User Documentation

Socneto

February 2020

1 Installation

This section describes the installation process of the platform and its requirements. To make the installation process as simple as possible, the project can be installed using *docker* and *docker-compose* tools which install all the required technologies (external and internal).

1.1 Requirements

In order to install the platform using *docker-compose* tool, the following packages are required:

- **docker**, version 18.0.9.7,
- **docker-compose**.

The whole platform requires a minimum of 16GB of memory, but optimal is 20GB. Because many docker images need to be installed, it requires at least 20GB space on a disk.

1.2 Installation Process

The *docker-compose* configuration is split into two files: *docker-compose.yml* and *docker-compose-local.yml*. The command to build, install and run the platform is the following:

```
docker-compose -f "docker-compose.yml" \
-f "docker-compose-local.yml" up
```

The command is expected to be run from the root directory of the repository. After all the docker images are started, the frontend can be found on address *localhost:8080*.

1.3 Docker Images

In our project, we use multiple external technologies. The following list contains all docker images which install these:

- **Apache Kafka** and its dependency **Apache Zookeeper**: Kafka can be found on ports 9094 and 9092, Zookeeper can be found on port 2181,
- **PostgreSQL**, which can be found on port 5432,

- **ElasticSearch** version *7.4.2*, running on ports 9200 and 9300,
- **Logstash** on ports 9600, 9999 and 9998,
- **Kibana** on port 5601.

The following list contains images that were developed within Socneto:

- **Storage**: the dockerfile can be found on path `/storage/dockerfile`. It uses **Maven** image version *3.6.0* and **OpenJDK** image version *11* to build itself and run. After installing, it will run on port 8888,
- **Job Management Service**: the dockerfile can be found on path `/job-management/Dockerfile`. It uses images **.NET Core SDK** version *3.1* and **ASP.Net**. After installing, it can be found running on port 6009,
- **Backend**: the dockerfile can be found on path `/backend/Dockerfile`. It uses images **.NET Core SDK** version *2.2* and **ASP.Net**. After installing, it can be found running on port 6010,
- **Frontend**: the dockerfile can be found on path `/frontend/Dockerfile`. It uses **Dart** version *2.7.1* image to build itself and run. After installing, it can be found on port 8080,
- **Acquirers: Twitter, Reddit and Custom Static Data**: their docker files can be found in directory `/acquisition/DataAcquirer/` (`Dockerfile.twitter`, `Dockerfile.reddit` and `Dockerfile.customstatic`). They all use images **.NET Core SDK** version *3.1* and **ASP.Net**,
- **Analysers: Sentiment and Topics**: their dockerfiles can be found on paths `/analysers/sentiment_an` and `/analysers/topic_modeling/dockerfile`. They both use **Python** image to install and run. Sentiment analyzer uses version *3.7.5* while Topic analyzer uses *3.8*.

1.4 Volumes

We bind two volumes to persist data between docker builds. Directory **data01** is used to store all data from ElasticSearch, and directory **data02** is used to store all data from the PostgreSQL database.

2 User Documentation

This section provides a guide on how to use Socneto platform. The user communicates with the platform via our frontend application. It is a web-based application, thus the user can use a browser to use it. If deployed using the installation process from Section 1, the frontend can be found at address *http://localhost:8080*. The recommended browser is Google Chrome.

2.1 Login

The index page (*http://localhost:8080*) contains login screen. There the user needs to input correct credentials to log in and continue using the frontend. The default credentials are *admin:admin*.

Additionally, the user can login only if all the core components of the platform are running. If any of them are not running, the user is informed by a warning at the bottom of login component and the login button is disabled. This can be seen in Figure 1.

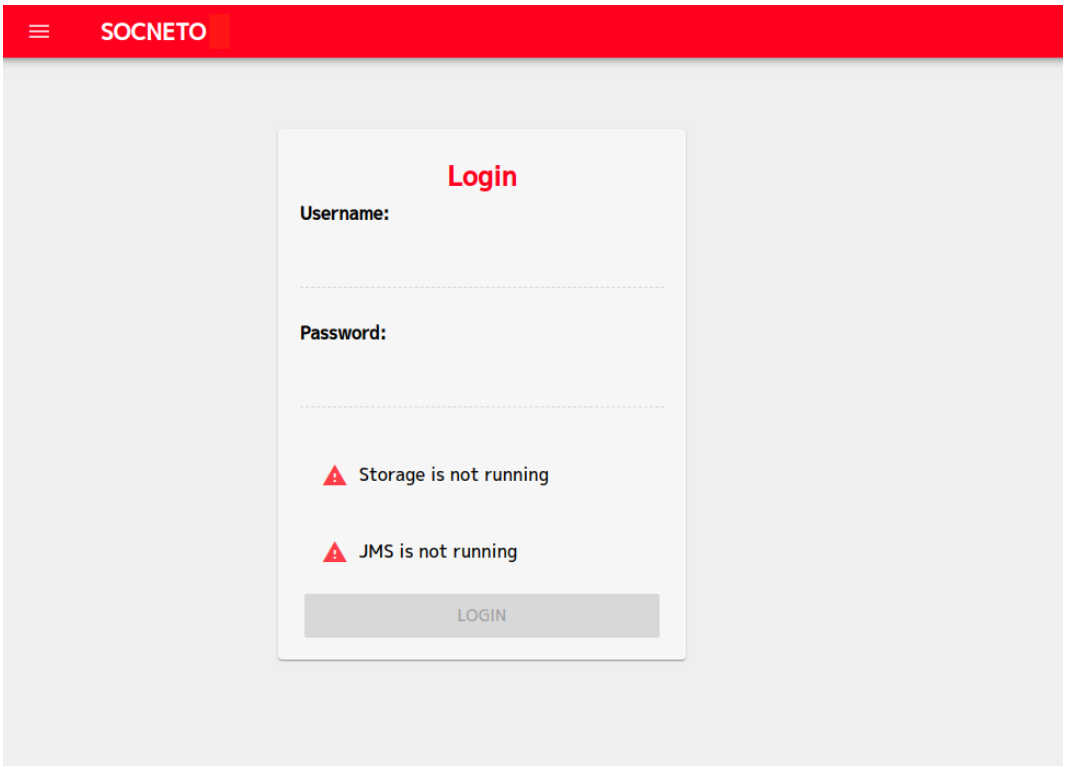

Figure 1: Login screen when storage and job management service are not running

2.2 Quick Guide

After logging in, the user is redirected to page `/#/workspace/<username>`, where a quick guide is displayed. It is a minimal, image-based, wizard-like guide to outline the usage of the application to any user. Snapshot of the guide can be seen in Figure 2.

