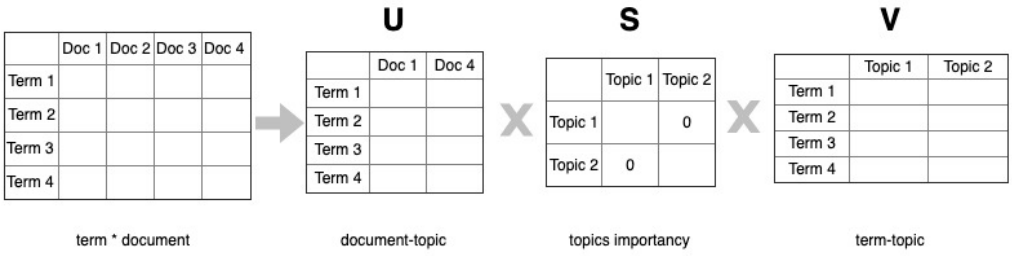


Figure 4.5: LSA matrix decomposition.

than NLTK, we chose spaCy. Besides, SpaCy also seems to have superior performance on Twitter-like data [14] (see Table 4.2). These two considered libraries are not state-of-the-art possibilities, but provide reasonable results. The performance can be improved by using for example Standford CoreNLP, which is more technically demanding, but offers better precision. Socneto, though, expects human receiving result and than recall is maybe more importnant metric than precision, as mentioned also in [14]. It is better to see all results and discard some wrongly classified than miss some results.

System Name	**Precision**	**Recall**	**F1 Score**
Stanford CoreNLP	0.526600541	0.453416149	0.487275761
Stanford CoreNLP (with Twitter POS tagger)	0.526600541	0.453416149	0.487275761
TwitterNER	0.661496966	0.380822981	0.483370288
OSU NLP	0.524096386	0.405279503	0.45709282
Stanford CoreNLP (with caseless models)	0.547077922	0.392468944	0.457052441
Stanford CoreNLP (with truecasing)	0.413084823	0.421583851	0.417291066
MITIE	0.322916667	0.457298137	0.378534704
spaCy	0.278140062	0.380822981	0.321481239
Polyglot	0.273080661	0.327251553	0.297722055
NLTK	0.149006623	0.331909938	0.205677171

Table 4.2: Comparison of available NER models for Twitter data. Data comes from Workshop on Noisy User-generated text [15] 2016.
source: [14]

SpaCy, in addition, offers a built-in preprocessing pipeline, which we used in both LDA and NER. Such a pipeline aims to transform the raw text into something more organized. In the case of spaCy, it contains a tokenizer, tagger, parser and named entity recognizer (see Figure 4.6), where:

- Tokenizer: Breaks the full text into individual tokens.
- Tagger: Tags each token with part-of-speech tag as noun, pronoun, adjective, punctuation etc. [16],

- Parser: Creates syntactic tree, finds noun phrases.
- Named Entity Recognizer (NER): Labels named entities.

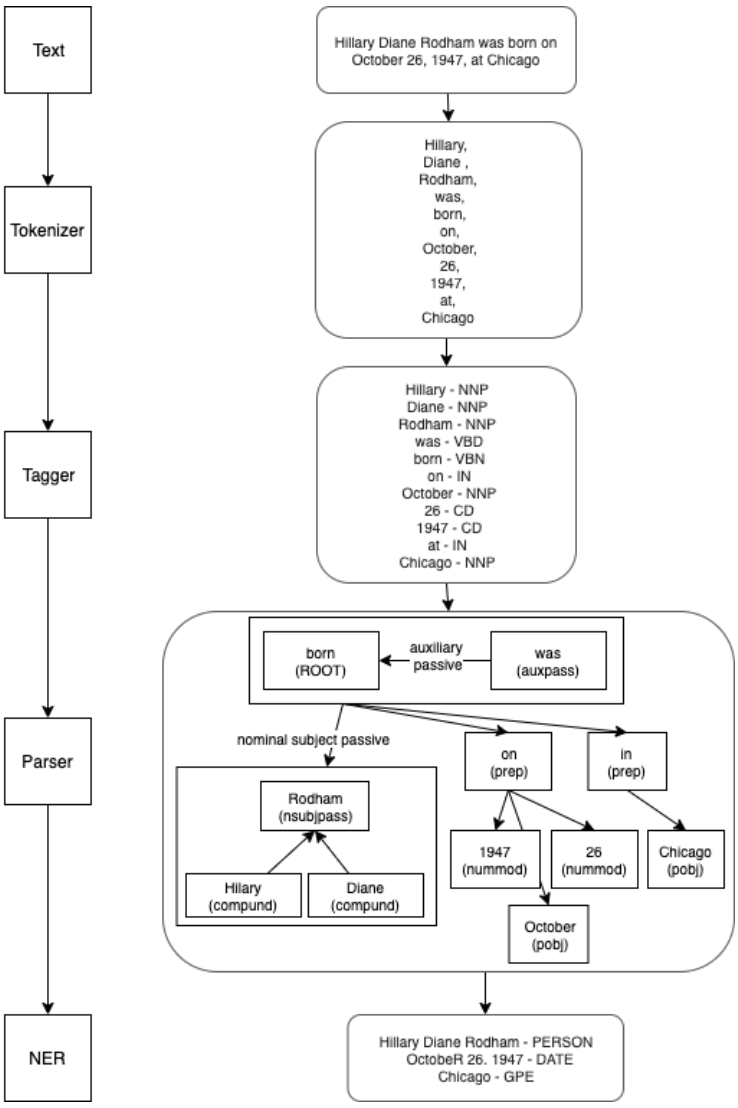


Figure 4.6: An example of processing in SpaCy pipeline.

4.4 Sentiment Analysis (SA)

We will consider the sentiment analysis task in the following form:

- **Input:** A snippet of text (article, tweet etc.)
- **Output:** Classification into one of classes - positive, neutral, negative.

Sometimes there is also considered so-called *subjectivity* [17]. It tries to classify if the opinion (both positive or negative) is objective or the author is personally interested and has strong emotions about his claims. For example, the following text could be recognized as objective: "The sound of this notebook is clear.", "The base is not stable enough." or "An internet connection in this area is bad." in contrast with "I hate the way the new touchpad works.". The subjectivity of the claim does not depend on its sentiment. This part of sentiment analysis was not included in our project yet, but it is one of the possible extensions.

4.4.1 Methods

According to NLP-progress page [18], actual state-of-the-art accuracy of this task on English datasets in the case of binary classification is above 95%. Our demands, however, are higher than just binary classification. Classifying into more classes is noticeably harder, so our expectations about model performance are lower. As in the case of the TM task, some gold data labels could be subject to dispute, but it is still easier to classify into one of three given categories.

