Grant Agreement nº	732463
Project Acronym:	OpenReq
Project Title:	Intelligent Recommendation Decision Technologies for Community-Driven Requirements Engineering
Call identifier:	H2020-ICT-2016-1
Instrument:	RIA (Research and Innovation Action)
Topic	ICT-10-16 Software Technologies
Start date of project	January 1st, 2017
Duration	36 months

D2.2+Requirements Intelligence Engine version 1

Lead contractor:	ENG
Author(s):	ENG, HITEC
Submission date:	July 2018
Dissemination level:	PU

Project co-funded by the European Commission under the H2020 Programme.

Abstract: This document describes the microservices that have been developed in the OpenReq project in order to provide analytical services that will be used within the process in order to classify, extract requirements and anomalies from tweets.

This document by the OpenReq project is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 Unported License.

This document has been produced in the context of the OpenReq Project. The OpenReq project is part of the European Community's h2020 Programme and is as such funded by the European Commission. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability is respect of this document, which is merely representing the authors view.

TABLE OF CONTENTS

1. Introduction

This document is structured into the following chapters:

1) Architecture: the architecture of the OpenReq project is described, with all its components (Knowledge Ontology on which OpenReq is based, the REST services made available by the platform and the various applications that use the Openreq services);
2) The description of the microservices that have been developed to enrich the OpenReq platform. For each microservice, a brief description of the microservice is given, a sequence diagram that reports the interaction between the different components referred to in the microservice and an example of the use of each microservice endpoint.
3) The problems encountered in the creation of microservices

We conclude by delineating further services and improvements to the existing ones.

2. Requirements Intelligence role in the architecture

The overall OpenReq architecture is composed of the OpenReq REST Services and the different stakeholder applications who use the services (see Figure 2). Besides the trial partner applications, the OpenReq Prototype (a showcase of major OpenReq functionalities) accesses the different OpenReq services. The OpenReq Prototype includes basic OpenReq functionalities offered to end users through an understandable and user-friendly user interface. In all cases where end users interact with OpenReq functionalities, a special focus will be given on usability.

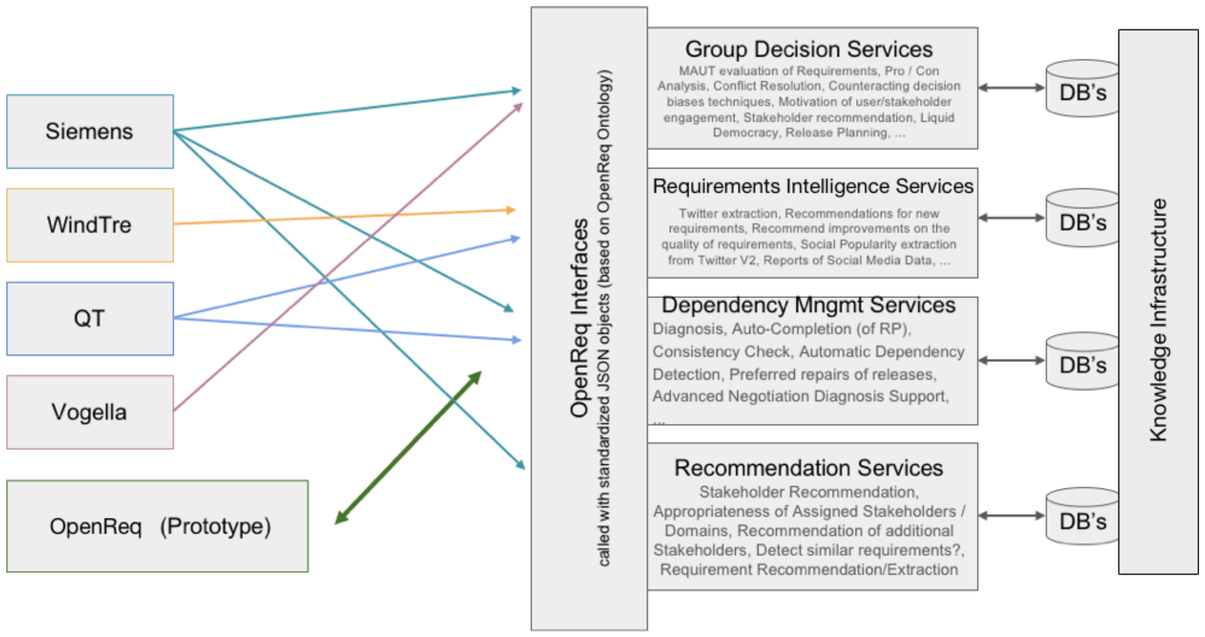

Siemens
WindTre
QT
Vogella
OpenReq (Prototype)
OpenReq Interfaces
called with standardized JSON objects (based on OpenReq Ontology)
Group Decision Services
MAUT evaluation of Requirements, Pro / Con Analysis, Conflict Resolution, Counteracting decision biases techniques, Motivation of user/stakeholder engagement, Stakeholder recommendation, Liquid Democracy, Release Planning, ...
Requirements Intelligence Services
Twitter extraction, Recommendations for new requirements, Recommend improvements on the quality of requirements, Social Popularity extraction from Twitter V2, Reports of Social Media Data, ...
Dependency Mngmt Services
Diagnosis, Auto-Completion (of RP), Consistency Check, Automatic Dependency Detection, Preferred repairs of releases, Advanced Negotiation Diagnosis Support, ...
Recommendation Services
Stakeholder Recommendation, Appropriateness of Assigned Stakeholders / Domains, Recommendation of additional Stakeholders, Detect similar requirements?, Requirement Recommendation/Extraction
DB's
DB's
DB's
DB's
Knowledge Infrastructure

Figure 1: OpenReq overall architecture.

The OpenReq Prototype and the industry trials exploit the basic functionalities provided by the OpenReq services. Furthermore, services themselves exploit and integrate the services of other components. For example, release planning uses functionalities of dependency detection, recommendation, and group decision making (for simplicity, this is not considered in the architecture figure). In the following, we give a short overview on the different services, knowledge infrastructure, and interfaces shown in Figure 1 (for more details we refer to the DoA).

a. **Recommendation.** The recommender engine will provide a set of basic microservices—e.g., querying for new requirements, responsible stakeholders, or reusable requirements. A conversation with the system can be triggered depending on the type of the query and the stakeholder activity. Generating recommendations will be delegated to collaborative or content-based recommenders.

b. **Dependency Management.** This component uses a combination of content-based recommendation, sentiment analysis, and natural language processing that support the detection and extraction of requirements dependencies and requirements tracing. A conflict resolver is used for repairing inconsistent requirements,

c. **Group Decision.** This component offers support for groups of users in group decision processes. In a first step, the preferences of the different stakeholders have to be collected (for example, regarding a specific requirement). After the preference acquisition is done, this component also supports the moderation of the decision process in such a way that consensus among stakeholders can be achieved in case of contradicting preferences.

d. **Requirements Intelligence.** This component will encapsulate an analytics backend and include text-mining algorithms that allow the analysis of natural language in text-based documents or user feedback (user feedback can be either explicit or implicit - see Figure 2). In addition to that also interactive visualization will be supported by this component. In particular, interactive visualization supports stakeholders in visualizing descriptive and predictive analytics data. An example of such a visualization is shown in Figure 3. Figure presents the trend of different app review types (e.g., a user requests a new feature or a bug to be fixed) over time, for a specific app and over its different versions.

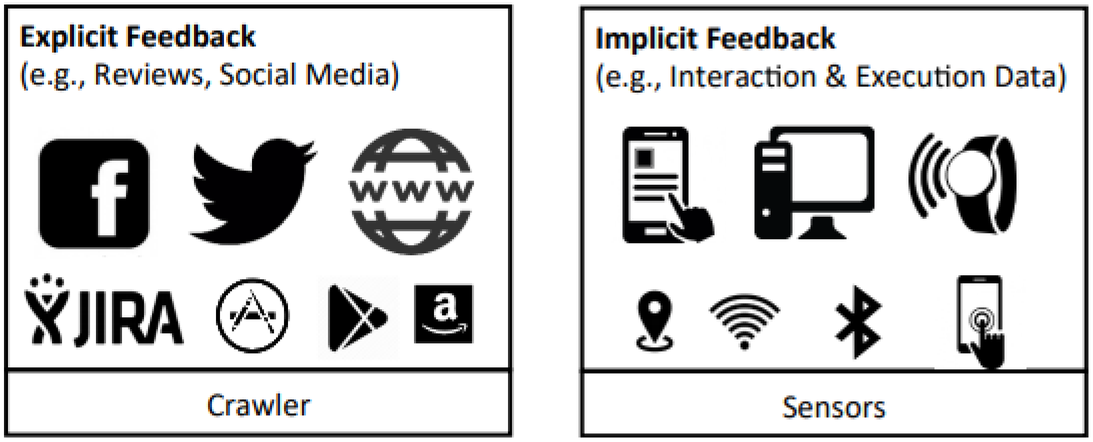

Explicit Feedback
(e.g., Reviews, Social Media)
WWW
JIRA
Crawler
Implicit Feedback
(e.g., Interaction & Execution Data)
Sensors

Figure 2: Example for explicit and implicit feedback

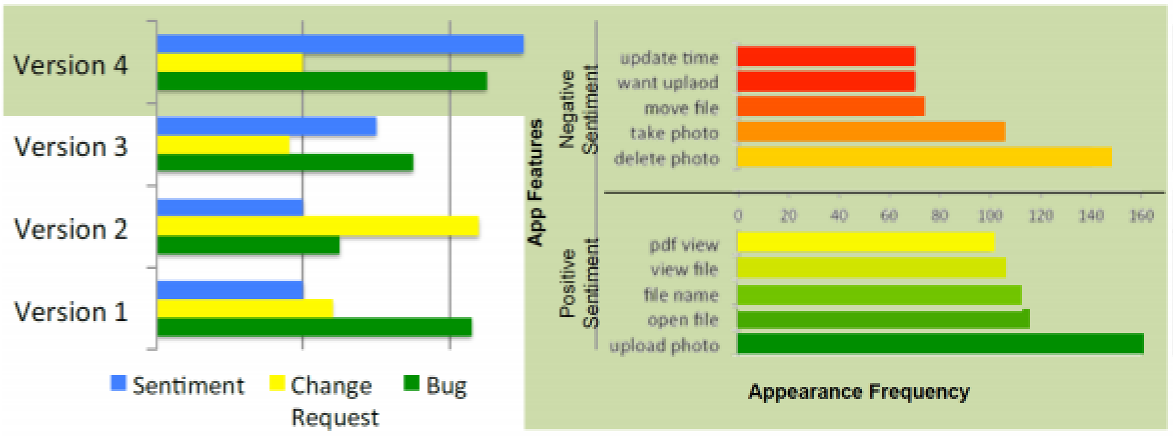


Figure 3: Example trend of different app review types

e. **Knowledge Infrastructure.** This component is responsible for managing OpenReq ontologies, glossaries, indexes (e.g., references), stakeholder profiles, user feedback and usage and interaction logs. OpenReq Ontologies will be (1) the RE ontology (language for modelling requirements including concrete instances, i.e., the requirements model), (2) the reuse and patterns ontology (language for requirements reuse and patterns), and (3) different domain ontologies (modelling the domains of the applications, e.g. the trials). The OpenReq reuse and pattern catalogue provides a reusable knowledge base about requirements. The catalogue will be organized according to one or more classification schemas that will be derived from some of the ontologies mentioned above. Communication among OpenReq components and the integration of OpenReq components with the trial clients (e.g., the Siemens Doors client) will be supported by OpenReq Interfaces. The approach to provide these interfaces will be the following.

f. OpenReq Interfaces. The OpenReq Interfaces will provide open, unified interfaces for easily integrating OpenReq into external tools in form of connectors (see Figure 1). Examples for high-level interfaces include among others (a) Register/Unregister for Recommendation, (b) push context, (c) display recommendation, (d) open artefact and (e) configure. As a proof of concept, we will integrate these interfaces in the tools used by the OpenReq industry partners and Open Source communities within the scope of the trial implementations. To assure the accessibility of OpenReq in Cloud contexts, OpenReq will be built upon the following service infrastructure.

3. Microservices

In this section, we describe each microservice, report their interaction in a sequence diagram, and show usage examples.