The user can navigate to the quick guide, by using *Quick guide* button in the top left corner of the page. The button is displayed on every page where the user is logged in.

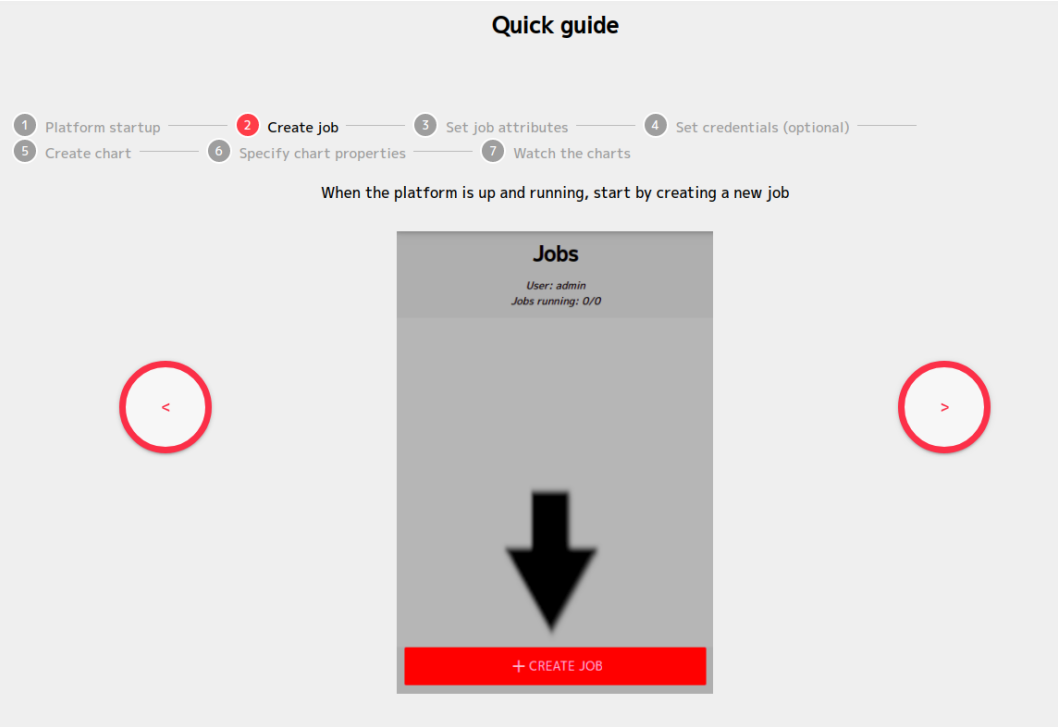

Figure 2: Quick guide

2.3 Platform Status

To find out which core components of the platform are running, the user can click on *hamburger menu icon* button on the left side of the page header. The button is available on all pages. It will unroll a small widget displaying the components and their status using a small colored circle next to their name. The color of the circle indicates the component's status. The colors can be:

- **Green** - component is running,
- **Red** - component is not running,
- **Gray** - component's status is unknown (this happens when the backend component is not running).

The unrolled widget can be seen in Figure 3. Clicking anywhere outside the widget will close it.

2.4 Jobs

One of the core activities a user can do in the frontend is creating jobs, which download data from social networks and analyze them consequently.

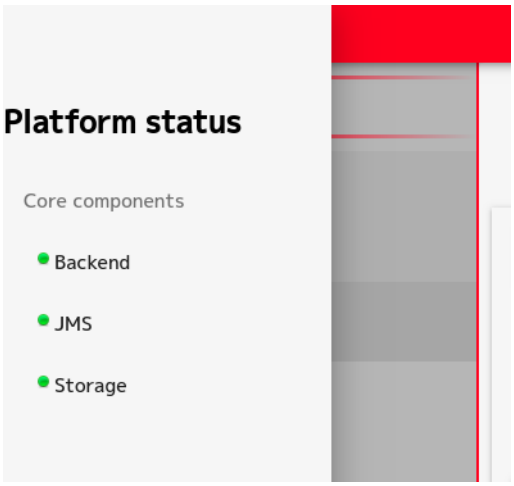

Figure 3: Platform status widget

2.4.1 Jobs List

After logging in, on page `/#/workspace/<username>`, the user can see the list of all jobs he has created. It is displayed on the left side of the page. It also contains selected basic information about them, including title and how long they have been running. Each running job can be stopped by a stop button to the left of the job's title. When a job is stopped, it will not download any new data or analyze any new posts.

A sample jobs list is illustrated in Figure 4. It contains two running jobs called *Ozzy Osbourne* and *Heavy Metal*. We can see that the first one has been running for 1 day and 2 minutes, while the other for 1 day and 22 hours. We can also see the square-shaped *stop button* next to each job. Above the list there is also displayed which user is logged in, and how many jobs he has created and how many of them are running.

2.4.2 Job Creation

To create a job, there is a button at the bottom part of the jobs list *Create Job*, which can also be seen in Figure 4. After clicking the button, a job creation dialog is shown. Similarly to the login screen, the user cannot create jobs, if any of the core components is not running. This is also displayed to the user the same way as on the login screen.