The chosen methods for this type of analysis are the Bidirectional Encoder Representations from Transformers (BERT) . The first idea of BERT was published in the first half of 2019 by Google AI Language [19] and since then, it is a leading approach to various NLP tasks. Not only that its results in many tasks are new state-of-the-art numbers, but its main advantage is a universality and easy use for different types of prediction. This allows distribution of non-specific pre-trained models, while fine tuning on any task could be done just by adding one simple layer on the top. This practice, known as *transfer learning* has a long history in image processing. Unlike previous widely used models, BERT model just represents some universal knowledge about language. The next section is devoted to the further explanation of BERT ideas.

4.4.2 BERT

BERT model is built upon the idea of *transformers*. A transformer is a neural network with two parts - encoder and decoder. Contrary to other types of neural networks, the input of a transformer is the whole sentence of text. Inputs are fed to the network in the form of embeddings, and the transformer tries to learn how to encode the sentence in the way that decoder can reproduce the original sentence. The whole training is an oscillation between decoder learning the best way to decode encoder output and encoder trying to learn better representation for the given sentence. In BERT, only the encoder is used. BERT model has three main features to describe - usage of embeddings, masked language modeling (MLM) and next sentence prediction. Each of them is described in the following text.

Embeddings

Following the transformers paper [20], inputs are transformed into three types of embeddings: token embeddings, sentence embeddings and special transformer positional embeddings (see Figure 4.7). These embeddings serve as a technical simplification for other model parts.

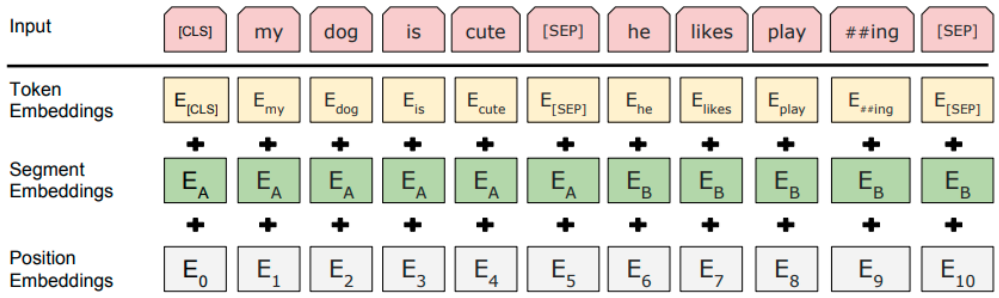


Figure 4.7: Structure of BERT input embeddings. Input is enriched of tokens for start and end of sentences and then represented by embeddings for each word (token embeddings), embeddings for sentence (segment embeddings, same for words in one sentence) and position embeddings (position of token in the text). *source: [19]*

MLM

MLM is a method for learning the language model. 15% of input words are masked. Some of them will be visible at the end, but most of that, 12%, is replaced by token [MASK]. This way the model improves the prediction of masked words based on the surrounding context.

Next Sentence Prediction (NSP)

NSP is a similar concept to MLM. In one part of the training, inputs of the model are pairs of sentences. In 50% cases, the second sentence follows the first one in the training data. Otherwise, the second member of the pair is a random sentence from the data. The model then learns how to predict the following sentence.

Whole Model

MLM and NSP are combined and reflected in the loss function. The resulting language model is not trained to perform a specific task and this is done by stacking one or more layers at the top of this model. Then it is possible to combine training of these new layers only with allowing the gradient to flow through all model layers.

4.4.3 Implementation

The main third-party implementation to use for BERT is transformers by HuggingFace [21]. This library contains implementations of recent state-of-the-art ideas including BERT and its variations together with a pre-trained model on more than 100 languages. Pre-trained models are important because time demands on training the model from scratch even on school clusters [4]. Orginal model was trained for four days on 4 to 16 Cloud TPUs. [5] Transformers library provides prepared model architectures for tasks such as next sentence prediction, multiple-choice, classification, sequence classification or question answering. Transformers supports integration with tensorflow and keras packages. Tensorflow [22] is an interface (and a package) dedicated to machine learning. Keras [23] is a wrapper over tensorflow, which provides a shielding from too technical details. We could use Transformers method *TfBertForClassification* for obtaining a model and then train it using Keras predefined method *fit* or create own training loop using *TensorFlow.GradientTape* construct. The first method requires padding all sentences to the same length, which could produce too long sentences and have a negative impact on running time. For the second method, padding sentences only to the length of maximum in one batch (as opposed to maximum over the whole dataset) is enough and there exists library *[24]*, a wrapper upon Keras, which can do it and is used in Socneto.

With a pre-trained model and working additional training, all that is needed is to fine-tune a model. Actually, there is more BERT-like models distributed in transformers library (and Ktrain too). Some of the differs from the original paper in later improvements and some of them are just larger. Socneto uses *distilbert* model, which is small enough for a reasonable usage and offers good performance. A comparison of results with different training parameters is offered in Section 4.4.4. Here remains only to describe the training data. Although there exist many sentiment datasets, a large part of them is a binary classification (positive-negative) only. One of the three-classed datasets is Twitter US Airline Sentiment [25] containing almost 14,000 tweets about six American airlines. 70% of them were used as training data, while 30% served as validation data for verification of accuracy and detection of overfitting after every training episode.

4.4.4 Model Selection

Fine tuning is not just about creating a layer and putting training data in. There are always hyperparameters to set and the performance of models can vary a lot depending on them. In our case, the main hyperparameters to set are the learning rate, the number of episodes and the style of learning.

Learning Rate

The learning rate influences the speed of the learning by setting the size of one step through the space of the network weights. Too big learning rate can cause chaotic jumping here and there, which will never hit the optimum of the loss function. On the contrary, too small learning rate is time inefficient and can also cause getting stuck in the saddle point. The

[4] `https://gitlab.mff.cuni.cz/ksi/clusters`
[5] `https://github.com/google-research/bert`

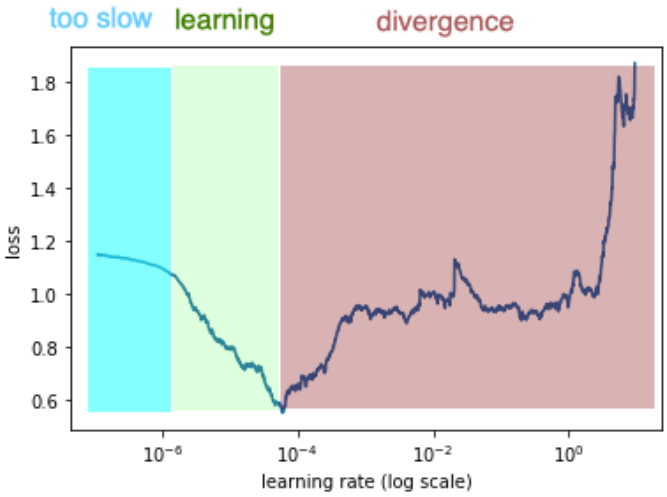


Figure 4.8: Learning rate for *distilbert* pre-trained model on airlines dataset.