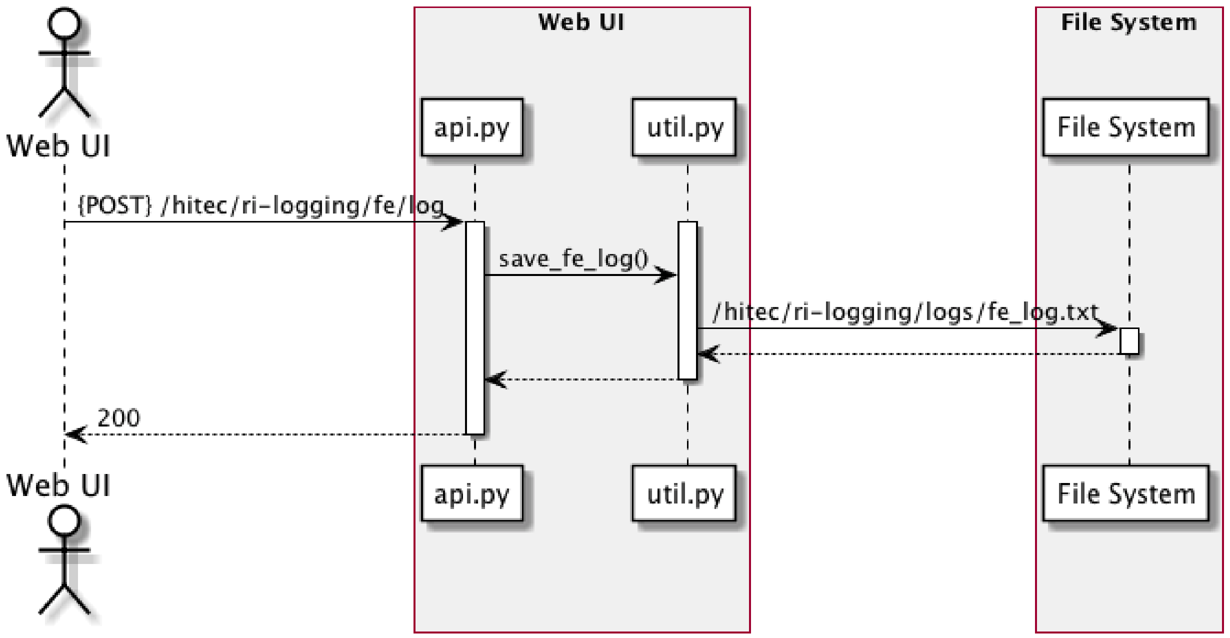
3.1. ri-collection-explicit-feedback-web

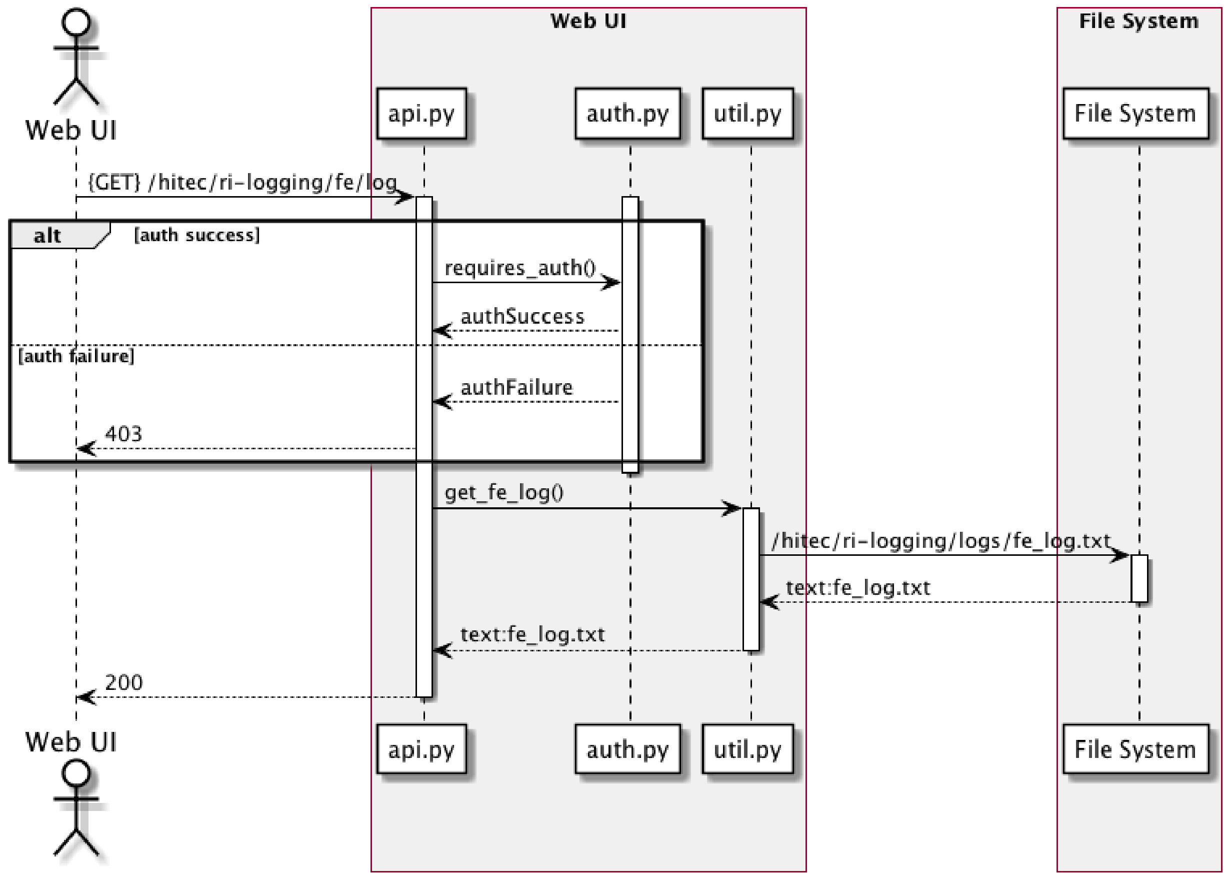
This microservice collects implicit feedback from user interactions with OpenReq user interfaces and user/microservice interactions with the backend.

The UI interactions are captured by a Javascript library that only needs to be imported by the UI. No integration within the UI code (except *<script src="<url>"></script>)* is needed. The events in the UI are logged through this microservice API and stored in a text file or database. The logs are accessible through this microservices API.

The backend interaction is captured by the web server software application, such as Apache HTTP Server, not part of the microservice. Every request and response from OpenReq microservices that reaches the backend is logged in a file. The log file is accessible through this microservices API but limited in its accessibility by the owners of a bearer token - currently the admins of the system.

Sequence diagram





Example usage

URL	https://openreq.esl.eng.it/eng/hitec/fe/log
Method	POST
URL params	None
Header	{"sessionId": "<sessionId>"}

Data params	{
	 "_id" : ObjectId("5b17ec2b2fd3113325b5a7d7"),
	 "ip" : "127.0.0.1",
	 "event_type" : "mouseover",
	 "header" : {
	 "Host" : "0.0.0.0:9798",
	 "Connection" : "keep-alive",
	 "Content-Length" : "26329",
	 "Pragma" : "no-cache",
	 "Cache-Control" : "no-cache",
	 "Sessionid" : "hsuPP5K47wmO9QgXNhtN",
	 "Origin" : "http://localhost",

	"User-Agent" : "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36",
	"Content-Type" : "application/json",
	"Accept" : "*/*",
	"Dnt" : "1",
	"Referer" : "http://localhost/rome-demo/index.html",
	"Accept-Encoding" : "gzip, deflate",
	"Accept-Language" : "en-US,en;q=0.9,fr;q=0.8,de;q=0.7,es;q=0.6,it;q=0.5,ru;q=0.4,ro;q=0.3"
	},
	"body" : {
	"type" : "mouseover",
	…<more items>
	}
	}

URL	https://openreq.esl.eng.it/eng/hitec/js/logger
Method	GET
URL params	none
Header	none
Data params	none

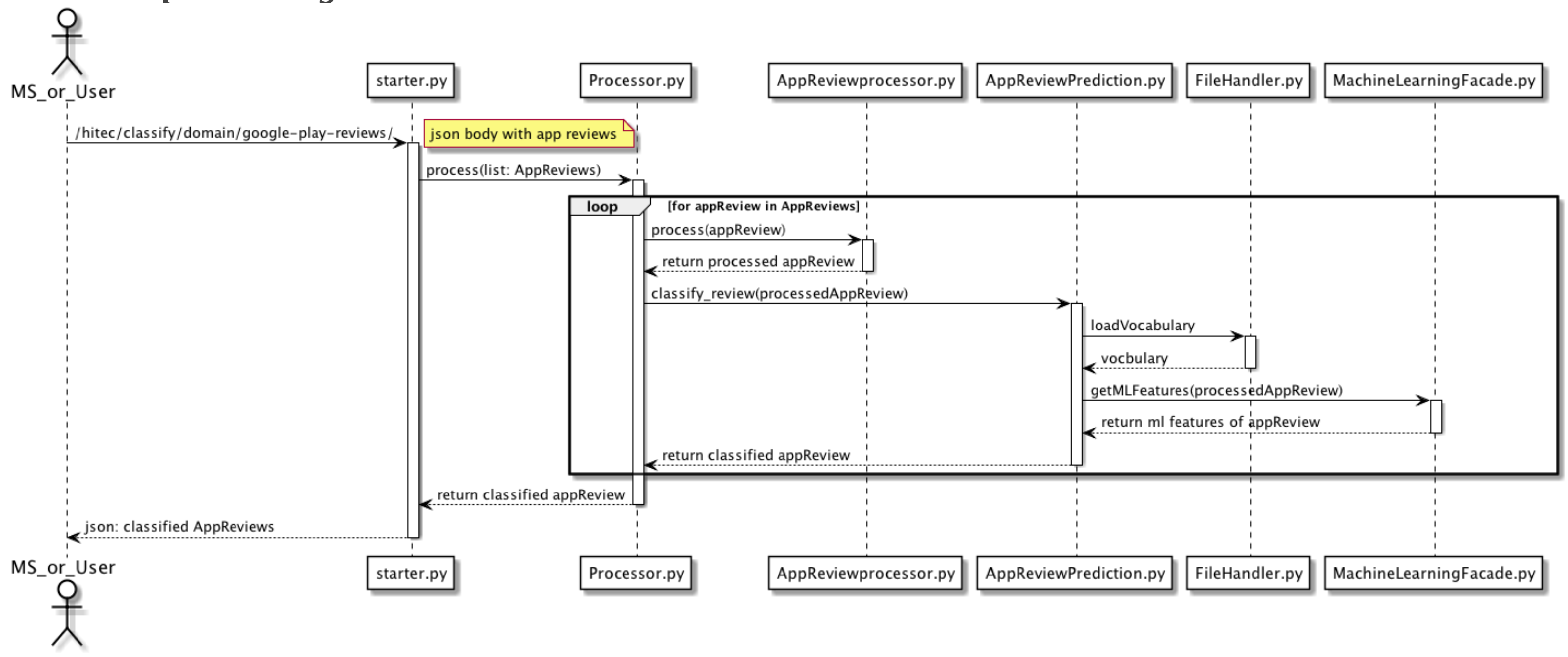
URL	https://openreq.esl.eng.it/eng/hitec/be/log
Method	GET
URL params	none
Header	{"Authorization": "Bearer <bearer-token>"}
Data params	none

URL	https://openreq.esl.eng.it/eng/hitec/
Method	GET
URL params	none
Header	{"Authorization": "Bearer <bearer-token>"}
Data params	none

3.2.ri-analytics-classification-google-play-review

The goal of this microservice is to classify a list of app reviews as either a "bug report" or a "feature request". The source code to get classified reviews (including data cleaning, machine learning feature extraction, and classification based on pre-trained models code that is necessary to perform these tasks is bundled in a single Docker container. The response of the microservice is a list of app reviews that now includes the class they belong to.

Sequence diagram


MS_or_User
starter.py
Processor.py
AppReviewprocessor.py
AppReviewPrediction.py
FileHandler.py
MachineLearningFacade.py
/hitec/classify/domain/google-play-reviews/
json body with app reviews
process(list: AppReviews)
loop
[for appReview in AppReviews]
process(appReview)
return processed appReview
classify_review(processedAppReview)
loadVocabulary
vocbulary
getMLFeatures(processedAppReview)
return ml features of appReview
return classified appReview
return classified appReview
json: classified AppReviews

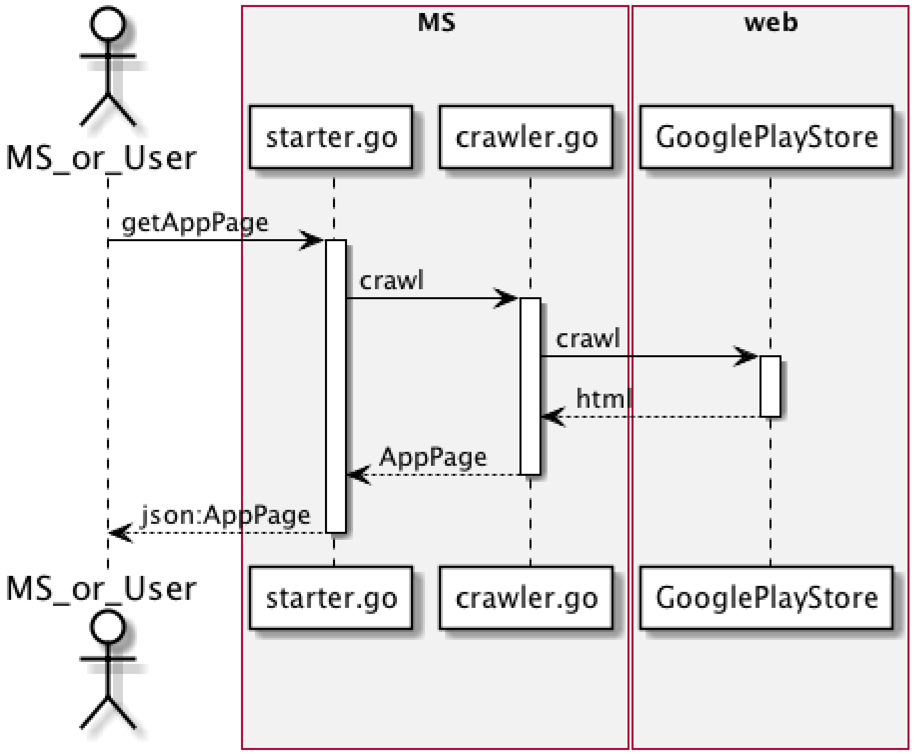
Example usage

URL	https://openreq.esl.eng.it/eng/hitec/classify/domain/google-play-reviews/
Method	POST
URL params	None
Data params	[{ “review_id”: "05df34353efd", "package_name":”com.myapp”, "author":”johndoe”, "date_posted":20180524, “rating”:5, “title”: “I like this app because…”, “body”: “it has really nice features”, “perma_link”: “https://….”, “cluster_is_feature_request”: false, “cluster_is_problem_report”: true },]

3.3.ri-collection-explicit-feedback-google-play-page

The goal of this microservice is to collect all available metadata of an app page from the Google Play Store—such as the name of the app, the category, and the average rating. The response contains all information with respect to that app page in JSON format.