The job creation dialogue is displayed in Figure 5. When creating a job, the user needs to specify a few parameters of the job in the dialogue:

- Title - specifies the job's title. This parameter is only for better orientation in the user's jobs list,
- Topic - the topic to be queried. The topic does not need to be only one word. If multiple words are specified, the acquirers will act as if logical *or* was between them, and start downloading posts, where at least one of the words is contained. To query multiple words as one topic, they need to be inside *double quotes*, for example, *"Star*

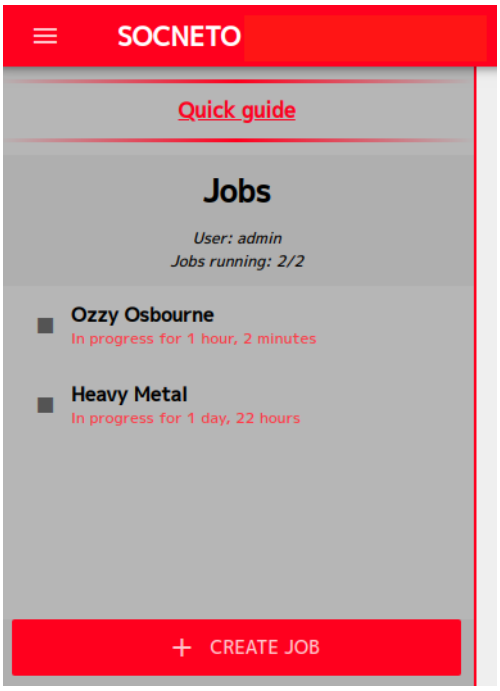

Figure 4: User's jobs list

Wars". The topic query also supports a few logical operators, such as `;` (`or`), `AND` and `NOT`, for example `humans AND elves NOT dwarves; hobbits; wizards`. This formula is then parsed into *DNF* format, so it would look like `humans AND elves AND NOT dwarves AND NOT hobbits AND NOT wizards` (note the `NOT` operator being applied to the whole clause, not only the next variable),

- Data acquirers - select which acquirers should be used in this job. Selected acquirers will start acquiring posts for the given topic when the job is created,
- Data analyzers - select which analyzers should be used in this job. Selected analyzers will analyze all posts acquired by selected acquirers.

All of the above-listed parameters are required, and the job can be submitted if all of them are specified (are not empty). This can be done by clicking on the *Submit* button in the dialogue. If any of these parameters is not specified, the button is disabled. This can be also seen in Figure 5.

Each of the selected acquirers can then be optionally parameterized by acquirer-specific parameters. These parameters are specified in the second step of the job creation dialogue. To get to this step, the user needs to click on the *Custom credentials (optional)* button. If not all of the required parameters are specified, the button is disabled, as can be seen in Figure 6.

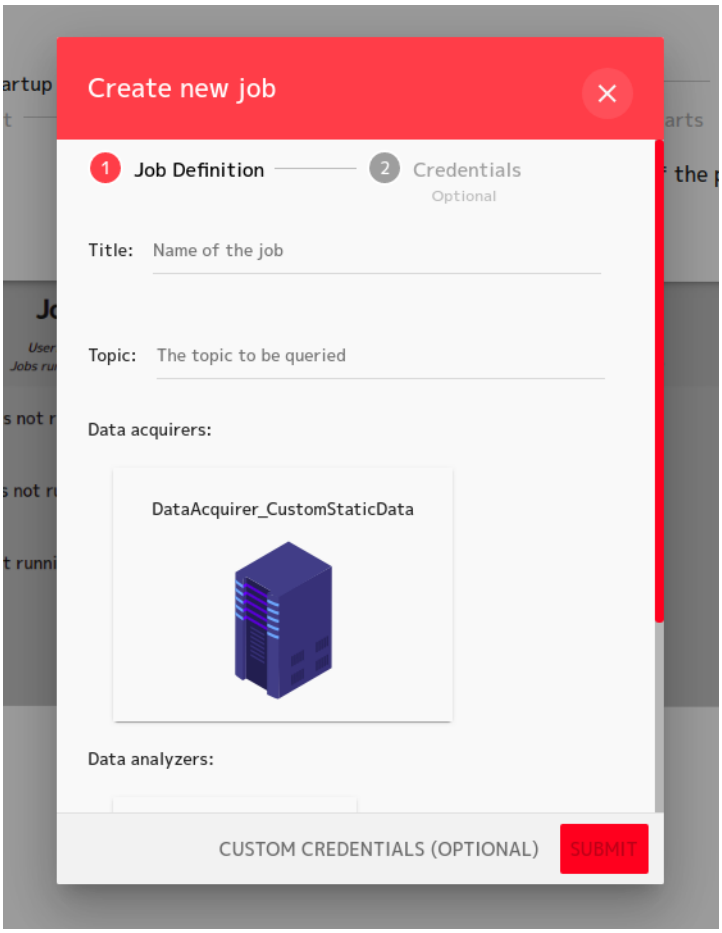

Figure 5: Dialogue displayed when creating a job

In this step, each selected acquirer has its own tab, and can be specified its own set of parameters. This can be seen in Figure 6. In this figure, we can see three defined parameters: `bucketName`, `objectName` and `mappingName`; and their values: `example-data`, `tweets.csv` and `tweets.csv.mapping`. Which parameters the default acquirers use is described in Section 2.4.3.

This step of job creation contains also *Translate* toggle. If it is turned on, the acquired posts will be translated into English by the acquirer. The toggle can be seen in Figure 6.

2.4.3 Acquirer's parameters

Socneto comes with a few acquirers already available. These acquirers use some required or optional parameters. The following list describes these parameters:

- **DataAcquirer_Twitter** acquirer downloads tweets from twittter.com using its API.