Ktrain library allows usage of Keras method for searching the best learning rate. The output of this method is a plot of the learning rate versus the loss function. The best learning rate is around the point, where the loss function starts to decrease, which is around 10^{-6} in our case (see Figure 4.8).

Training Strategy

It is better to change the learning rate during training. In the beginning, when the model is bad, we want to find a better solution quickly. After some iteration, however, our model starts to be clever and it is unwanted to forget all and change the direction completely. Ktrain offers three possibilities of training - *fit_one_cycle*, *fit*, and *autofit* (see Figure 4.9). One cycle policy [26]goes from the lower bound to the upper bound in each cycle. *Autofit* method performs one cycle policy with cycle length equal to epoch for every epoch. It is also able to automatically decrease the learning rate if the loss on the validation set does not improve and stop training when it starts to diverge. The third method, *fit*, is simple - it just uses the constant learning rate all the time.

4.4.5 Experiments

In this section, different learning strategies effect is presented in a short overview. Experiments where performed for different learning rates and different strategies.

Figure 4.10 plots progression of loss function and accuracy over 10 training epochs for

Figure 4.9: A comparison of different Ktrain training methods. On the left plot, learning rate is oscillating in every cycle between minimal and maximal learning rate. Cycle length can generally be different from epoch, but typically one cycle is one epoch long. The middle image shows just constant learning rate during whole training and the rightmost figure shows possibilities of autofit method to change learning rate after n periods without loss decrease (where n is a parameter).

tree different learning rates. For the *fit* method, where constant learning rate is used, loss function starts to grow immediately after the first epoch, so further training just worsens results. One cycle policy for learning rates $4 \cdot 10^{-6}$ and $7 \cdot 10^{-6}$ is learning well for five or four epochs respectively, but best results are actually same or worse than one epoch with constant learning rate. Autofit starts to diverge, even when decrease of learning rate after 2 epochs with an increasing loss function was set for the greatest learning rate.

Figure 4.10: Loss and accuracy progress trough 10 training episodes.

There were performed other experiments using reduce of learning rate with different starting learning rate value. As starting on $8 \cdot 10^{-7}$ was slow, but with increasing learning rate over 20 episodes, starting with $6.25 \cdot 10^{-6}$ was to much and loss function grew for every episode. Neither of mentioned models was able to outperform the chosen one.

For usage in Socneto, model *fit_76* from figure 4.10, as it has the best accuracy. It is the model trained using only constant learning rate $7 \cdot 10^{-6}$ for 10 epoch.

	precision	**recall**	**f1 score**
neutral	0.87	0.82	0.84
positive	0.90	0.88	0.89
negative	0.94	0.87	0.96

Table 4.3: Results for selected model for Socneto.

As it is possible to see in table 4.4.5, on the development data, the model is very good in recognizing negative texts (as the precision form negative is quite good) and the most hardly recognized are neutral posts.

Chapter 5

Extensibility Guide

This document discusses an implementation of a custom data acquirer (Section 5.1) and a custom data analyser (Section 5.2.2). Each section introduces concrete contract that has to be followed (see Section 3.10). When each acquirer or analyser is registered, it can be used during a new job configuration (see Figure 5)

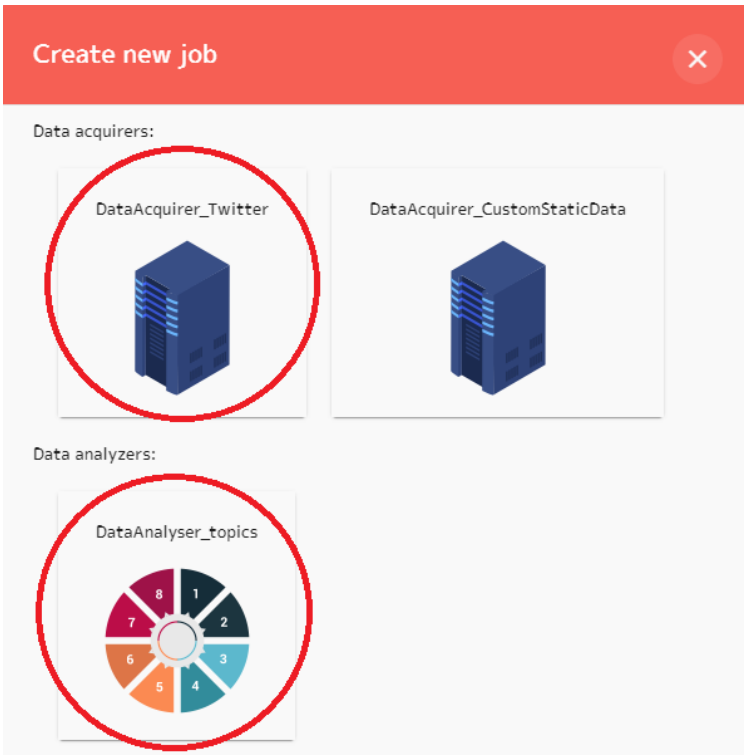


Figure 5.1: Screenshot of New Job form on Socneto front end. It shows all properly registered components

5.1 Data Acquirer - QuoteLoader

QuoteLoader loads data from a free API `http://quotes.rest/qod.json` returning random quotes.

5.1.1 Contract

Registration QuoteLoader sends registration request to the the topic `job_management.registration.request`

```
{
  "componentId": "quoteLoaderDataAcquirer",
  "componentType": "DATA_ACQUIRER",
  "updateChannelName": "job_management.job_configuration.
     quoteLoaderDataAcquirer"
  }
}
```

Note that the `inputChannelName` is not present since QuoteLoader is data acquirer and does not have an input.