Sequence diagram



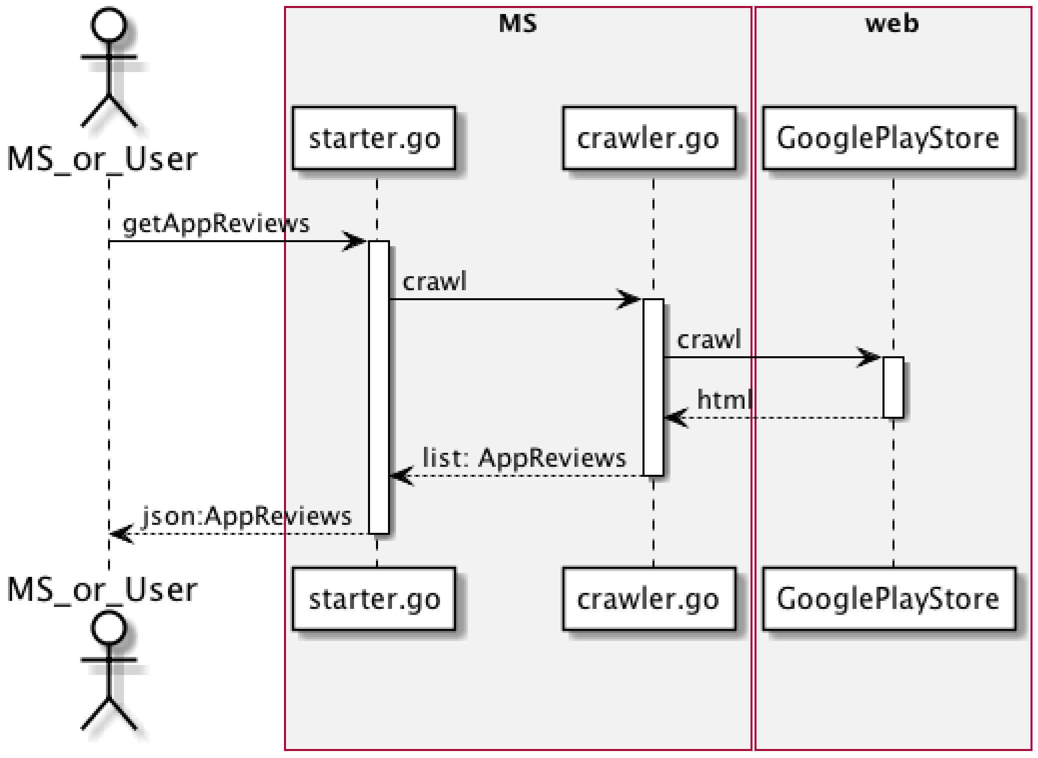
Example usage

URL	https://openreq.esl.eng.it/eng/hitec/crawl/app-page/google-play/{package_name}
Method	GET
URL params	Package_name: the package name of the app page to crawl
Data params	None

3.4.ri-collection-explicit-feedback-google-play-review

The goal of this microservice is to collect data from the Google Play Store—the official store for Android apps. In particular, this service collects the user reviews of a given app. The response contains a list of app reviews belonging to a certain app in JSON format.

Sequence diagram



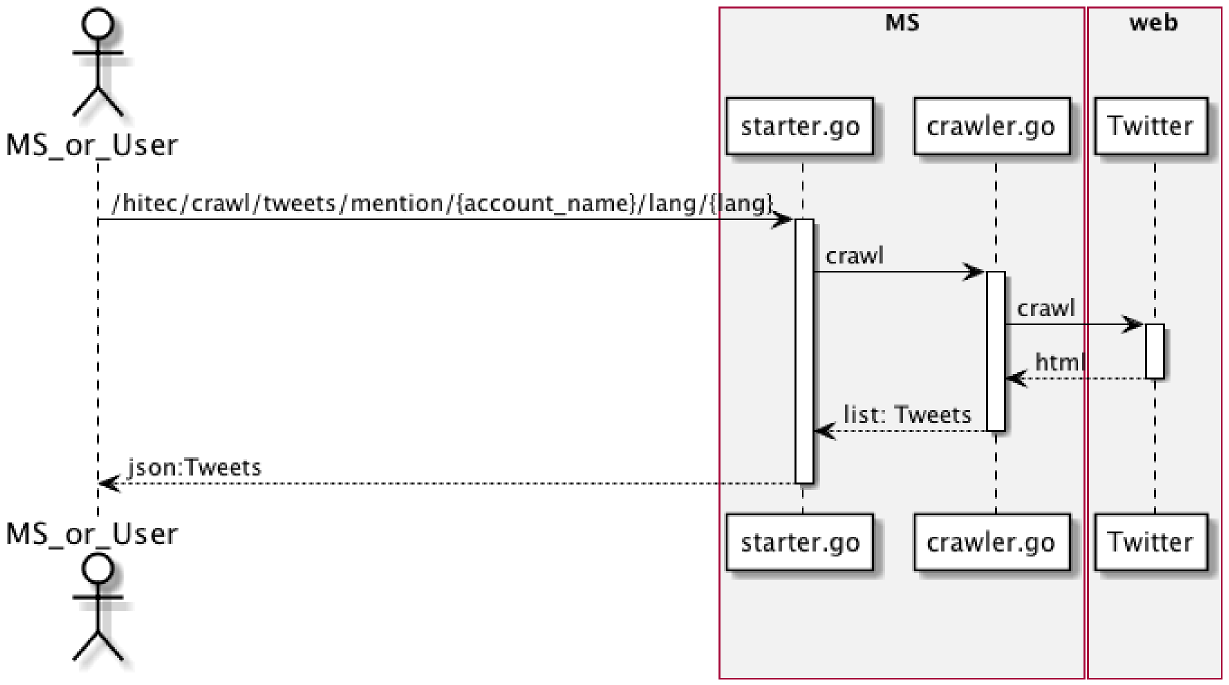
Example usage

URL	https://openreq.esl.eng.it/eng/hitec/crawl/app-reviews/google-play/{package_name}/limit/{limit}
Method	GET
URL params	package_name : the name of the App of which we crawl the reviews, limit: the maximum number of reviews to retrieve
Data params	None

3.5.ri-collection-explicit-feedback-twitter

The goal of this microservice is to collect data from Twitter. In particular, this service collects tweets that mention a given account. The response contains a list of tweets in JSON format.

Sequence diagram



Example usage

URL	https://openreq.esl.eng.it/eng/hitec/crawl/tweets/mention/{account_name}/lang/{lang}
Method	GET
URL params	account_name : the name of the Twitter profile to crawl, lang: the language the tweets should be written in
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/crawl/tweets/mention/{account_name}/from/{date}/lang/{lang}
Method	GET
URL params	account_name : the name of the Twitter profile to crawl, date: specify the date from which the crawler starts,

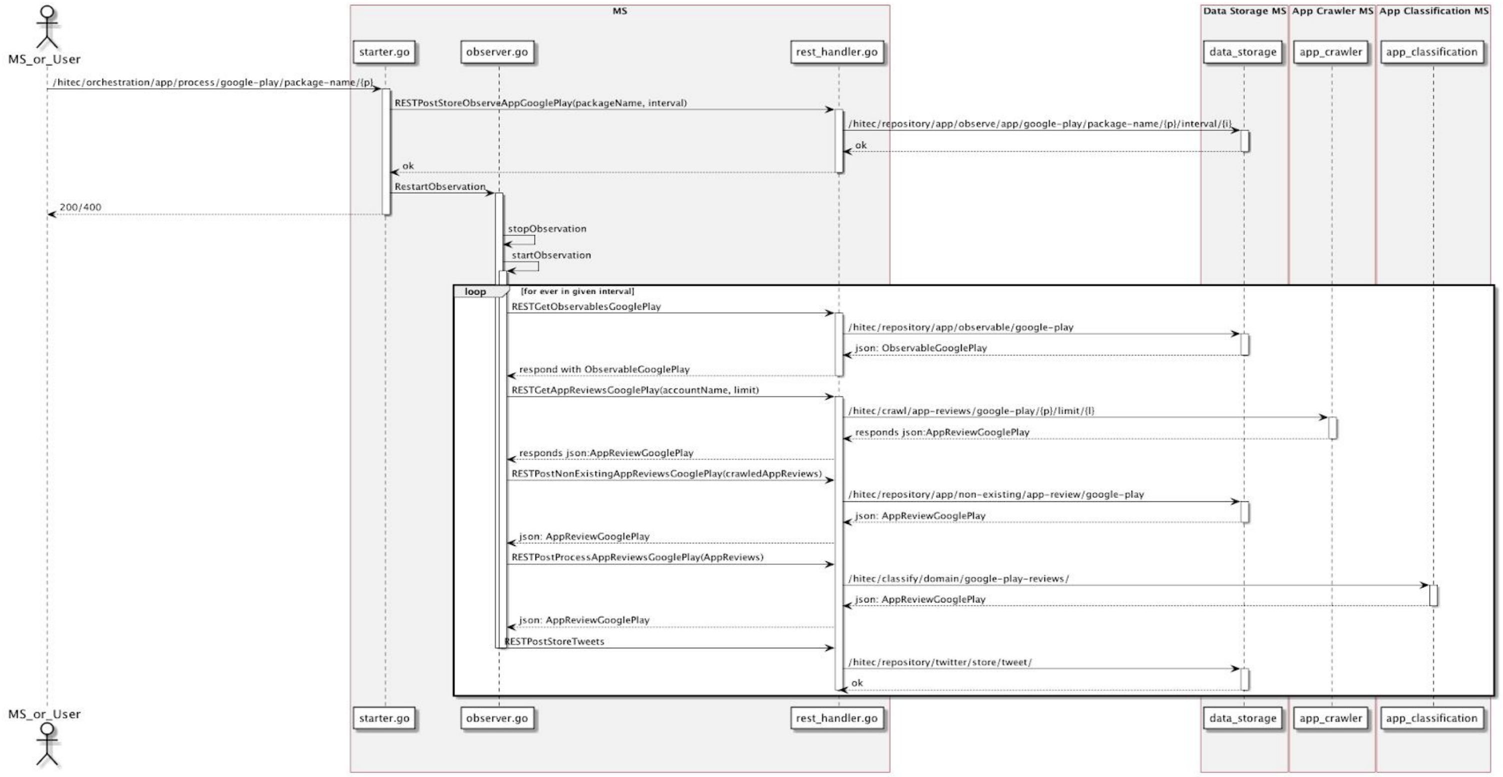
	lang: the language the tweets should be written in
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/crawl/tweets/mention/{account_name}/history-in-days/{days}/lang/{lang}
Method	GET
URL params	account_name : the name of the Twitter profile to crawl, days: the days we want to crawl, counting backwards from current date, lang: the language the tweets should be written in
Data params	None

3.6.ri-orchestration-app

This microservice is responsible to coordinate all microservices that belong to the domain of app store data. The main goal of this microservice is to define apps that should continuously be observed by OpenReq. In a given interval, the apps and their reviews are crawled, classified, and stored in the database.