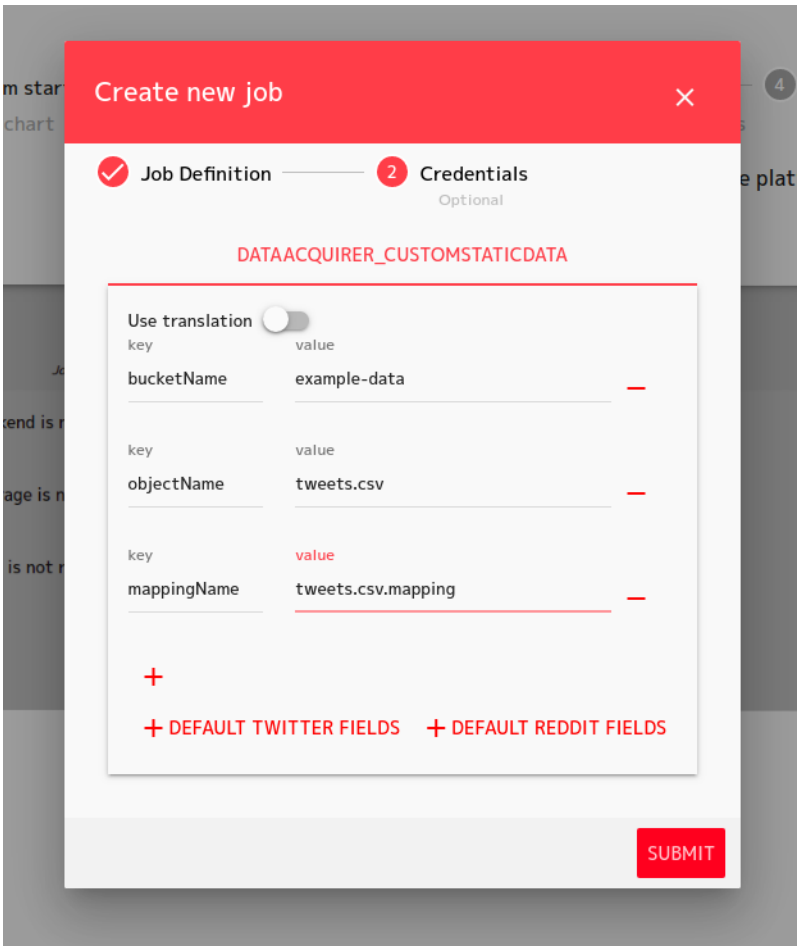

Figure 6: Optional custom credentials for acquirers

This API requires a user to be logged in. Thus, credentials of a user can be specified for this acquirer using the following parameters: `ApiKey`, `ApiSecretKey`, `AccessToken` and `AccessTokenSecret`. If none of them are specified, default credentials for a user with *Standard API access* are used,

- **DataAcquirer_Reddit** acquirer downloads posts from reddit.com using its API. This API requires a user to be logged in. Thus, credentials of a user can be specified for this acquirer using the following parameters: `RedditAppId`, `RedditAppSecret` and `RedditRefreshToken`. If none of them are specified, default credentials for a user with free access are used,
- **DataAcquirer_CustomStaticData** acquirer uses a MinIO server to download already acquired data. The acquirer requires parameters, which specify where the

dataset files are located on the server. These parameters are `bucketName`, `objectName` and `mappingName`. More on this can be read in Section 3.1.

2.5 Job Data

After selecting a job from jobs list, the user is redirected to page `/#/workspace/<username>/job/<jobId>`. It displays job's data. It is divided into three tabs:

- **Detail** - displays some basic information and analyses for the job (more in Section 2.6),
- **Analyses** - displays user defined charts of analyses of job's acquired posts (more in Section 2.7),
- **Posts** - displays list of all acquired posts by the job (more in Section 2.9.

These tabs can be seen on the top of Figure 7. This page also displays jobs list on the left side.

2.6 Job Detail

The first tab in job's data page (Section 2.5) displays basic information about the job, such as: *topic query* (as specified during job creation), *status* (running or stopped), *posts count* (number of posts acquired by acquirers for the current job), *date* when the job was started and its *running time*. These are displayed on the very top of the page, as can be seen in Figure 7.

Following this information, some basic analyses are displayed. These analyses are:

- **Posts frequency** - line chart whose x-axis is time (split into hour long interval) and y-axis is number of posts acquired for given hour (can be seen in Figure 7),
- **Language frequency** - table displaying languages (in *ISO 3166 Alpha-2 format*) and how many acquired posts were written in given language,
- **Author frequency** - table displaying authors and how many acquired posts were posted by given author.

2.7 Charts

The second tab in the job's data page (Section 2.5) displays user-defined charts of downloaded posts' analyses. The user can create any number of charts for any *running or stopped* job. Which data is displayed in the chart depends entirely on the user's specification of the chart. A user-defined chart called *Topics - pie* can be seen in Figure 9.