Starting the job QuoteLoader starts listening on a topic `job_management.job_configuration.quoteLoaderDataAcquirer` for a job like this:

```
{
    "jobId": "11d46859-3569-4bb1-9ec0-20eeac6ce132",
    "command": Start,
    "attributes":{},
    "outputChannelNames": [
        "job_management.component_data_input.storage_db",
        "job_management.component_data_input.some_analyser"]
}
```

Once the job notification arrives, QuoteLoader starts sending GET HTTP requests to `http://quotes.rest/qod.json`. QuoteLoader must remain listening to the topic while processing the newly arrived job. The endpoint returns JSON with the quote that looks like this (truncated for clarity):

```
{
    "quote": "You make a living by what you earn; you make a
       life by what you give.",
    "author": "Winston Churchill",
    "language": "en",
    "date": "2020-02-29",
    "id": "XZiOy4u9_g4Zmt7EdyxSIgeF"
}
```

The quote is then transformed into a post message and sent to both output channels `"job_management.component_data_input.storage_db"` and `"job_management.component_data_input.some_analyser"`.
The post message looks like this:

```
{
  "id" : <new random uuid>,
  "jobId" : "11d46859-3569-4bb1-9ec0-20eeac6ce132", // from job
      notification
  "originalId" : "XZiOy4u9_g4Zmt7EdyxSIgeF", //quote id
  "text" : "You make a living by what you earn; you make a life
      by what you give.",
  "authorId" : "Winston Churchill",
  "language" : "en",
  "datetime" : "2020-02-29"
}
```

Stopping the job A stop job notification comes from the same topic as the start job notification. QuoteLoader listens on a topic `job_management.job_configuration.quoteLoaderDataAcquirer` for a job like this:

```
{
    "jobId": "11d46859-3569-4bb1-9ec0-20eeac6ce132",
    "command": Stop
}
```

Quote loader stops the job while still listening on a the topic for new jobs.

5.2 Data Analyser - Hashtags

This section describes on a real implementation, how can be a new analyser added to Socneto.

5.2.1 Contracts

Hashtag analyser is implemented to simulate the easy extensibility of analysers. This component has a simple goal: compute frequencies of hashtags in posts.

Registration The analyser sends a registration request to the the topic `job_management.registration.request`.
The registration request looks as follows:

```
{
  "componentId": "HashTags",
  "componentType": "DATA_ANALYSER",
  "inputChannelName": "job_management.component_data_input.
     HASH_TAG",
"updateChannelName": "job_management.job_configuration.HASH_TAG
    "
  }
}
```

Starting the Job The Hashtag analyser is stateless, so it does not need update messages. All posts are always analysed and sent to storage on a topic:
`job_management.component_data_analyzed_input.storage_db`

Analysing posts When the post message is received, the module creates a frequency map of found hashtags and sends the analysis message to the storage

Example post message:

```
{
  "id" :"9e36b903-7f1b-43dd-b4f8-f64a8f95245d",
  "jobId" : "11d46859-3569-4bb1-9ec0-20eeac6ce132",
  "originalId" : "XZiOy888_g4Zmt7EdyxSazc",
  "text" : "#czech_castle #top09 Kalousek na hrad.",
  "authorId" : "Andrej",
  "language" : "cz",
  "datetime" : "2020-02-29"
}
```

Example analysis:

```
{
  "postId": "9e36b903-7f1b-43dd-b4f8-f64a8f95245d",
  "jobId": "11d46859-3569-4bb1-9ec0-20eeac6ce132",
  "componentId": HashTags,
  "results": {
    "valueName": {
        "numberValue": null
        "textValue": null
        "numberListValue": null
        "textListValue": null
        "numberMapValue": {
                "czech_castle": 1,
                "top09": 1
        },
        "textMapValue": null
    }
  }
}
```

Stopping the Job Because this analyser is stateless, it does not need to work with the stop job procedure.

5.2.2 Java Skeleton

The Hashtag analyser is implemented on top of our prepared Java Spring skeleton for an easy analyser implementation. This implementation is stored in *generic_analyser* folder. There is a list of steps, that must be followed to integrate a new analyser into the platform from this skeleton.

- A new analyser must have configured these properties *in application.properties*:
  - Unique component id
    - property: `component.config.componentId`

  - Topic for new posts
    - property: `component.config.topicInput`
    - conventionally with prefix: `job_management.component_data_input`
  - Topic for job updates
    - property: `component.config.topicUpdate`
    - conventionally with prefix: `job_management.job_configuration`
  - Spring profile
    - property: `spring.profiles.active`
  - Output format (described in Section 3.7.5)

- Orchestration is prepared and the only one thing is to implement the `cz.cuni.mff.socneto.storage.ana` interface. There are two methods, that must be overridden:
  - `Map<String, String> getFormat()`
    - an output format for registration message (the structure is described in Section 3.7.5)
  - `<MapString, AnalysisResult> analyze(String text)`
    - input: a text to analyse
    - output: a result in the same format as the format returned in the method *getFormat*

This approach should help an expert users of Socneto to implement a new analyser with reusing common and required parts of all analysers.

Chapter 6

Testing

Socneto framework is tested on the following four levels, ordered from the lowest level to the highest:

- Unit - Automatic tests of methods or classes
- Integration - Automatic tests of integration of multiple classes
- Component - Automatic tests of the whole component
- System - Manual tests of the whole application

6.1 Requirements

- .Net Core 3.1 SDK
- .Net Core 2.2 SDK (for backend)
- Python 3.7.1
- Powershell Core 6.2.4

6.2 Unit Tests

Job Management Service

Test location: `<socneto-root-dir-path>/job-management/Tests`
Run tests command:

```
dotnet test <socneto-root-dir-path>/job-management/Tests
```

Backend

Test location: `<socneto-root-dir-path>/backend/Tests`
Run tests command:

```
dotnet test <socneto-root-dir-path>/backend/Tests
```

Data Acquirers

All data acquirers shares the same code base therefore are all tested with the same unit tests.

Test location: `<socneto-root-dir-path>/acquisition/DataAcquirer/Tests`
Run tests command:

```
dotnet test <socneto-root-dir-path>/acquisition/DataAcquirer/
    ↪ Tests
```

Storage

Test location: `<path-to-module>/src/test/java`
Run tests command:

```
mvn test -DskipTests=false
```

6.3 Integration Tests

Job Management Service

Test location: `<socneto-root-dir-path>/job-management/Tests.Integration`
Run tests command:

```
dotnet test <socneto-root-dir-path>/job-management/Tests.
    ↪ Integration
```

Data Acquirers

All data acquirers shares the same code base therefore are all tested with the same unit tests.