Sequence diagram


MS
Data Storage MS
App Crawler MS
App Classification MS
MS_or_User
starter.go
observer.go
rest_handler.go
data_storage
app_crawler
app_classification
/hitec/orchestration/app/process/google-play/package-name/{p}
RESTPostStoreObserveAppGooglePlay(packageName, interval)
/hitec/repository/app/observe/app/google-play/package-name/{p}/interval/{i}
ok
ok
RestartObservation
200/400
stopObservation
startObservation
loop
[for ever in given interval]
RESTGetObservablesGooglePlay
/hitec/repository/app/observable/google-play
json: ObservableGooglePlay
respond with ObservableGooglePlay
RESTGetAppReviewsGooglePlay(accountName, limit)
/hitec/crawl/app-reviews/google-play/{p}/limit/{l}
responds json:AppReviewGooglePlay
responds json:AppReviewGooglePlay
RESTPostNonExistingAppReviewsGooglePlay(crawledAppReviews)
/hitec/repository/app/non-existing/app-review/google-play
json: AppReviewGooglePlay
json: AppReviewGooglePlay
RESTPostProcessAppReviewsGooglePlay(AppReviews)
/hitec/classify/domain/google-play-reviews/
json: AppReviewGooglePlay
json: AppReviewGooglePlay
RESTPostStoreTweets
/hitec/repository/twitter/store/tweet/
ok
MS_or_User
starter.go
observer.go
rest_handler.go
data_storage
app_crawler
app_classification

Example usage

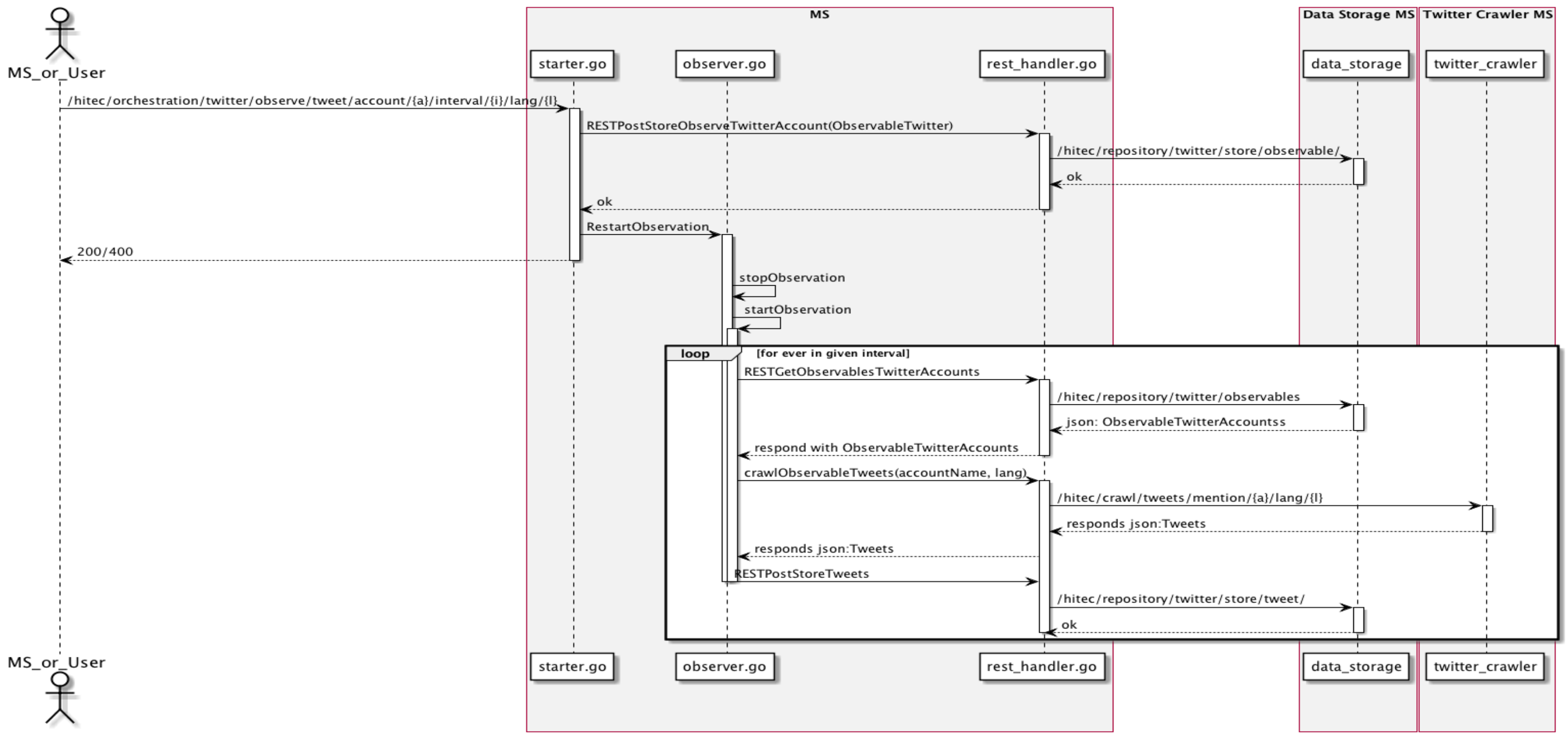
URL	https://openreq.esl.eng.it/eng/hitec/orchestration/app/process/google-play/package-name/{package_name}
Method	POST
URL params	package_name : the package name of the app to observe
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/orchestration/app/observe/google-play/package-name/{package_name}/interval/{interval}
Method	POST
URL params	package_name: the package name of the app to observe, interval: how often data should be crawled
Data params	None

3.7.ri-orchestration-twitter

This microservice is responsible to coordinate all microservices that belong to the domain of Twitter data. The main goal of this microservice is to define Twitter accounts that should continuously be observed. In its current state, the microservice crawls tweets that mention a given account and stores the result in the database.

Sequence diagram


MS
Data Storage MS
Twitter Crawler MS
MS_or_User
starter.go
observer.go
rest_handler.go
data_storage
twitter_crawler
/hitec/orchestration/twitter/observe/tweet/account/{a}/interval/{i}/lang/{l}
RESTPostStoreObserveTwitterAccount(ObservableTwitter)
/hitec/repository/twitter/store/observable/
ok
ok
RestartObservation
200/400
stopObservation
startObservation
loop
[for ever in given interval]
RESTGetObservablesTwitterAccounts
/hitec/repository/twitter/observables
json: ObservableTwitterAccountss
respond with ObservableTwitterAccounts
crawlObservableTweets(accountName, lang)
/hitec/crawl/tweets/mention/{a}/lang/{l}
responds json:Tweets
responds json:Tweets
RESTPostStoreTweets
/hitec/repository/twitter/store/tweet/
ok
MS_or_User
starter.go
observer.go
rest_handler.go
data_storage
twitter_crawler

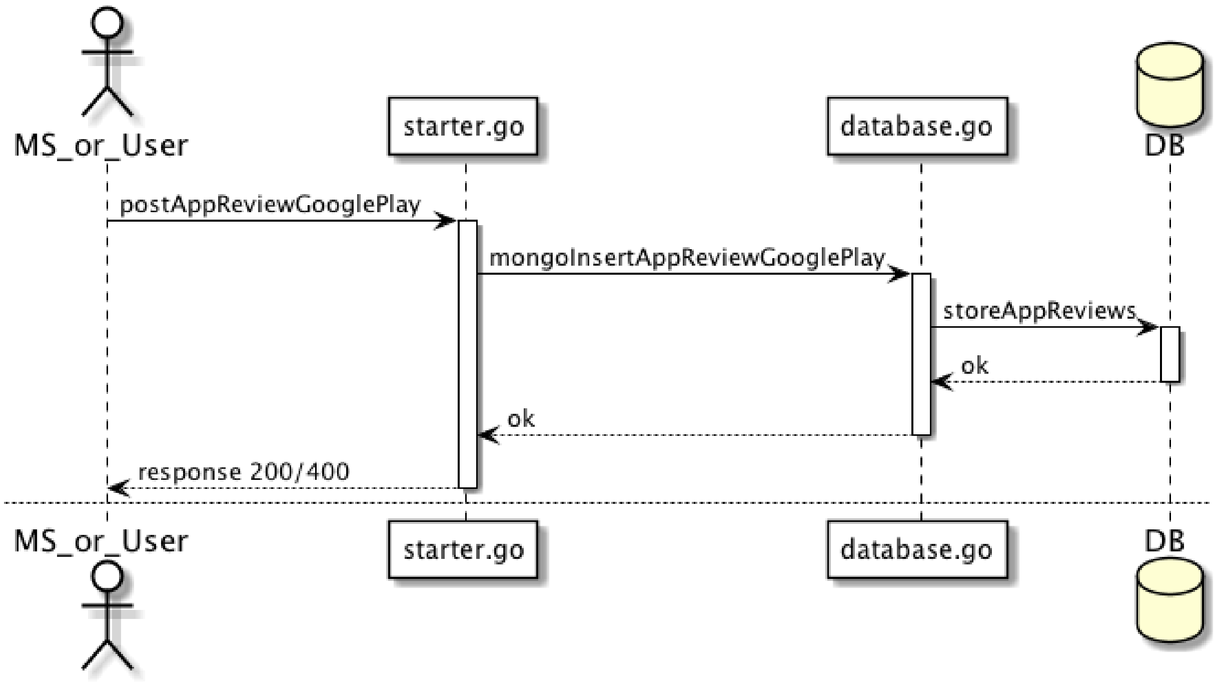
Example usage

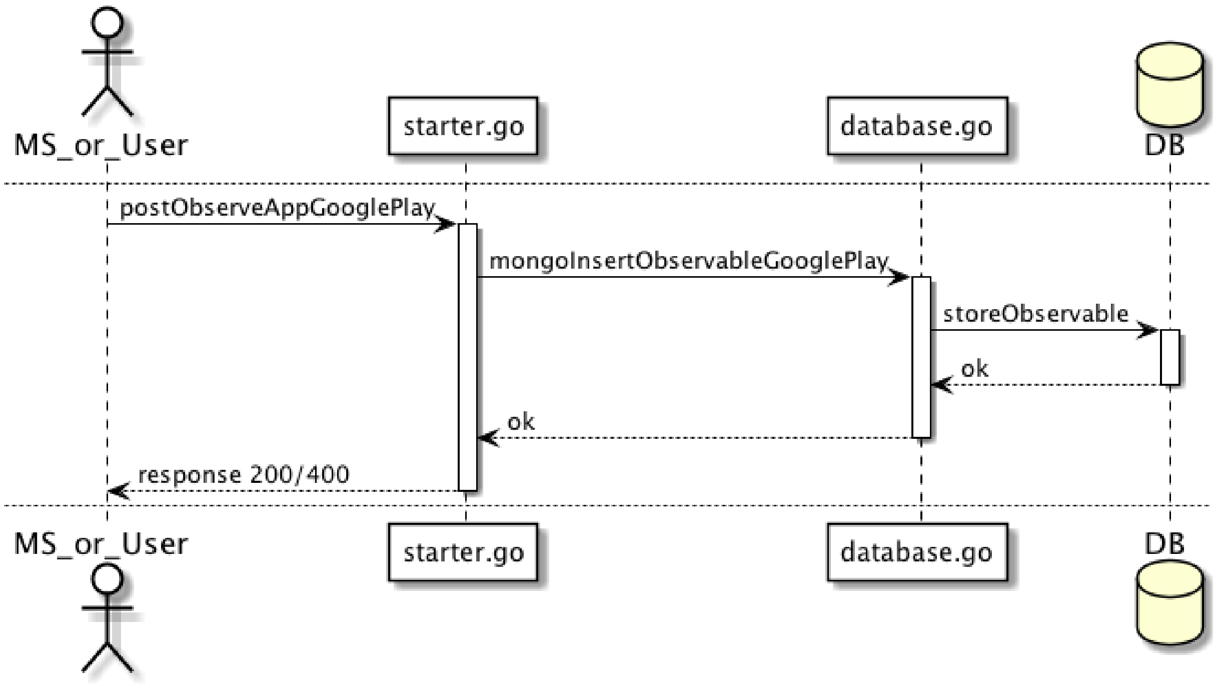
URL	https://openreq.esl.eng.it/eng/hitec/orchestration/twitter/observe/tweet/account/{account_name}/interval/{interval}/lang/{lang}
Method	POST
URL params	account_name : the name of the Twitter profile to observe, lang: the language the tweets should be written in
Data params	None

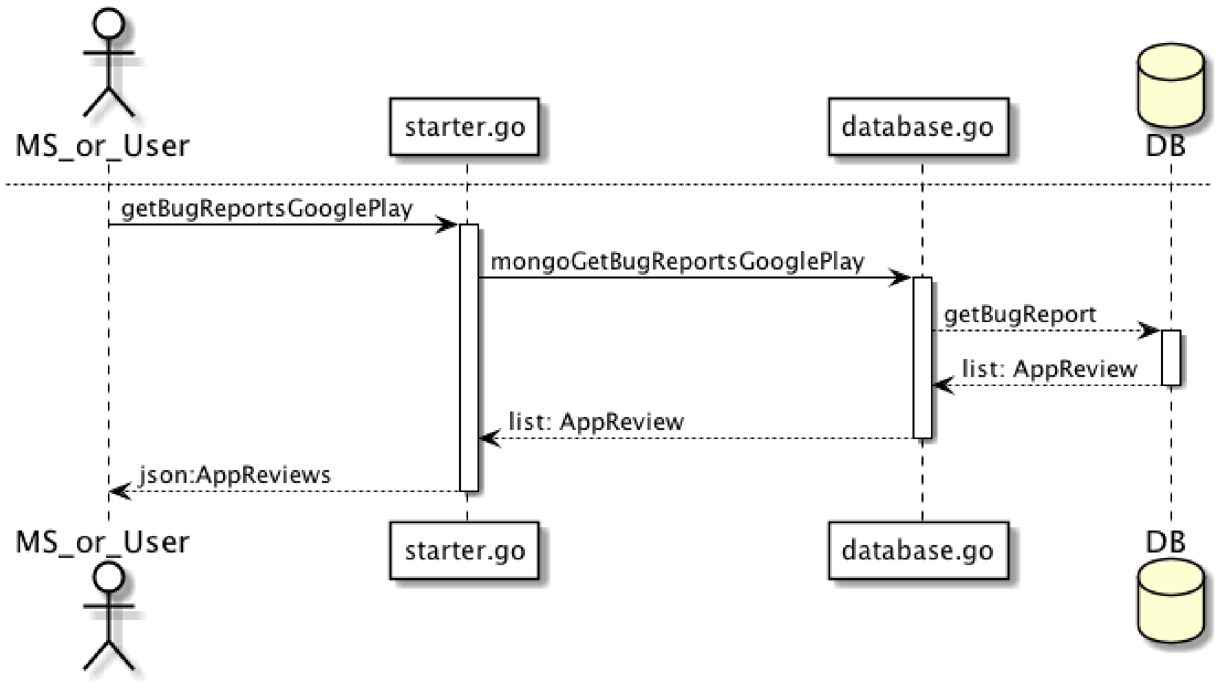
3.8. ri-storage-app

This microservice is the interface to the actual database. It persists JSON objects.