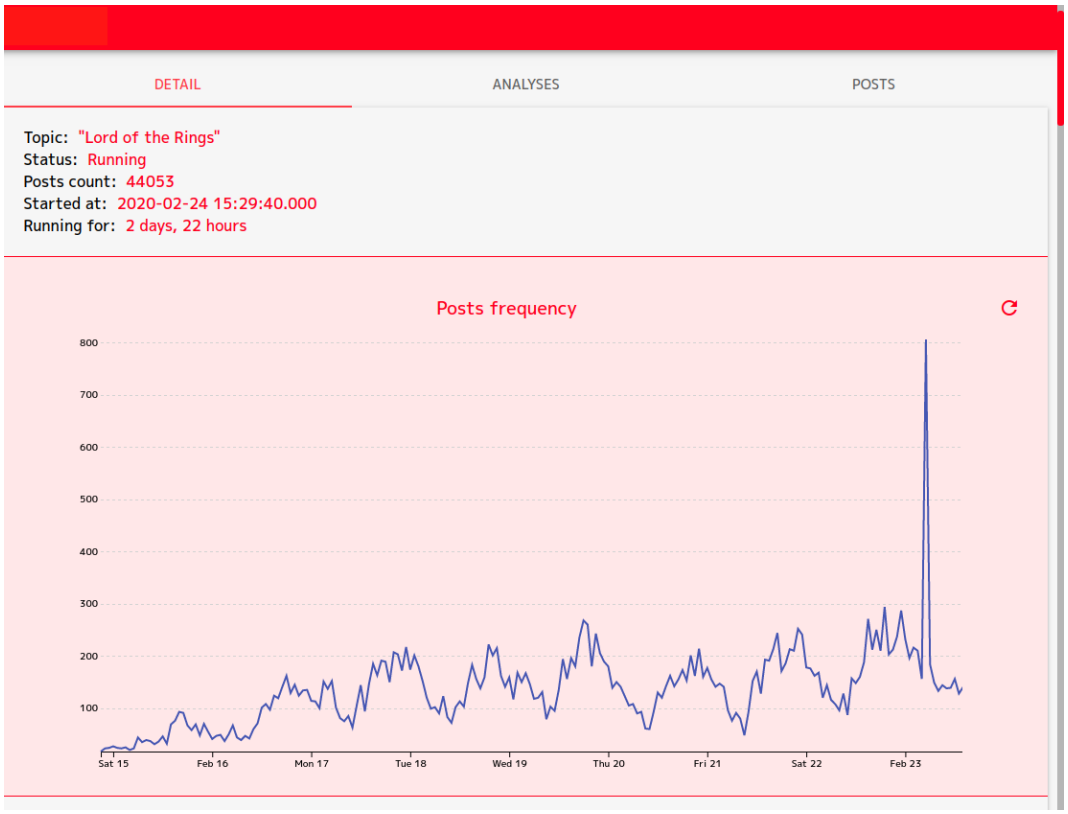

Figure 7: Example of a job detail tab

2.7.1 Chart Creation

At the bottom of the charts list is a large '+' button. Clicking it will show a chart creation dialogue. The dialogue can be seen in Figure 8. The chart creation process differs slightly for different types of charts. The user is required to specify the following parameters:

- **Chart title** - used only for user's better orientation in the list of charts,
- **Chart type** - select from five different types of chart: *line chart*, *pie chart*, *bar chart*, *table*, *word cloud* and *scatter chart*,
- **Analyses' properties** - specify which properties from analyzers should be displayed in the charts:
  - **Line chart** - the x-axis of the line chart can be either a numeric property from posts' analyses or posts' posted time. This can be specified by checking or unchecking *Use post time on X axis* checkbox. If it is unchecked, the *X-Axis* analysis property must be specified. Either way, at least *Line 1* analysis prop-

erty must be specified, and consequently, any number of following lines can be specified,

- **Pie chart, Bar chart, Word cloud and Table** - only one analysis property can be specified for these types of charts. The values of this analysis property are then aggregated over all the acquired posts and shown in the charts,
- **Scatter chart** - exactly two analysis properties must be specified for this type of chart. The first one is used for the x-axis, and the second one for the y-axis. The scatter chart is then created according to the values of theses analysis properties appearing in the acquired posts.

. Not all analyzers' output fields can be displayed by all charts. Line and scatter charts are able to display only numeric field. Pie, bar and table charts can display only map and list fields. Word cloud can display only map and list of text fields. More on analyzers' output fields can be found in *Development documentation* document, in Section *3.7.5 Communication*.

This dialogue can be seen in Figure 8, where pie chart type is selected, and the selected property to be aggregated in the chart is *topics* from analyzer *DataAnalyser_topics*.

2.8 Charts List

The charts in the charts list are not updated automatically as new analyses arrive in the platform. However, they can be refreshed by clicking on the *bent arrow* button in the top-right corner of the chart's widget. There is also a *trash* button which removes the chart entirely. These buttons can be seen on the top right corner of Figure 9.

Some of the slices of *Pie chart* may be so small, that their labels are written over each other. To highlight any of the slice labels, mouse hover over the slice. This is illustrated in Figure 9.

If there is a large amount of data, line charts cannot show them all at once and they become paginated. When this happens, buttons to move the chart forward or backward on the x-axis will appear to the right and left of the chart. This can be seen in Figure 10.

2.9 Posts

The third tab in job's data page (Section 2.5) displays all acquired posts of the job. This list can be seen in Figure 11. Since there will mostly be a large number of these posts, they are paginated, displaying 20 posts in one page. They can also be filtered out by three criteria:

- Date range - specify a date range, from which the filtered posts should be picked. The filtered posts will be then picked from the 0:00 of the beginning date to the 23:59 of the ending date,
- Contain words - specify words, from which at least one must be contained in the filtered posts,
- Exclude words - specify words that the filtered posts can't contain.

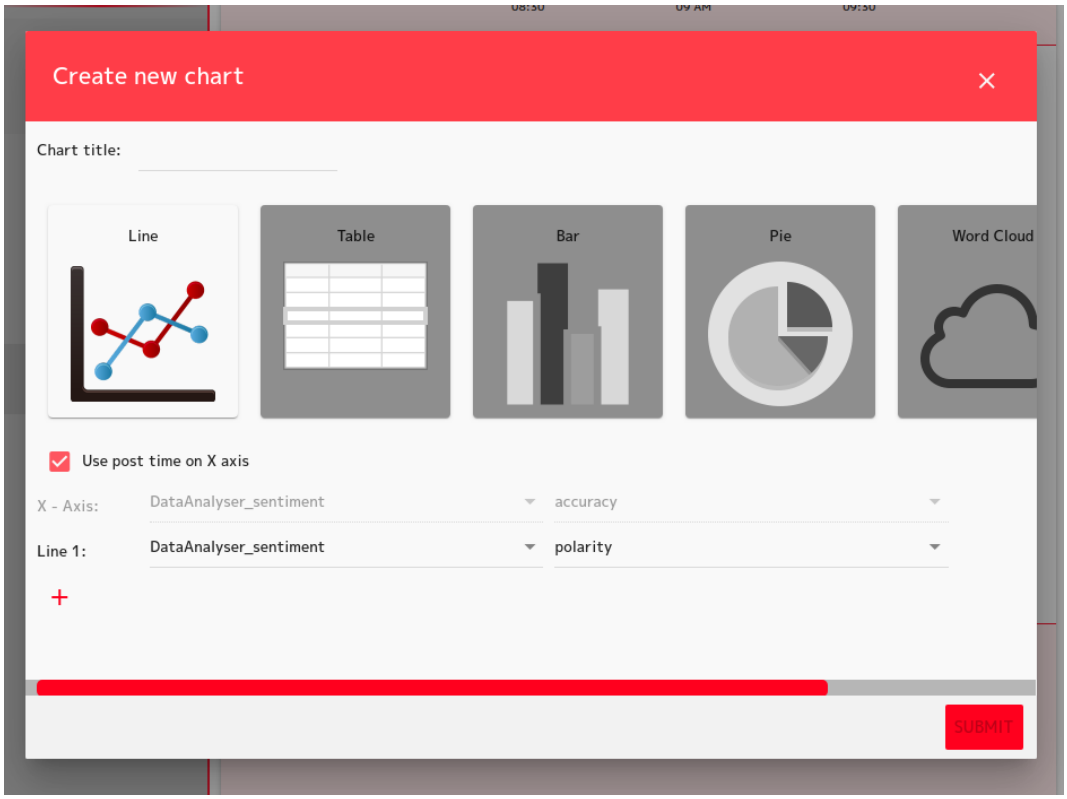

Figure 8: Chart creation dialogue with line chart type and sentiment analysis property selected

The posts are updated automatically as the filters are updated. However, as well as charts, they are not updated automatically as new posts arrive in the platform. They can be refreshed by clicking on the *bent arrow* button in the top-right corner of the posts list. The button can be seen on the top right part of Figure 11.