Test location: `<socneto-root-dir-path>/acquisition/DataAcquirer/Tests.Integration`
Run tests command:

```
dotnet test <socneto-root-dir-path>/acquisition/DataAcquirer/
    ↪ Tests.Integration
```

6.4 Component Tests

Component tests can be run separately, as described below, or at once with the following command:

```
<socneto-root-dir-path>/tests/component_tests/run_test_da.ps1
```

Job Management Service

Test location: `<socneto-root-dir-path>/tests/component_tests/code/test_da.ps1`
Run tests Powershell script:

```
<socneto-root-dir-path>/tests/component\_tests/run\_test\_da.
    ↪ ps1
```

Data Acquirers

Test location: `<socneto-root-dir-path>/tests/component_tests/code/test_da.ps1`
Run tests Powershell script:

```
<socneto-root-dir-path>/tests/component\_tests/run\_test\_da.
    ↪ ps1
```

Data Analysers

Test location: `<socneto-root-dir-path>/tests/component_tests/code/test_an.ps1`
Run tests Powershell script:

```
<socneto-root-dir-path>/tests/component\_tests/run\_test\_an.
    ↪ ps1
```

Backend

Test location: `<socneto-root-dir-path>/tests/component_tests/code/test_be.ps1`
Run tests Powershell script:

```
<socneto-root-dir-path>/tests/component\_tests/run\_test\_be.
    ↪ ps1
```

Chapter 7

Discussion

7.1 Architecture

Socneto was designed to be extensible, therefore a monolithic application was not a considered feasible. The domain of the application was split into dedicated services allowing them to run independently coordinated by one dedicated Job Management Service.

Socneto consists of multiple data acquirers streaming data to multiple data producers and eventually into the storage. It is unsuitable to use synchronous communication for streaming therefore Socneto internal communication is based upon asynchronous message broker Kafka. At the beginning, this decision was slowing us down, because it could solve more complex tasks than we needed. But in the end, we achieved our goal to have a scalable, distributed and extensible architecture.

The only change in the architecture during implementation was to remove the NoSQL database from the storage module, because it was not useful for any use-case and Elasticsearch is sufficient for the storing acquired data. If we had more time, there could be more research about choosing the right messaging broker, for example, we could use a more lightweight solution. Otherwise, this architecture satisfies a possible production usage. The only unimplemented part is authentication and authorization between FE, Backend, JSM and Storage modules.

7.2 Infrastructure

Socneto was designed to run on premises. This gave us absolute freedom of the architectural and hosting aspects of the development. The university provided us with multiple Virtual Machines which were used to test and deploy Socneto. Given the fact that Socneto is a data processing framework, it could not be hosted on a single PC.

The alternative to this decision was to run Socneto in some cloud platform. The disadvantages are:

- The lack of professional experience with the application scope of the size of Socneto.
- The time required to get accustomed to cloud development.

- Unsuitability of the cloud for prototyping
- Limited options of free subscriptions and price of premium ones.

These disadvantages outweighted the advantages a cloud would have. The cloud is a infrastructure-as-a-service. All production grade features such as scaling, load balancing, security are supported out-of-the-box. The decision in favour of on premises deployment was based upon the fact that in this stage Socneto is not a production grade software making the use of the advantages limited.

Socneto design allows it to run in Docker - each component is hosted in a separated docker container. Docker (docker-compose) was chosen, because it provides fast deployment with easy configuration without requiring all installed dependencies. This approach can be extended with automatic testing and deployment pipeline, which is used in real-world projects.

Important benefit is that each service can be implemented using different technology. In Socneto, each component is written in a language most suitable for the purpose of the given component. In this case, data analysers are based on python since it is the most popular language for data science related tasks. Java was used for data storing related service since it integrates best with Java based Elastic Search. Due to that there were no conflicts in the beginning of the project related to technological uniformity.

7.3 Social Networks

After the Cambridge Analytics issue [1], most social network providers limited their API usage which influenced the decision of which social networks to support. Socneto supports Twitter and Reddit. Twitter was chosen, because it offers easily accessible data with many customers for free and Reddit because it contains longer textual data with comments. Facebook was not used, because it restricted API was not possible to utilize for free.

Lately, social networks started to shift toward audiovisual content rather than textual. Supporting such network will introduce a whole new level of complexity and would impact every aspect of Socneto architecture. For that reason Socneto focuses only on textual data leaving space for future improvement.

7.4 Data Analysis

As a part of this project, three text analyses where implemented - named entity recognition, sentiment analysis and latent Dirichlet analysis. We mainly used third-party libraries with existing pre-trained models. For sentiment analysis, fine-tuning of the model was performed together with the best model selection. We definitely do not have state-of-the-art solutions, but our analyses have a reasonable performance for usage and can be easily improved with more training (which include more time and also more training data).

[1] `https://en.wikipedia.org/wiki/Cambridge_Analytica`

7.5 Market Potential

Development of Socneto has two directions. The first one is to open it to public and let each potential customer to build their personalized frameworks around it and monetize the support as many other projects already have done (most notable are Hadoop Ecosystem[2] and Project BlackLight[3])

The other is to extend it ourselves the way as our competitors did (as discussed in Section 2.3). Before we may sell Socneto to the first possible customer, we should extend analysers with more functionality (geographical data, image recognition) and also we have to cover other data sources (Instagram, Pinterest). The platform needs to improve security and all components need polishing. But the current state is prepared for further improvements.

7.6 Summary

Socento successfully delivered the specified functionality. Starting with acquisition of data from various sources and its following analysis with complex linguistic models performing sentiment analysis and topic modelling. The application fulfills requirements made to data storage and visualization. Extensibility was proved by implementing custom dataset acquirer and hasthag analyser on top of what was originally planned.

To sum up the positive sides, the application is a well-designed data processing framework that employs principles such as service oriented architecture, asynchronous messaging and NoSQL databases. Socneto uses modern platform for deployment making it easier for users' better integration.

The weaker side of Socneto is the lack of security and proper user management expected from a production-grade software, but not essential for an academic project. Testing could be improved to give us a confidence that Socneto is ready for being used by customers.

7.7 Future Work

Socneto lack of production-grade features is currently the most pressing issue that should be solved before Socneto can be released. The problem with security and user management would be solved with migration to the cloud which would require some adjustments. It would also make it easier to implement continuous integration process.

In terms of the functionality, data acquirers could utilize more data offered by social networks such as geo codes, post statistics and comments. It could also start processing non textual data and implement analyses that would understand it. Socneto currently supports only primitive data flow with three steps. Some data acquirer gives data to some data analyser which then gives analyses to storage. This can be expanded to support custom data flow chaining compatible analysis. For example, the translation service is coupled with Twitter and Reddit data acquirers. It could be turned into a stand-alone component that would be part of the pipeline.