Sequence diagram







Example usage

URL	https://openreq.esl.eng.it/eng/hitec/repository/app/store/app-review/google-play/
Method	POST
URL params	None
Data params	{ “review_id”: "05df34353efd", "package_name":”com.myapp”, "author":”johndoe”, "date_posted":20180524, “rating”:5, “title”: “I like this app because…”, “body”: “it has really nice features”, “perma_link”: “https://….”, “cluster_is_feature_request”: false, “cluster_is_problem_report”: true }

URL	https://openreq.esl.eng.it/eng/hitec/repository/app/observe/app/google-play/package-name/{package_name}/interval{interval}
Method	POST

URL params	package_name: package name of the app we want to observe, Interval: specifies how often we want to observe it, e.g., daily
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/repository/app/non-existing/app-review/google-play/
Method	POST
URL params	None

Data params	[
	{
	“review_id”: "05df34353efd",
	"package_name":”com.myapp”,
	"author":”johndoe”,
	"date_posted":20180524,
	“rating”:5,
	“title”: “I like this app because…”,
	“body”: “it has really nice features”,
	“perma_link”: “https://….”,
	“cluster_is_feature_request”: false,
	“cluster_is_problem_report”: true
	}
	]

URL	https://openreq.esl.eng.it/eng/hitec/repository/app/unprocessed/app-review/google-play/package-name/{package_name}/limit/{limit}
Method	GET
URL params	package_name: package name of the app we want to get data, limit: how many app reviews we want to receive
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/repository/app/observable/google-play
Method	GET
URL params	None
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/repository/app/bug-report/google-play/package-name/{package_name}

Method	GET
URL params	package_name: package name of the app we want to get data
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/repository/app/feature-request/google-play/package-name/{package_name}
Method	GET
URL params	package_name: package name of the app we want to get data
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/repository/app/app-page/google-play/package-name/{package_name}/is-recent
Method	GET
URL params	package_name: package name of the app we want to get data
Data params	None

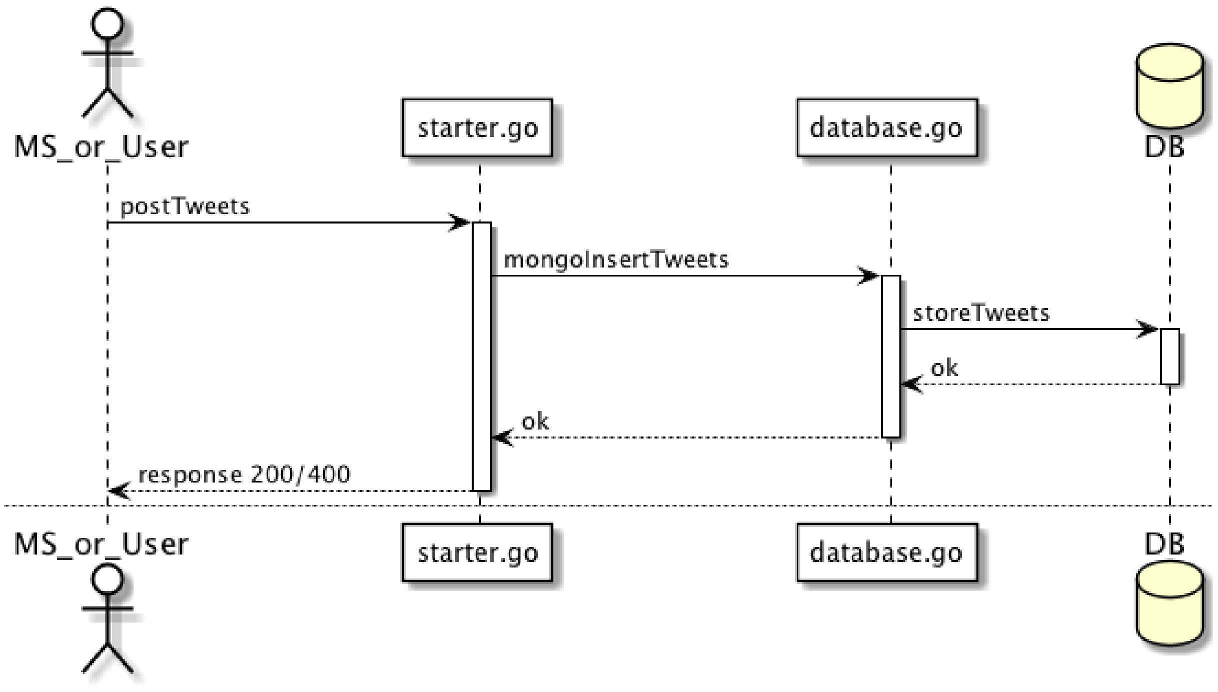
URL	https://openreq.esl.eng.it/eng/hitec/repository/app/app-review/google-play/package-name/{package_name}/is-recent"
Method	GET
URL params	package_name: package name of the app we want to get data
Data params	None

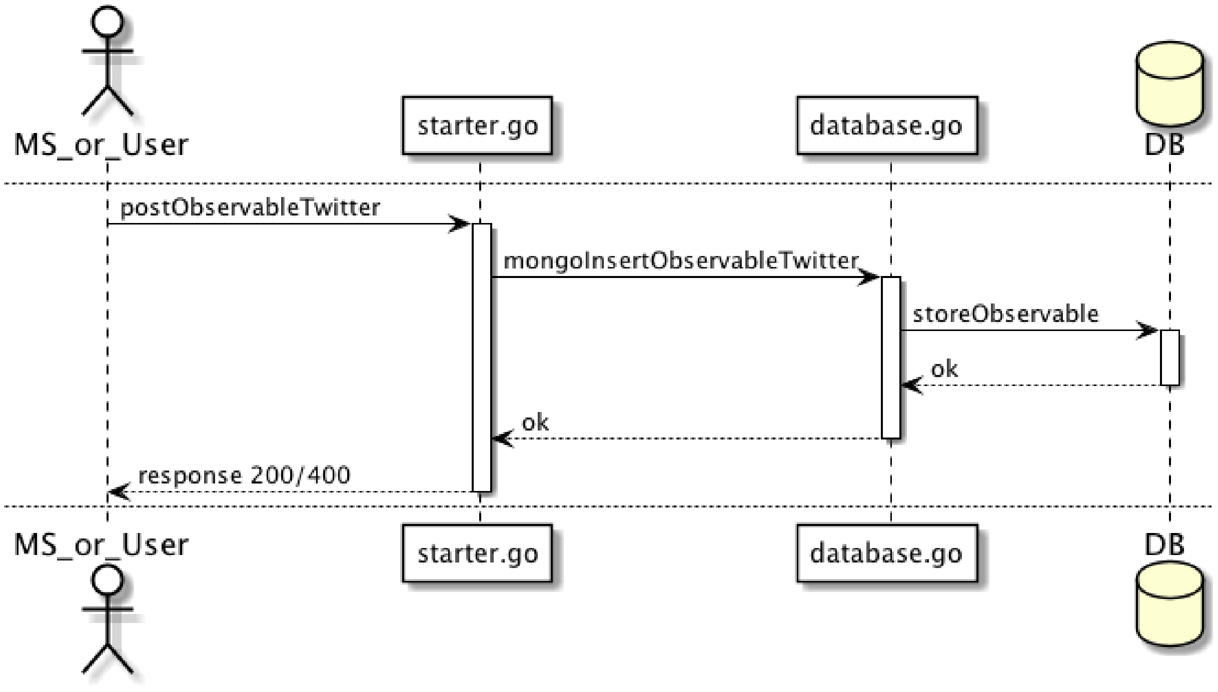
URL	https://openreq.esl.eng.it/eng/hitec/repository/app/bug-reports/google-play/package-name/{package_name}/is-recent
Method	GET
URL params	package_name: package name of the app we want to get data
Data params	None

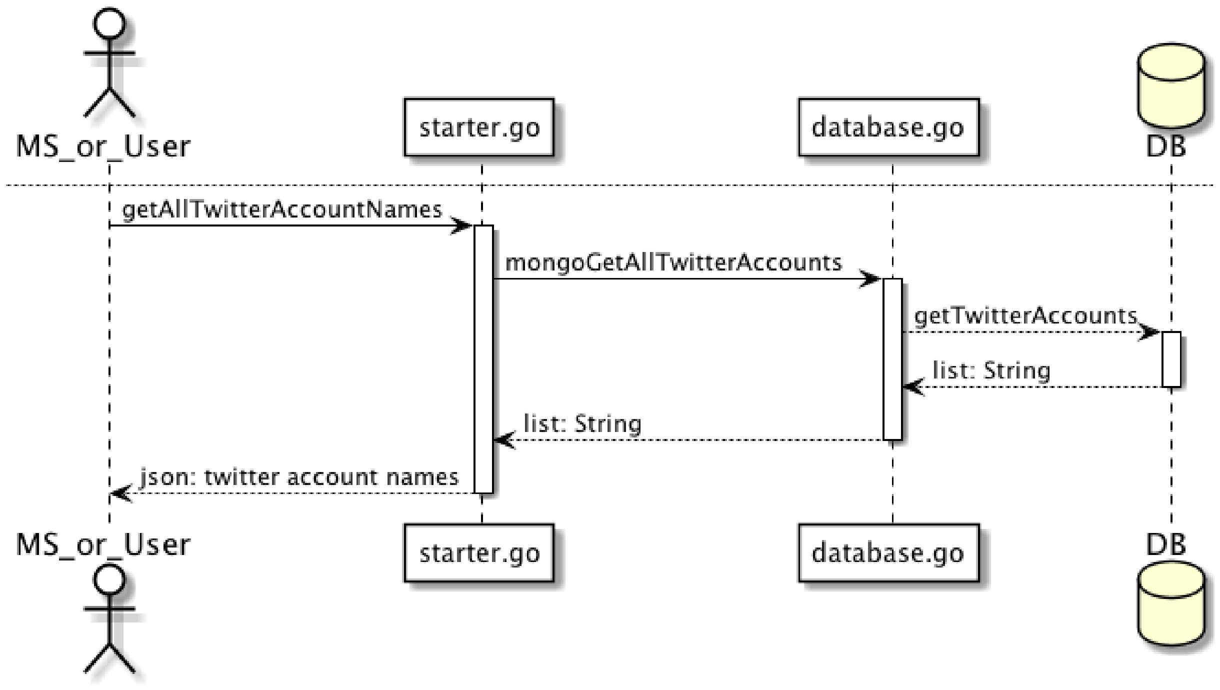
3.9.ri-storage-twitter

This microservice is the interface to the actual database. It persists all JSON objects that are related to Twitter.

Sequence diagram







Example usage

URL	https://openreq.esl.eng.it/eng/hitec/repository/twitter/store/tweet/
Method	POST
URL params	None
Data params	{ "created_at": 20180524, "favorite_count":0, "retweet_count":2, "full_text": “this is my tweets”, "status_id": “124323156”, "user_name": “janedoe”, "in_reply_to_screen_name": “johndoe”, "lang": “en”, "tweet_class": “problem_report” }

URL	https://openreq.esl.eng.it/eng/hitec/repository/twitter/store/observable/
Method	POST
URL params	None
Data params	{ "account_name": “my whatsapp name”,

	"interval":"daily, "Lang":"en" }

URL	https://openreq.esl.eng.it/hitec/repository/twitter/account_name/{account_name}/class/{tweet_class}
Method	GET
URL params	account_name: the account name of a twitter profile, tweet_class: e.g., problem_report
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/repository/twitter/account_name/{account_name}/all
Method	GET
URL params	account_name: the account name of a twitter profile
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/repository/twitter/account_name/all
Method	GET
URL params	None
Data params	None

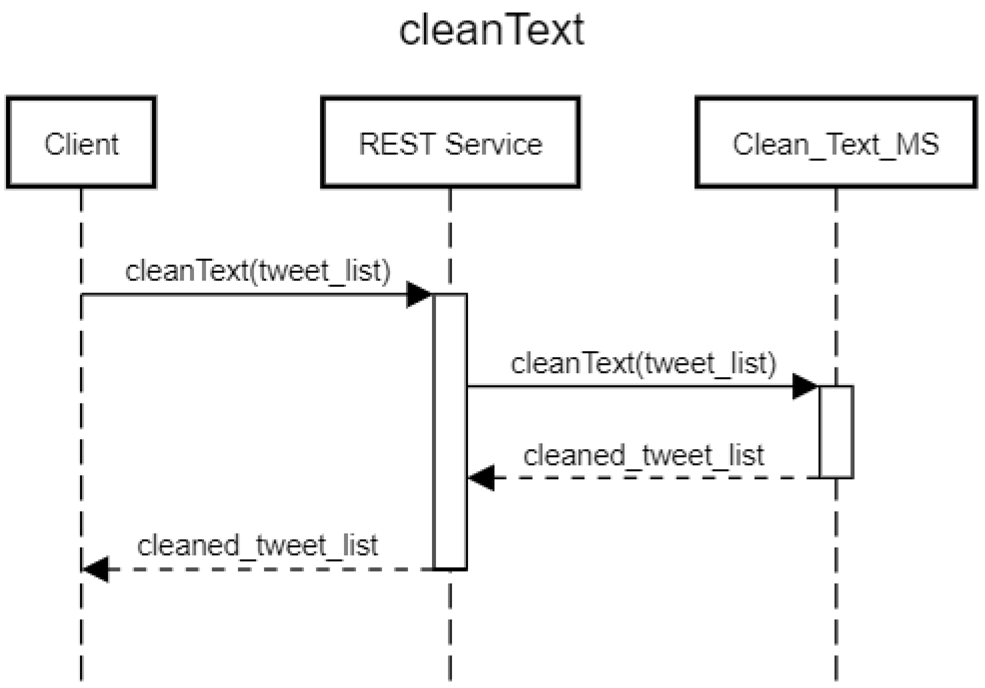
URL	https://openreq.esl.eng.it/eng/hitec/repository/twitter/reset/tweet
Method	GET
URL params	None
Data params	None

URL	https://openreq.esl.eng.it/eng/hitec/repository/twitter/observables
Method	GET
URL params	None
Data params	None

3.10. analytic-back-end-clean-text

This microservice cleans text. It takes as input a list of tweet messages and returns the same list cleaned—it removes special characters, punctuation signs, HTML tags, @username, stopwords and URLs. You can see Swagger documentation at /apidocs.