2.9.1 Post Detail

Any of the posts can be clicked on, to unroll more information about it. Only properties that are available are displayed. These properties are: *post's original identifier*, *post's original text* (if it was translated by the acquirer), *language* (in *ISO 3166 Alpha-2 format*) of the post, *identifier of author* of the post and *date* at which the post was posted. This is also illustrated in Figure 11. There, the very first post is unrolled, showing its properties.

2.9.2 Export

These (un)filtered posts can be also exported into CSV format file by pressing button *Export to CSV* located at the bottom of the posts list. It is displayed only if the list contains at

least one post.

The exported CSV contains all the posts, not only posts from the currently displayed page. It contains all the attributes mentioned in Section 2.9.1.

3 Advanced User Documentation

Next to using only frontend, Socneto platform has several advanced features:

- Using custom data set for analyses,
- Adding custom analyzer or acquirer

3.1 Custom Datasets

Socneto allows users to use their own datasets as long as they comply with the following requirements.

- **File format** - supported file formats are `csv` and `json`. In case of json, the file is assumed to be in `json line` format, where each line is a json record. We will refer to this format as to plain `json`,
- **Content** must be transformable into an entity `Post`. The transformation is done via mapping file accompanying the dataset.

3.1.1 Dataset Transformation

The components processing data from custom dataset expect each record to have the following fields:

- `originalPostId` - a unique post string identifier (for example `1234abcd-efgh`),
- `text` - the text of the given post,
- `language` - the language code of the post (ISO 3166 Alpha-2 format). The post be in any language, but only sentiment of English posts can be analysed. Example: `en`, `cs` or `null`,
- `source` - field prefilled by the Data acquirer. Example: `twitter_acquirer`,
- `authorId` - the name or id of the author. Can be `null`,
- `dateTime` - date time when the post was created (in ISO8601 format). Example: `2020-01-03T23:42:53`

Socneto implements basic automatic transformation that maps records from the dataset to the `Post` format. The transformation requires a json configuration file specifying:

- **mapping** of fields from dataset records to `Post`,

- **specification** of fixed values (e.g. language may be the same across the dataset or the value is not present),
- **file format** - either ‘csv‘ or ‘json‘,
- **format specific** configuration

The mapping configuration files have the following fields:

- `dataFormat` - format of the dataset, `json` or `csv`. This field is required,
- `mappingAttributes` - root element with mappings and element specific values. This field is required,
- `fixedValues` - map where a key is a name of a field of the `Post` entity and the value is a text of the field. This field is not required,
- `indices` (for `csv` data format only) - map where a key is a name of a field of the `Post` entity and value is an index of the record with target value. Required for `csv` file format,
- `elements` (for `json` data format only) - map where a key is a name of a field of the `Post` entity and value is a name of an element of the record with target value. Mapping to names of csv columns is not supported, use `indices` instead. Required for `json` file format,
- `hasHeaders` - indication whether the `csv` file contains headers or not (they will be skipped if true). Required for `csv` file format,
- `dateTimeFormatString` - if dateTime mapping is specified, then this field is used as mask for custom date time formats. If no format string is specified then the default one is used. This field is not required.

3.1.2 JSON Mapping File Example

```
{
    "dataFormat":"json",
    "mappingAttributes":{
        "hasHeaders":true,
        "dateTimeFormatString":"yyyy'-'MM'-'dd'T'HH':'mm':'ss",
        "fixedValues":{
            "language":"en"
        },
        "elements":{
            "originalPostId":"PostId",
            "text":"Text",
            "authorId":"UserId",
            "dateTime":"PostDateTime"
        }
```

```
    }
}
```

This file is used to map the following json record:

```
{
    "PostId":"1195555506847744",
    "Text":"The movie xyz was excelent",
    "UserId":"abc",
    "PostDateTime":"2019-11-16T18:48:03"
}
```

The resulting `Post` (shown in json) is the following:

```
{
    "originalPostId":"1195555506847744",
    "text":"The movie xyz was excelent",
    "source":"twitter_acquirer", // filled with the acqurier
    "authorId":"abc",
    "dateTime":"2019-11-16T18:48:03",
    "language":"en"
}
```

3.1.3 CSV Mapping File Example

```
{
    "dataFormat":"csv",
    "mappingAttributes":{
        "hasHeaders":true,
        "dateTimeFormatString":"ddd MMM dd HH:mm:ss PDT yyyy",
        "fixedValues":{
            "language":"en"
        },
        "indices":{
            "originalPostId":1,
            "text":5,
            "authorId":4,
            "dateTime":2
        }
    }
}
```

This mapping file is used to map the following record (in csv format):

```
target,id,date,flag,user,text
0,abd087, Sat May 16 23:58:44 UTC 2008,xyz,robotickilldozr,xyz is fun
```

The resulting `Post` (shown in json) is the following:

```
{
```

```
    "originalPostId":"abc087",
    "text":"xyz is fun",
    "source":"twitter_acquirer", // filled with the acqurier
    "authorId":"abc",
    "dateTime":"2009-05-16T23:58:44",
    "language":"en",
}
```

CAUTION: When parsing of the date time is not successful, then the `dateTime` field of the resulting `Post` entity is set to '0001-01-01T00:00:00'.

NOTE: Mappings do not support any nested json files. If user needs so, then there is a possibility to extend the system with own implementation supporting such a mapping.

When transformation fails to produce valid `Post` object then the object is not produced and will be ignored. User is noticed about the problem with a warning message that can be found in the logs.