[2] `https://hadoopecosystemtable.github.io/`
[3] `http://projectblacklight.org/`

Chapter 8

Conclusion

This document presents an extensible platform for downloading and analysing data from social network. It established the field of interest, introduced competitors and the timeline of the project. It delivers comprehensive architectural and code documentation with an extensibility guide and discusses achieved results. In conclusion, Socneto was a successful project that served its developers as a ground for proving that university prepared them for their careers.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Massachusetts Institute of Technology, 2016.

[2] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, Inc., 2009.

[3] "Pdt 3.5 main page." [Online]. Available: https://ufal.mff.cuni.cz/pdt3.5

[4] D. P. Company, *Final Evaluation Of The Results Of Eurotra: A Specific Programme Concerning The Preparation Of The Development Of An Operational Eurotra System For Machine Translation*. Diane Publishing Company, 1995. [Online]. Available: https://books.google.cz/books?id=_TVpIqThJP0C

[5] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, September 1990. [Online]. Available: https://ideas.repec.org/a/bla/jamest/v41y1990i6p391-407.html

[6] D. M. Blei, A. Y. Ng, M. I. Jordan, and J. Lafferty, "Latent dirichlet allocation." *Journal of Machine Learning Research*, vol. 3, no. 4/5, pp. 993 – 1022, 2003. [Online]. Available: https://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds

[7] "Details/paper used for recent ner implementation." [Online]. Available: https://github.com/explosion/spaCy/issues/2107

[8] E. Strubell, P. Verga, D. Belanger, and A. McCallum, "Fast and accurate sequence labeling with iterated dilated convolutions," *CoRR*, vol. abs/1702.02098, 2017. [Online]. Available: http://arxiv.org/abs/1702.02098

[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[10] V. Sankar, R. P. Patil, and D. S. Mahajan, "Word2vec using character n-grams."

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[12] H. C. Daume and D. Marcu, *Practical structured learning techniques for natural language processing*. Citeseer, 2006.

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

[14] "Named entity recognition for twitter." [Online]. Available: http://blog.thehumangeo.com/twitter-ner.html

[15] "Workshop on noisy user-generated text (w-nut)." [Online]. Available: https://noisy-text.github.io/2017/index.html#

[16] "Annotation in spacy." [Online]. Available: https://spacy.io/api/annotation

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

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

[19] 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

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need. arxiv 2017," *arXiv preprint arXiv:1706.03762*, 2017.

[21] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[23] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[24] "Ktrain." [Online]. Available: https://github.com/amaiya/ktrain

[25] "Twitter us airline sentiment." [Online]. Available: https://www.kaggle.com/crowdflower/twitter-airline-sentiment

[26] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

Appendix A

Appendix

A.1 Entities

All entities are in JSON format, data types of values are inside < and > symbols.

A.1.1 Registration Message

```
{
  "componentId": <string>,
  "componentType": <DATA_ACQUIRER or DATA_ANALYSER>,
  "updateChannelName": <string>,
  "inputChannelName": <string>,
  "attributes": <json object>
  }
}
```

A.1.2 Job Configuration

```
{
    "selectedDataAcquirers":[<string>],
    "selectedDataAnalysers":[<string>],
    "topicQuery": <string>,
    "jobName": <string>,
    "attributes":{
        "acquirer_1":{
            "parameter1": <string>
        }
        ...
    }
}
```

A.1.3 Update Message

```
{
    "jobId": <uuid>,
    "command": <'Start' or 'Stop'>,
    "attributes":{
        "attribute_1": <string>,
        ...
    },
    "outputChannelNames": [<string>]
}
```

A.1.4 Post Message

```
{
  "id" : <uuid>,
  "jobId" : <uuid>,
  "originalId" : <long>,
  "text" : <string>,
  "originalText" : <string>,
  "authorId" : <string>,
  "language" : <string>,
  "datetime" : <datetime>
}
```

A.1.5 Analysis Message

```
{
  "postId": <uuid>,
  "jobId": <uuid>,
  "componentId": <string>,
  "results": {
    "valueName": {
        "numberValue": <double>,
        "textValue": <string>,
        "numberListValue": [<double>],
        "textListValue": [<string>],
        "numberMapValue": {<string,number>},
        "textMapValue": {<string,string>}
    }
  }
}
```

A.1.6 Log Message

eventType values: FATAL, ERROR, WARN, INFO, METRIC

```
{
  "componentId": <string>,
  "eventType": <string>,
  "eventName": <string>,
  "message": <string>,
  "timestamp": <datetime>,
  "attributes": {<any object>}
}
```

A.2 Detailed specification as approved

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).

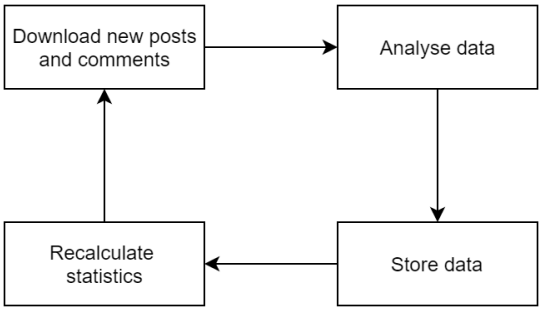


Figure 1: Endless pipeline

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]

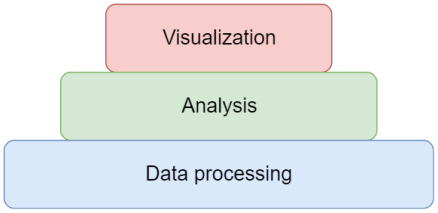


Figure 2: Conceptual separation of application responsibilities: to develop the data processing platform, to analyze the data, and to present them to the user

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).

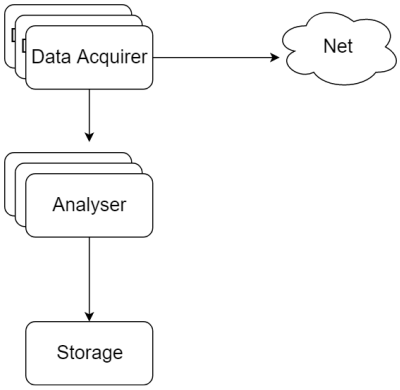


Figure 3: A simplified version of the pipeline processing data

[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.