Sequence diagram



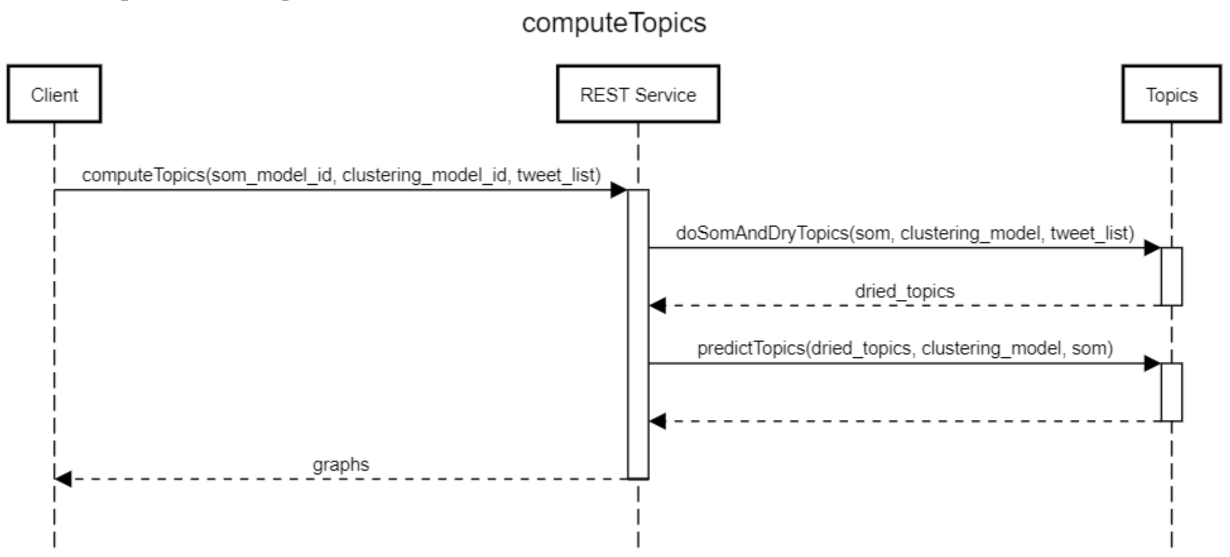
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/cleanText
Method	POST
URL params	None
Data params	[{"message":"Posso essere chiamato da un operatore 3"}, {"message":"Ci sono appena andato a un negozio e o faccio una promozione o niente! Voglio pagare quando chiamo e non vo https: / /t.co /lhRqzJPfpT"}]

3.11. analytic-back-end-compute-topics

This microservice extracts topics from a list of tweets. It takes as input the id of the models. For example word2vec for creating a vector representation of text, clustering and self-organizing maps to aggregate tweets messages, and the messages that should be examined. It returns a graph for each topic found.

Sequence diagram


computeTopics
Client
REST Service
Topics
computeTopics(som_model_id, clustering_model_id, tweet_list)
doSomAndDryTopics(som, clustering_model, tweet_list)
dried_topics
predictTopics(dried_topics, clustering_model, som)
graphs

Example usage

URL	https://openreq.esl.eng.it/eng/openReq/computeTopics
Method	POST
URL params	None
Data params	{"w2v_model_id":12345, "Som_model_id":67890, "codebook_cluster_model_id":24680, "tweets":[{"message":"Posso essere chiamato da un operatore 3"}, {"message":"Ci sono appena andato a un negozio e o faccio una promozione o niente! Voglio pagare quando chiamo e non vo "}]

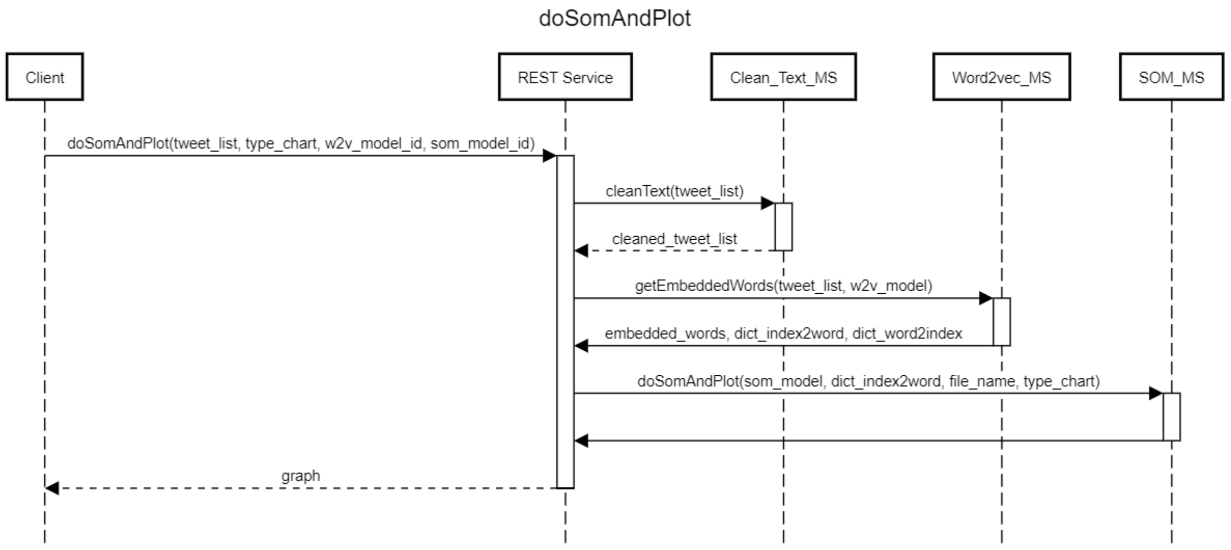
Response example	[
	  {
	    "directed": false,
	    "graph": {},
	    "links": [
	      {
	        "source": 0,
	        "target": 1
	      },
	      {

```
          "source": 0,
          "target": 4
        },
        {
          "source": 0,
          "target": 7
        }
      ],
      "multigraph": false,
      "nodes": [
        {
          "id": 0,
          "name": "offerta",
          "pos": [
            0.39845688850044275,
            0.01701041209436488
          ]
        },
        {
          "id": 1,
          "name": "promozione",
          "pos": [
            0.5467295096202749,
            -0.18447308286329653
          ]
        },
        {
          "id": 4,
          "name": "numero",
          "pos": [
            0.7412007341069454,
            0.06619163623027753
          ]
        },
        {
          "id": 7,
          "name": "abbonamento",
          "pos": [
            -0.007811836655060454,
            0.12406653407660848
          ]
        }
      ]
    },
    {
      "directed": false,
      "graph": {},
      "links": [
        {
          "source": 0,
        "target": 1
      },
      {
        "source": 0,
        "target": 2
      },
      {
        "source": 0,
        "target": 11
      }
    ],
    "multigraph": false,
    "nodes": [
      {
        "id": 0,
        "name": "wind",
        "pos": [
          0.1025021960790923,
          -0.00583486483102523
        ]
      },
      {
        "id": 1,
        "name": "operatore",
        "pos": [
          -0.2125952797941764,
          0.19447134425055665
        ]
      },
      {
        "id": 2,
        "name": "telefono",
        "pos": [
          -0.07288693121191979,
          -0.23931113221735195
        ]
      },
      {
        "id": 11,
        "name": "posso",
        "pos": [
          0.4806237910010984,
          -0.02734068831978047
        ]
      }
    ]
  }
]
```

3.12. analytic-back-end-do-som-and-plot

This microservice is useful to apply the Self Organizing Map (SOM) model (i.e., an artificial neural network (ANN) which produces a low-dimensional, discretized representation of the input space of the training samples, called a map) on a set of tweets and return a graph with Minimum Spanning Tree applied on the SOM codebooks. It takes as input the type of the chart: “d3” or “json”; the ids of the models, word2vec and SOM, and the list of the tweets or the URL of the CSV file (header ”messages”) containing the tweet messages.

Sequence diagram


doSomAndPlot
Client
REST Service
Clean_Text_MS
Word2vec_MS
SOM_MS
doSomAndPlot(tweet_list, type_chart, w2v_model_id, som_model_id)
cleanText(tweet_list)
cleaned_tweet_list
getEmbeddedWords(tweet_list, w2v_model)
embedded_words, dict_index2word, dict_word2index
doSomAndPlot(som_model, dict_index2word, file_name, type_chart)
graph

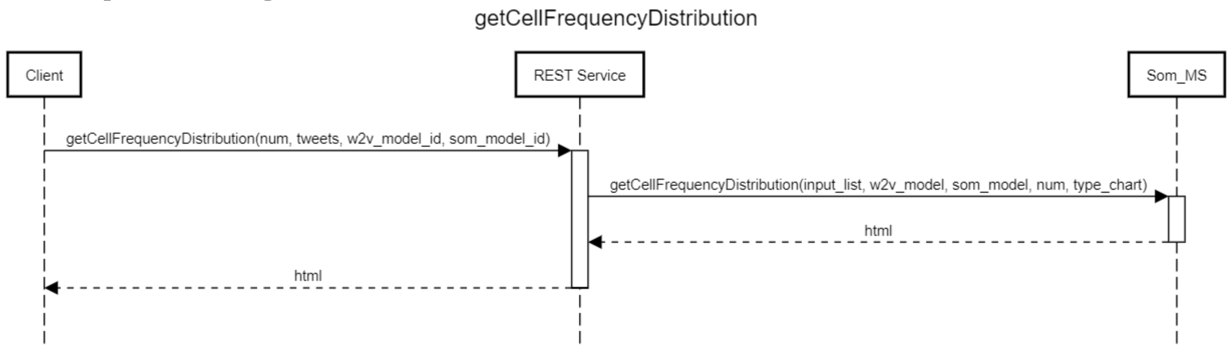
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/doSomAndPlot
Method	POST
URL params	None
Data params	{“type_chart”: "d3/json", "w2v_model_id":12345, "Som_model_id":67890, "url_input":"http://www.someurl.csv", "tweets":[{"messaggio":"Posso essere chiamato da un operatore 3"}, {"messaggio":"Ci sono appena andato a un negozio e o faccio una promozione o niente! Voglio pagare quando chiamo e non vo "}]

3.13. analytic-back-end-get-cell-frequency-distribution

This microservice shows the frequency of each cell of the SOM. It takes as input the type of chart, the ids of the models (word2vec and SOM), the maximum number of cells, and the tweet messages or the URL of the CSV (header ”messages”) containing the tweet messages. It returns the HTML containing the graph.

Sequence diagram


getCellFrequencyDistribution
Client
REST Service
Som_MS
getCellFrequencyDistribution(num, tweets, w2v_model_id, som_model_id)
getCellFrequencyDistribution(input_list, w2v_model, som_model, num, type_chart)
html
html

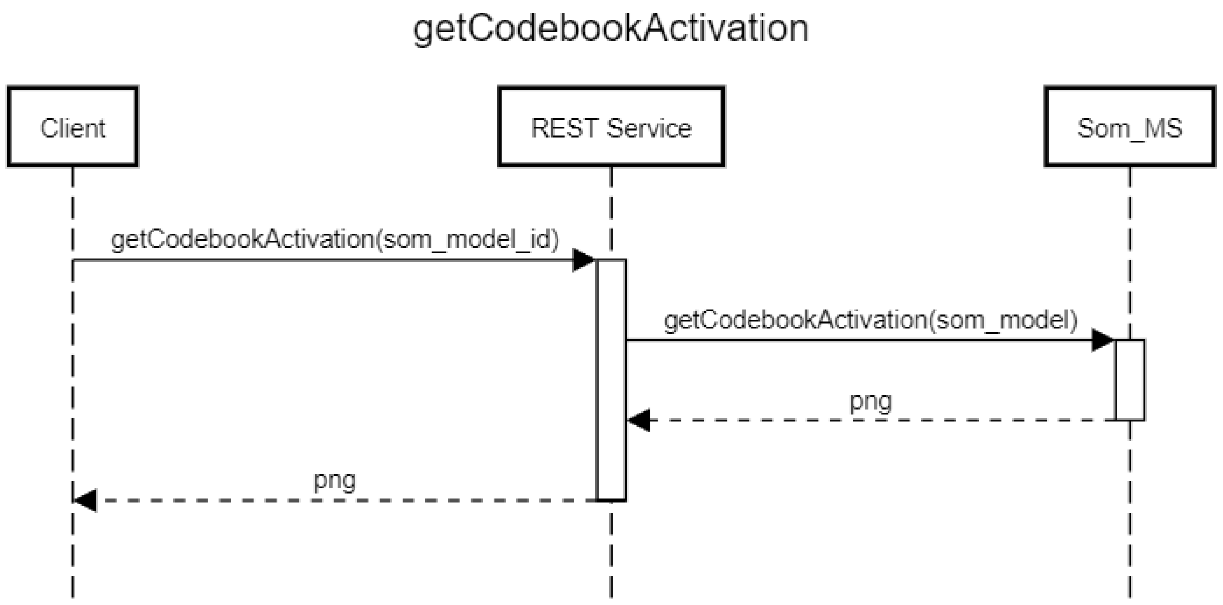
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/getCellFrequencyDistribution
Method	POST
URL params	None
Data params	{“type_chart”: "bubble/bar", "w2v_model_id":12345, "Som_model_id":67890, "Num":30， "url_input":"http://www.someurl.csv" "tweets":[{"message":"Posso essere chiamato da un operatore 3"}, {"message":"Ci sono appena andato a un negozio e o faccio una promozione o niente! Voglio pagare quando chiamo e non vo "}]

3.14. analytic-back-end-get-codebook-activation

This microservice shows the number of activations of codebooks during the SOM training. It takes as input the id of the SOM model and returns an image.