3.1.4 Full Example of Adding Custom Dataset

The following text works with this dataset that can be found on kaggle.com. The dataset will be referred to as `tweets.csv`. It is a single `csv` file with 1.6 million records with the following columns:

- `target` - the polarity of the tweet e.g. 0 = negative, 2 = neutral, 4 = positive,
- `id` - the id of the tweet,
- `date` - the date when the tweet was tweeted,
- `flag` - the query which was used to obtains the given tweet. If there is no query, then this value is NO_QUERY,
- `user` - the user that tweeted the tweet,
- `text` - the content of the tweet.

The mapping configuration file used (referred to as `tweets.mapping`) is the same as was presented in the Section 3.1.3. The following steps lead to successfully feeding Socneto with these data.

1. **Upload files** - connect to `http://acheron.ms.mff.cuni.cz:39111` and login with the following credentials: username `socnetoadmin`, password `TajnOHeslo`. After logging in hit red plus button and create bucket `movie-tweets`. Hit the upload button again and upload `tweets.csv` and `tweets.mapping` to that bucket. After this step, you should see the screen like in Figure 12,
2. **Create a job** - create a new job via frontend, using process described in Section 2.4.2. Use acquirer called `DataAcquier_CustomStaticData`.

3.2 Custom Acquirers and Analyzers

The process of implementing custom acquirers and analyzers is thoroughly described in *Development Documentation* in *Section 3.10 Extensibility*.

If a new component is correctly implemented and started, it will be displayed in the job creation dialogue from Section 2.4.2.