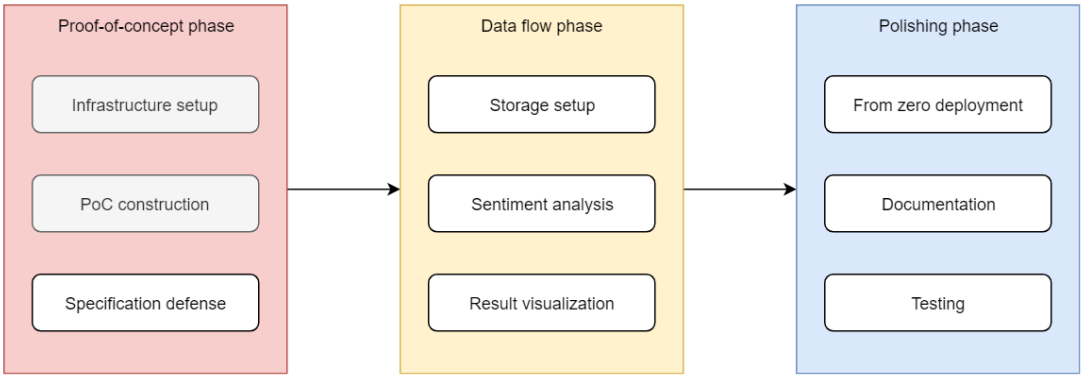


Figure 4: Project development phases

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.

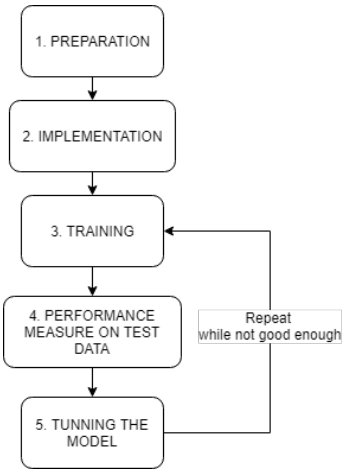


Figure 5: Machine learning workflow

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.

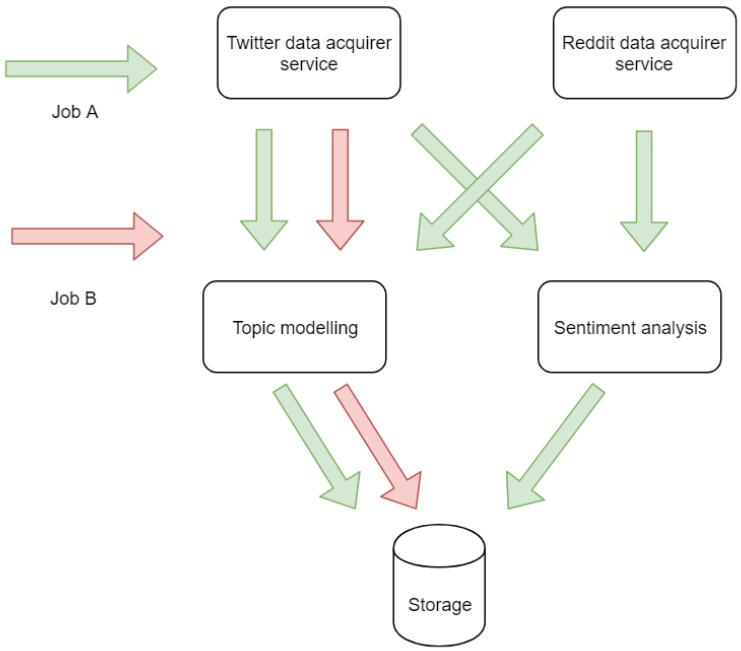


Figure 6: Two jobs running in parallel

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]

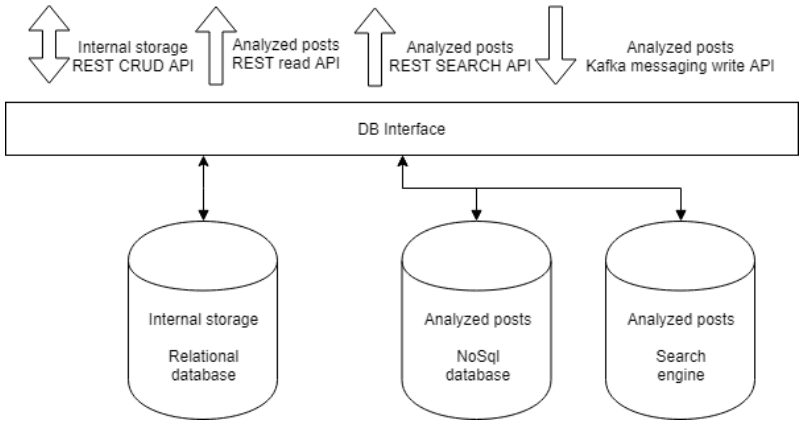


Figure 7: Storage architecture

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`

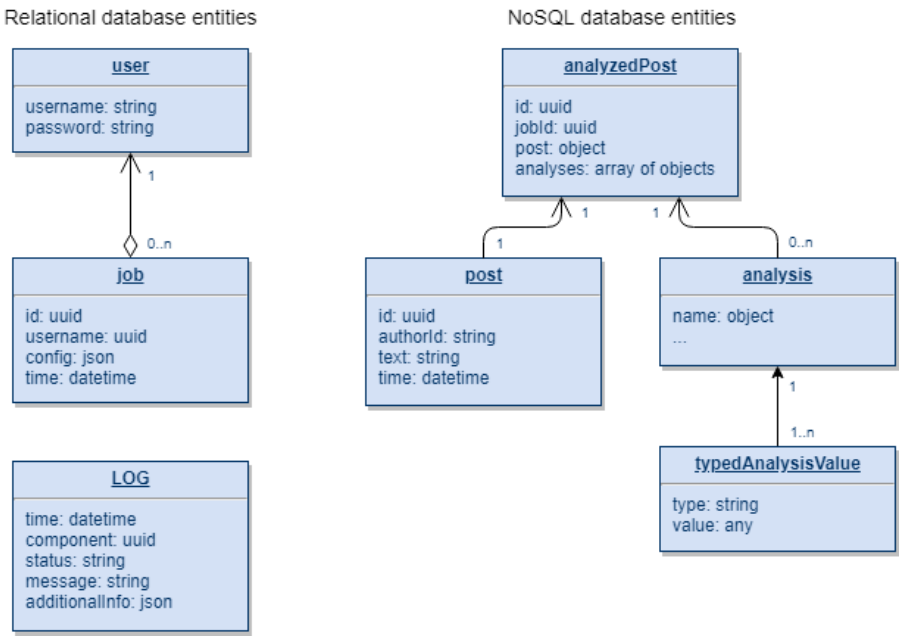


Figure 8: Database schema

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.