Sequence diagram


getCodebookActivation
Client
REST Service
Som_MS
getCodebookActivation(som_model_id)
getCodebookActivation(som_model)
png
png

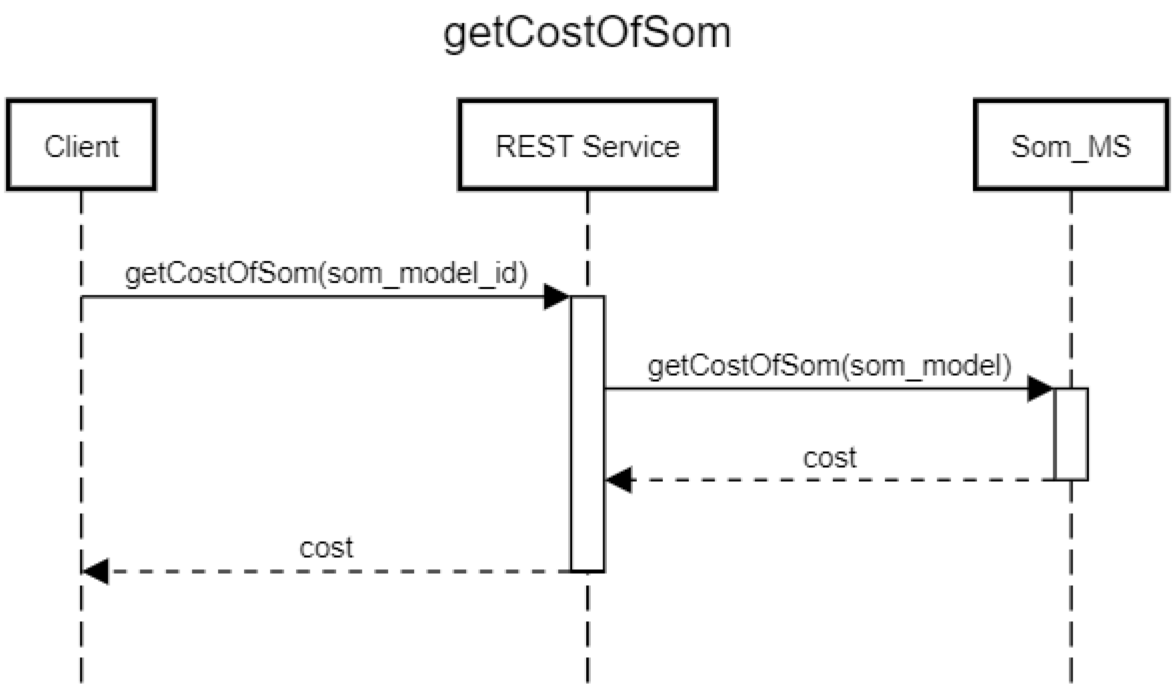
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/getCodebookActivation
Method	GET
URL params	som_model_id
Data params	None

3.15. analytic-back-end-get-cost-of-som

This microservice shows the costs of the trained SOM. It takes as input the id of the SOM and returns the cost of the trained model.

Sequence diagram



Example usage

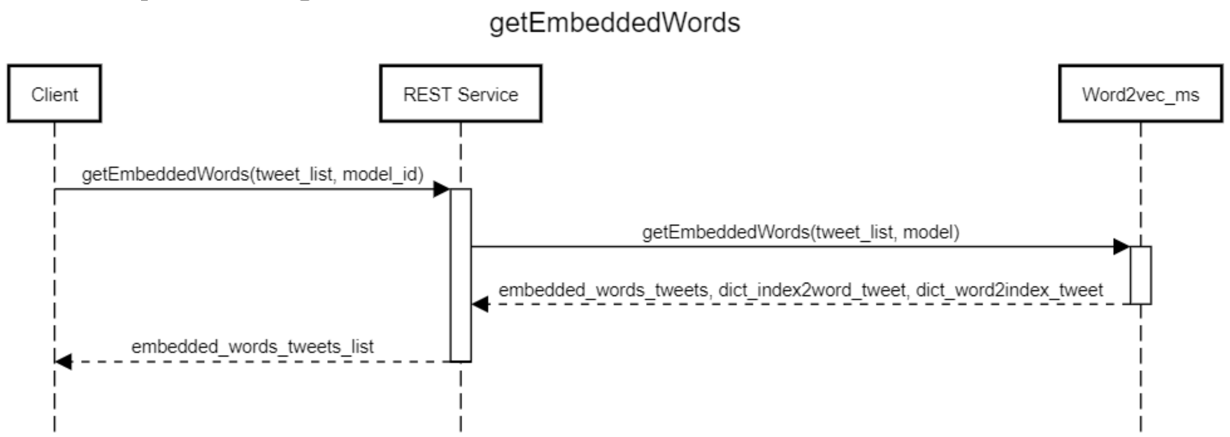
URL	https://openreq.esl.eng.it/eng/openReq/getCostOfSom
Method	POST
URL params	None
Data params	{"som_model_id":67890}

3.16. analytic-back-end-get-embedded-words

This microservice is useful to apply the word2vec model (a vector based model to represent text) to the tweets in input.

The microservice takes as input the id of the model word2vec and the tweet messages that we want to embed and return the n dimensional vectors, in this case the dimensions are 100. As result each word of the message is converted into a 100 dimensional vector.

Sequence diagram


getEmbeddedWords
Client
REST Service
Word2vec_ms
getEmbeddedWords(tweet_list, model_id)
getEmbeddedWords(tweet_list, model)
embedded_words_tweets, dict_index2word_tweet, dict_word2index_tweet
embedded_words_tweets_list

Example usage

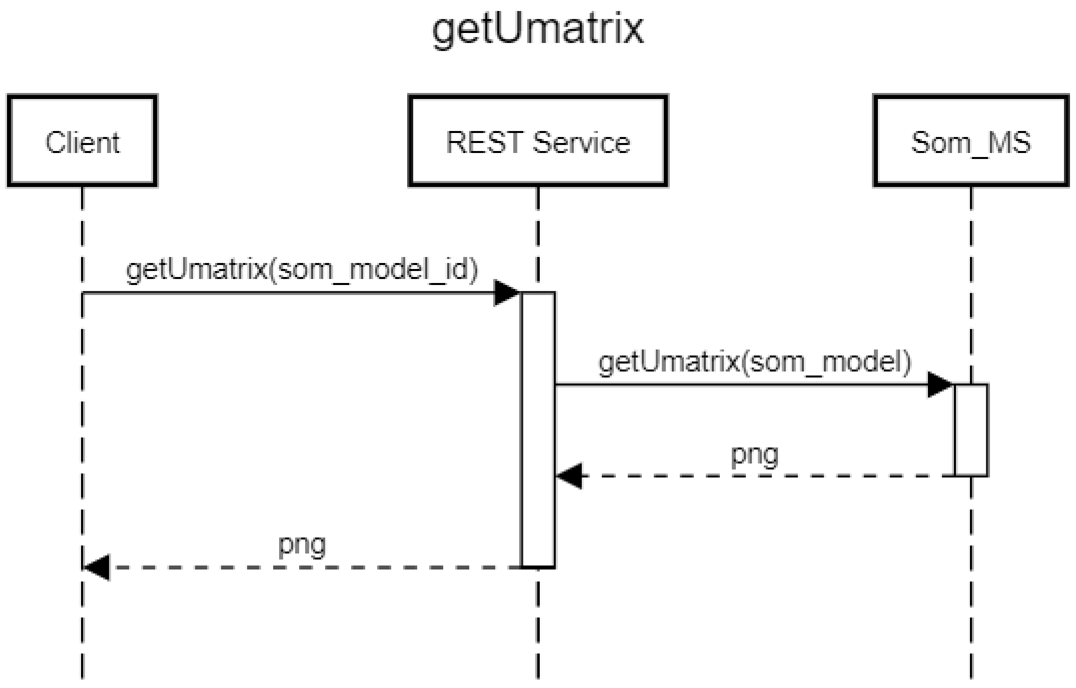
URL	https://openreq.esl.eng.it/eng/openReq/getEmbeddedWords
Method	POST
URL params	None
Data params	{"w2v_model_id":12345, "tweets":[{"message":"Posso essere chiamato da un operatore 3"}, {"message":"Ci sono appena andato a un negozio e o faccio una promozione o niente! Voglio pagare quando chiamo e non vo "}]

3.17. analytic-back-end-get-umatrix

This microservice calculates the umatrix (i.e., a visual representation of a SOM) of a specific trained SOM.

The microservice takes as in input the id of the SOM model and returns the image of the Umatrix.

Sequence diagram



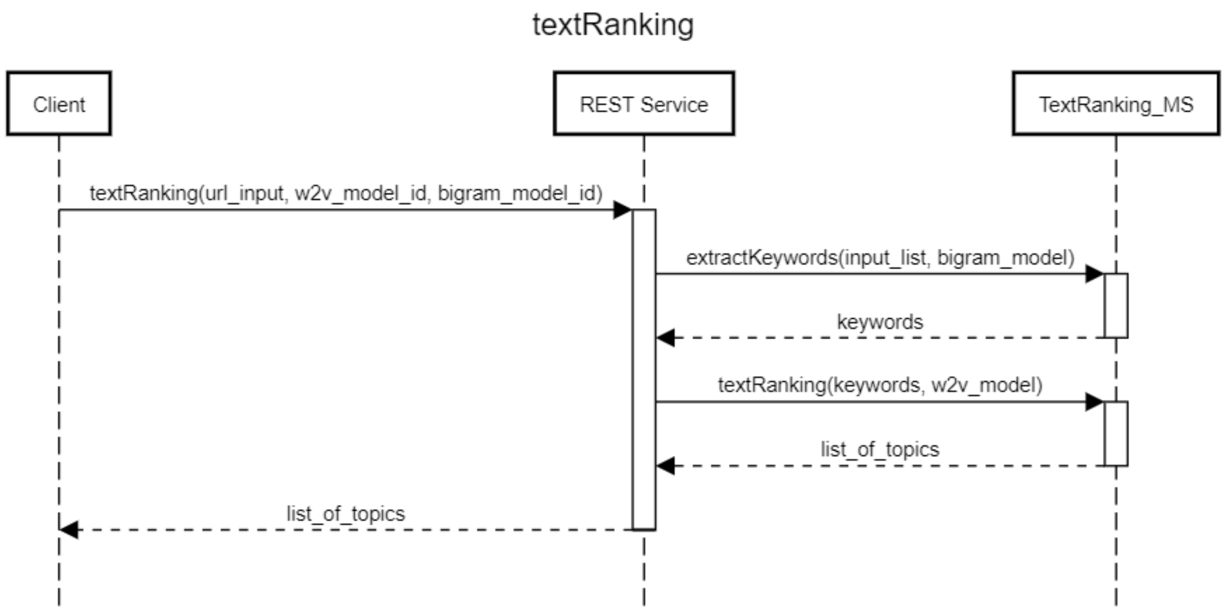
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/getUmatrix
Method	POST
URL params	None
Data params	{"som_model_id":67890}

3.18. analytic-back-end-text-ranking

This microservice performs text ranking on a set of tweets. It takes as input a list of tweets or a CSV (header ”messages”) with the tweets and the id of the word2vec and bigram models . It returns lists of words that represent topics.