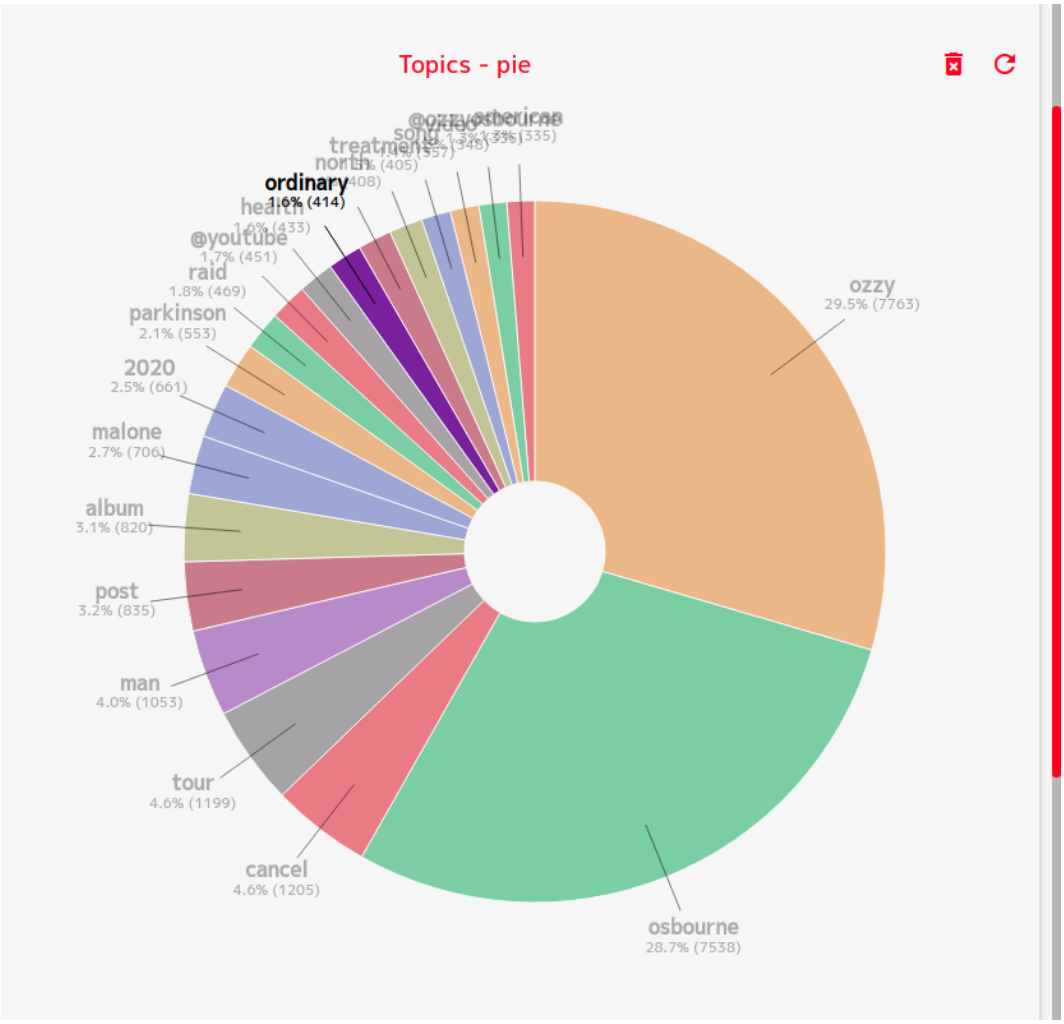

Figure 9: Highlighting one of pie chart's slices

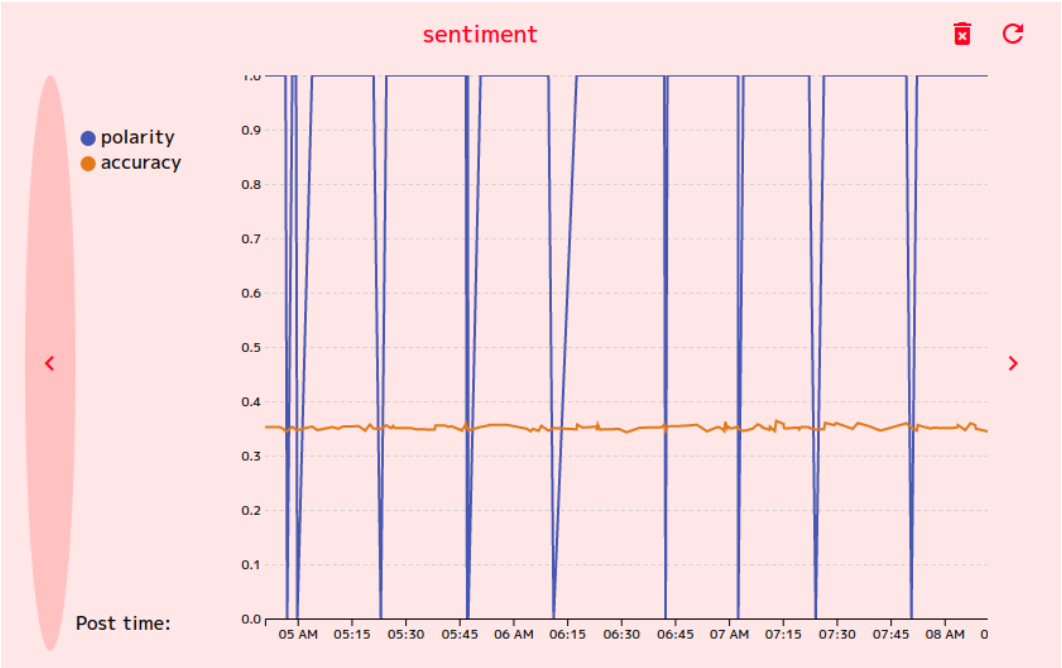

Figure 10: Line chart with 'forward' and 'backward' buttons on the sides

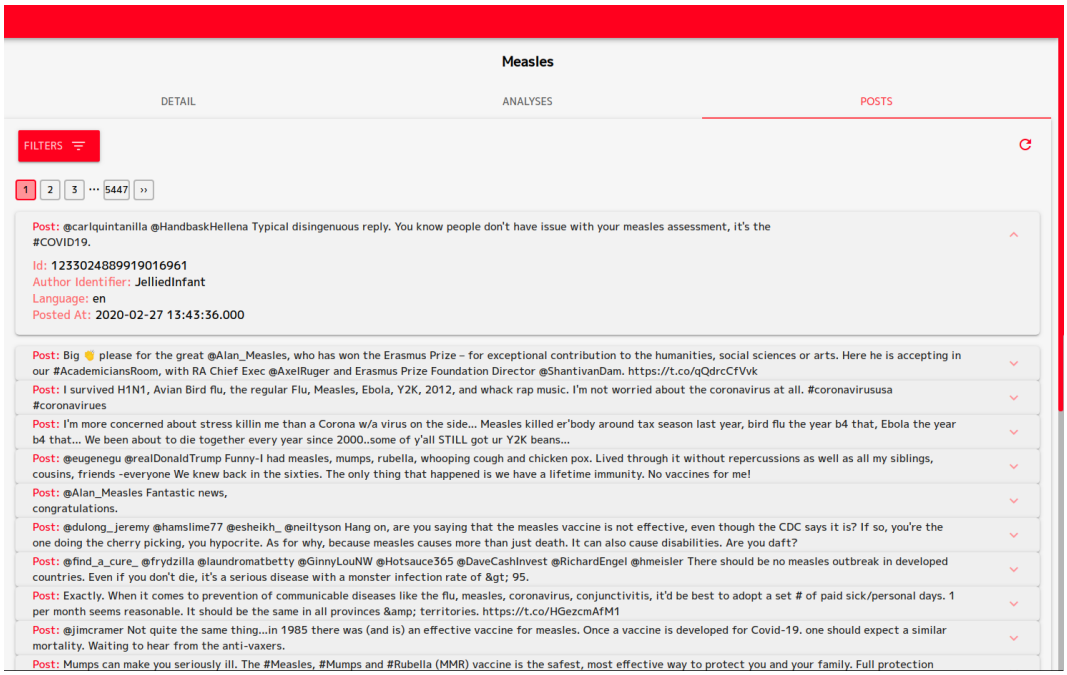

Figure 11: Illustration of filtered posts list with some posts' detail unrolled

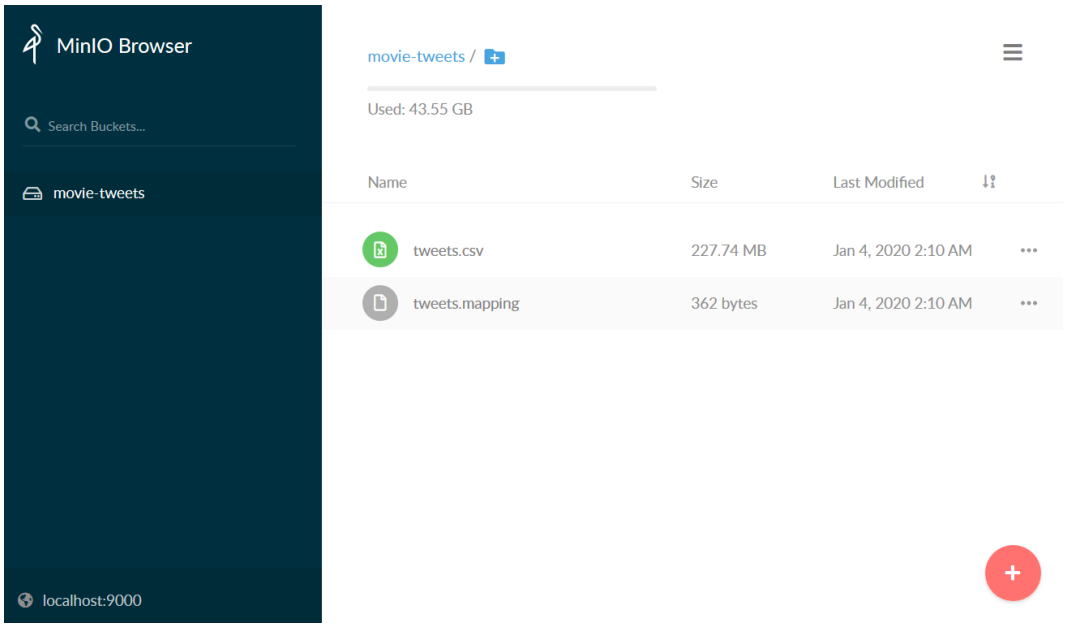

Figure 12: Example of files uploaded to minio server