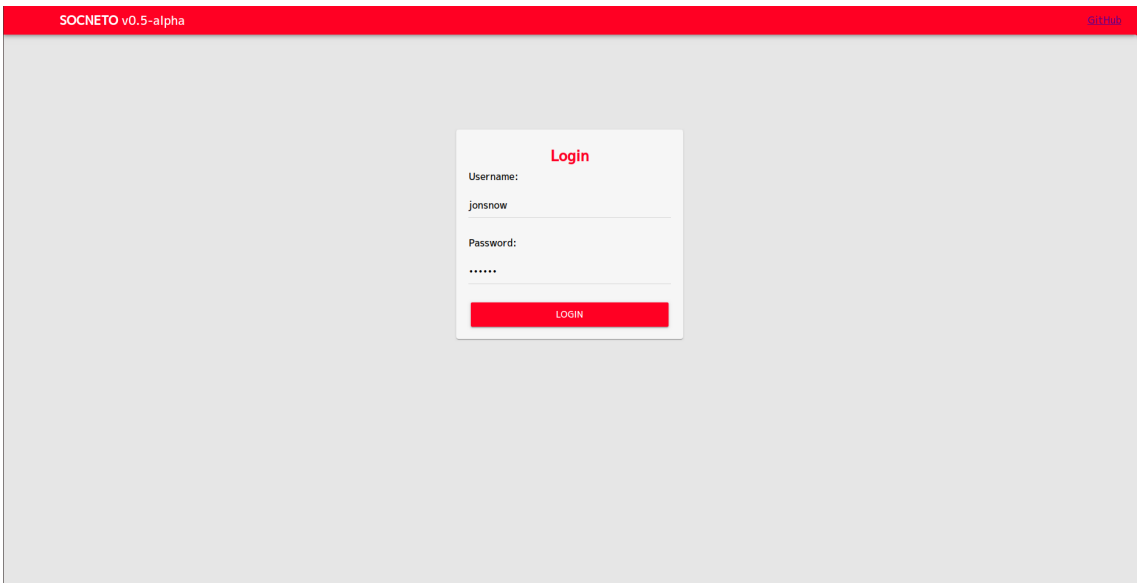


Figure 9: Login page

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

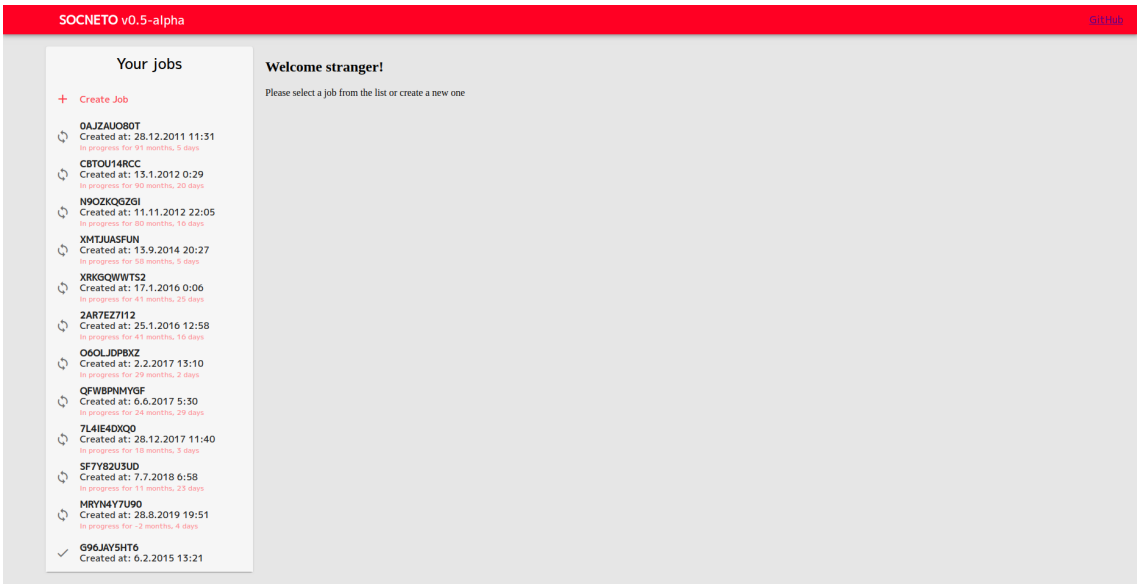


Figure 10: Dashboard

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.

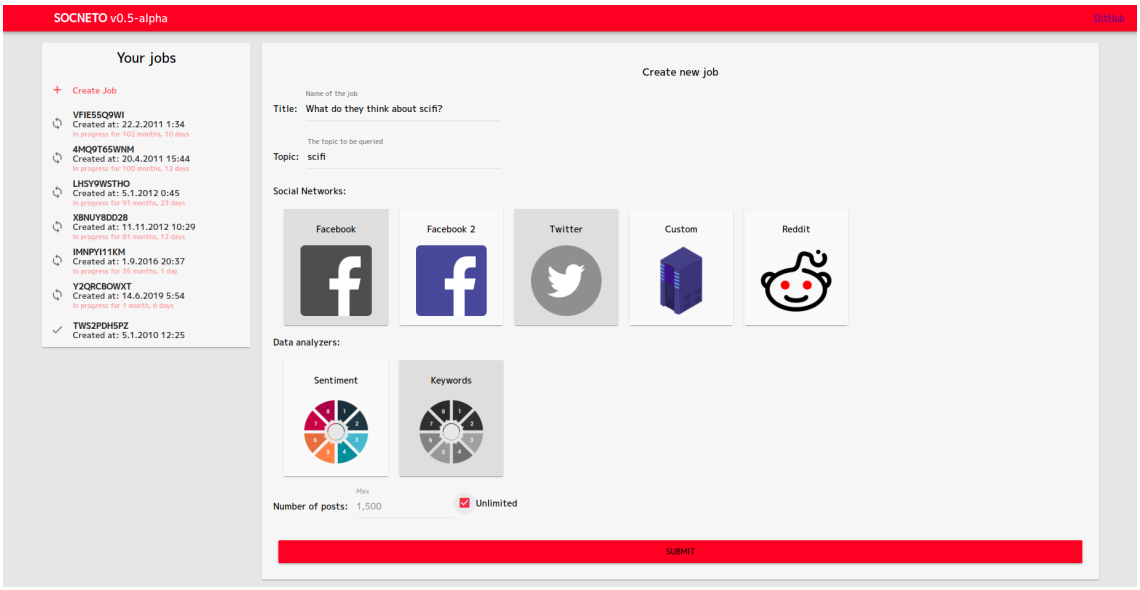


Figure 11: New job

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