Sequence diagram


textRanking
Client
REST Service
TextRanking_MS
textRanking(url_input, w2v_model_id, bigram_model_id)
extractKeywords(input_list, bigram_model)
keywords
textRanking(keywords, w2v_model)
list_of_topics
list_of_topics

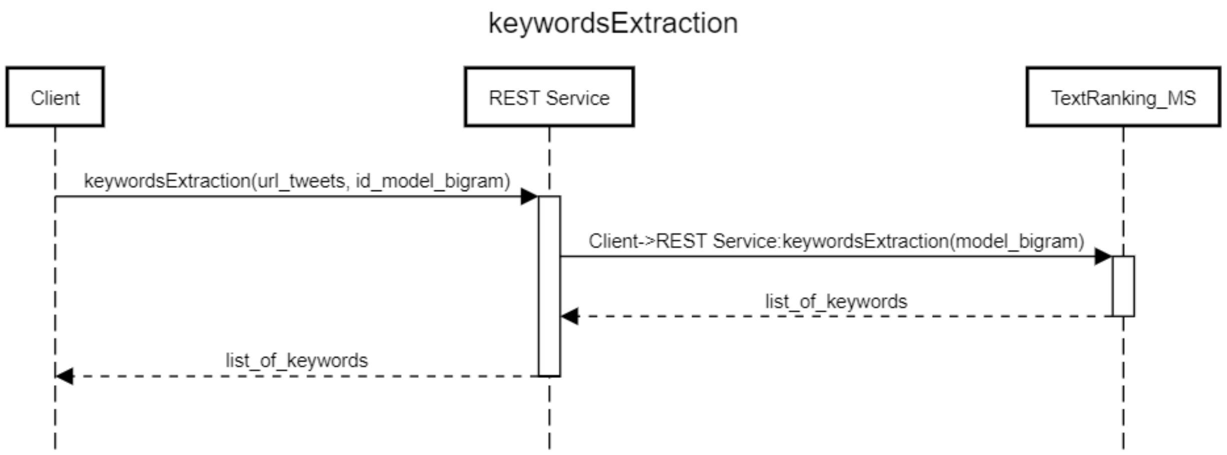
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/textRanking
Method	POST
URL params	None
Data params	{"url_input":"http://www.someurl.csv", "tweets":[{"message":"Posso essere chiamato da un operatore 3"}, {"message":"Ci sono appena andato a un negozio e o faccio una promozione o niente! Voglio pagare quando chiamo e non vo "}], "w2v_model_id":12345, “bigram_model_id”,: 24680,

3.19. analytic-back-end-keywords-extraction

This microservice extracts keywords from a text. It takes as in input the list of tweet messages or the URL of a CSV file (header ”messages”) containing a list of tweet messages and the id of the bigram model previously trained calling the API *trainNgram*.

Sequence diagram


keywordsExtraction
Client
REST Service
TextRanking_MS
keywordsExtraction(url_tweets, id_model_bigram)
Client->REST Service:keywordsExtraction(model_bigram)
list_of_keywords
list_of_keywords

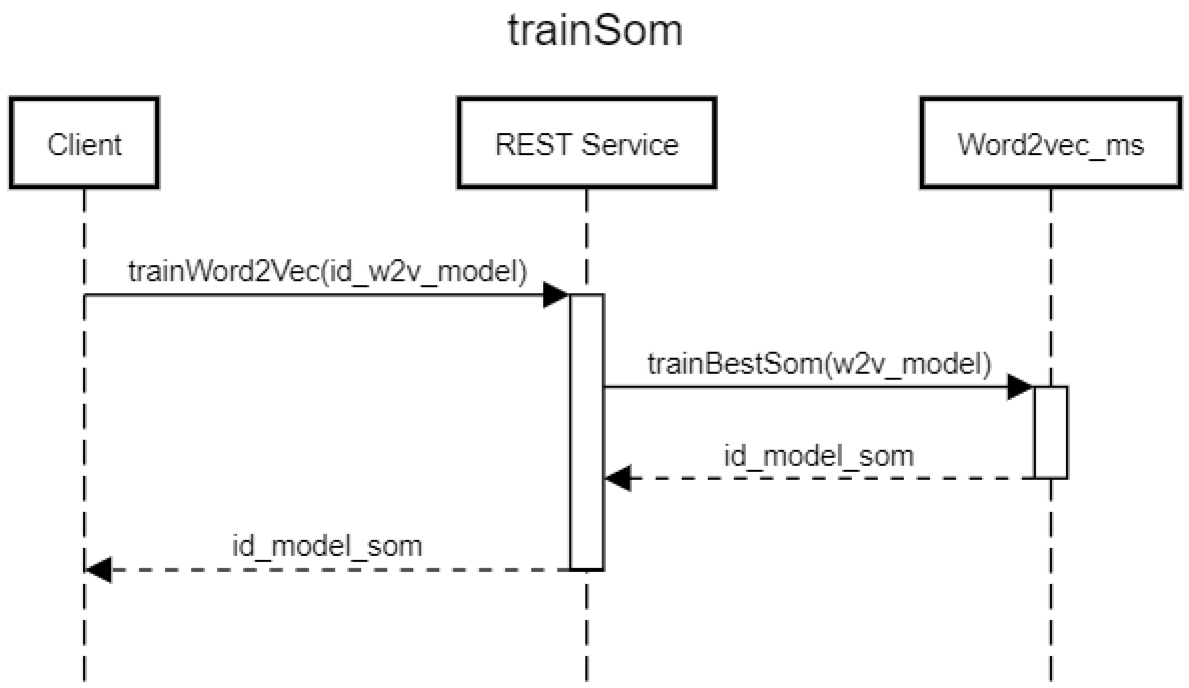
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/keywordsExtraction
Method	POST
URL params	None
Data params	{"url_input":"http://www.someurl.csv", "tweets":[{"message":"Posso essere chiamato da un operatore 3"}, {"message":"Ci sono appena andato a un negozio e o faccio una promozione o niente! Voglio pagare quando chiamo e non vo "}], "bigram_model_id":123456}

3.20. analytic-back-end-train-som

This microservice trains a SOM. The microservice takes as input the id of the word2vec model previously trained calling the API *trainWord2vec* and returns the id of the trained model.

Sequence diagram



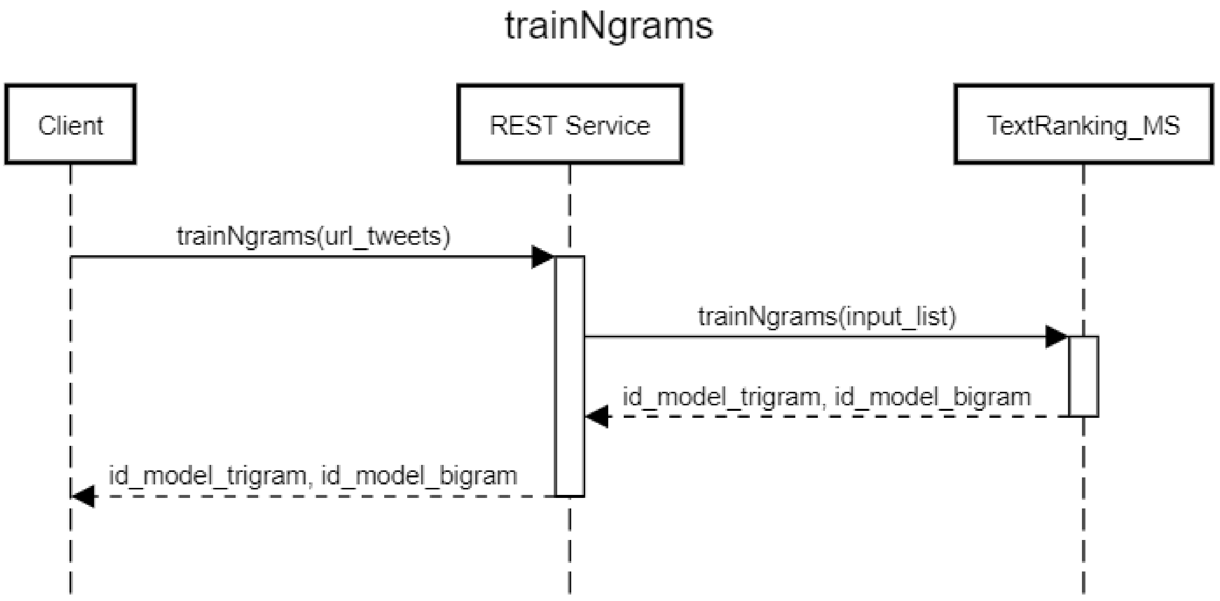
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/trainSom
Method	POST
URL params	None
Data params	{"w2v_model_id":123456}

3.21. analytic-back-end-train-ngrams

This microservice trains the bigram needed to extract keywords from tweet messages. It takes as input the URL of the CSV file (header “messages”) with the tweet messages or a list of tweets messages and returns an id of the trained model.

Sequence diagram



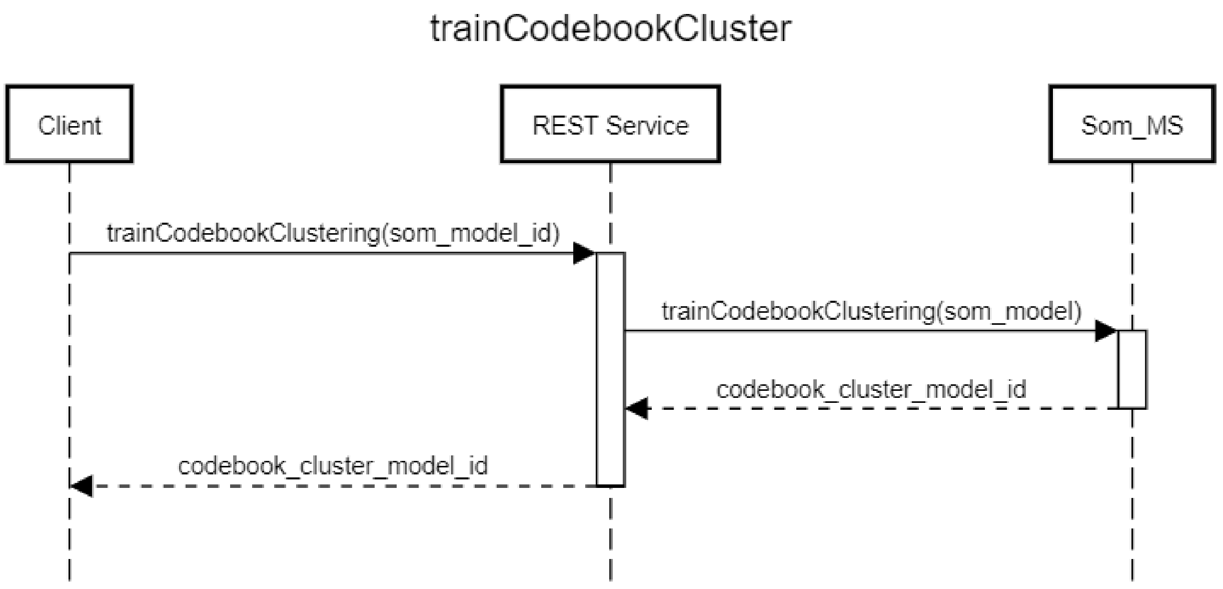
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/trainNgrams
Method	POST
URL params	None
Data params	{"url_input":"http://www.someurl.csv", "tweets":[{"message":"Posso essere chiamato da un operatore 3"}, {"message":"Ci sono appena andato a un negozio e o faccio una promozione o niente! Voglio pagare quando chiamo e non vo "}]}

3.22. analytic-back-end-train-codebook-cluster

This microservice trains the clustering model applied on the codebooks of the SOM, this is the model used to extract the topics from the tweet messages and given as parameter to the API *computeTopics*. It takes as input the id of the SOM and returns the id of the trained model.

Sequence diagram


trainCodebookCluster
Client
REST Service
Som_MS
trainCodebookClustering(som_model_id)
trainCodebookClustering(som_model)
codebook_cluster_model_id
codebook_cluster_model_id

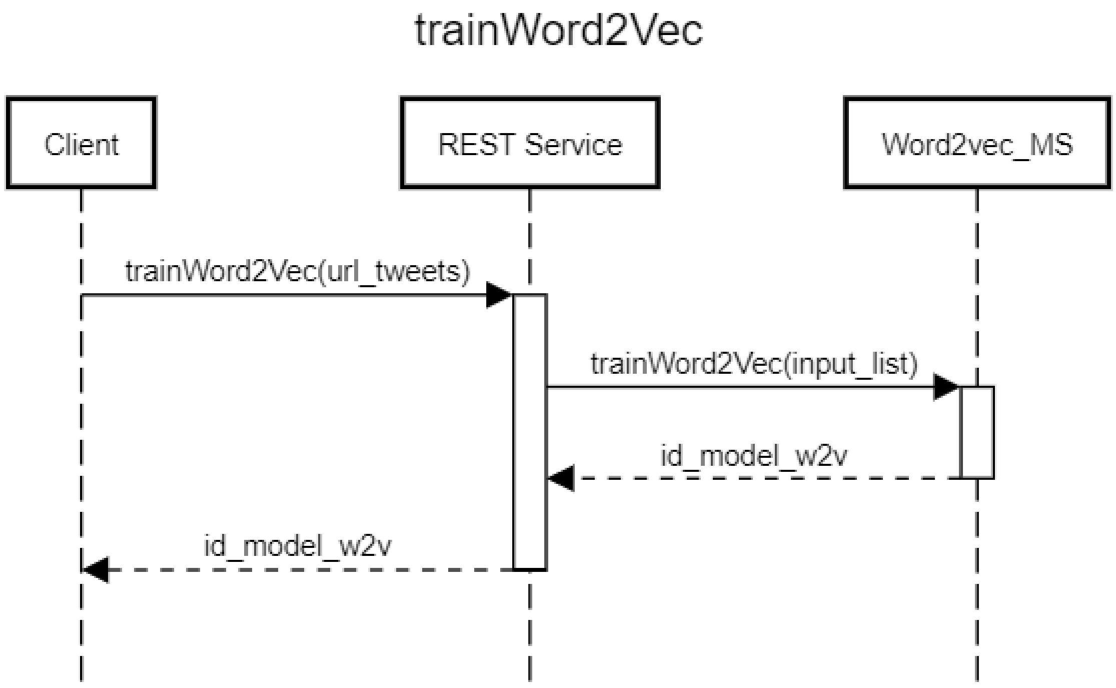
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/trainCodebookCluster
Method	POST
URL params	None
Data params	{"som_model_id":123456}

3.23. analytic-back-end-train-word-to-vec

This microservice trains a Word2vec model. It takes as input an URL of the CSV file (header ”messages”) containing the tweet messages or the list of tweet messages and returns the id of the trained model.

Sequence diagram



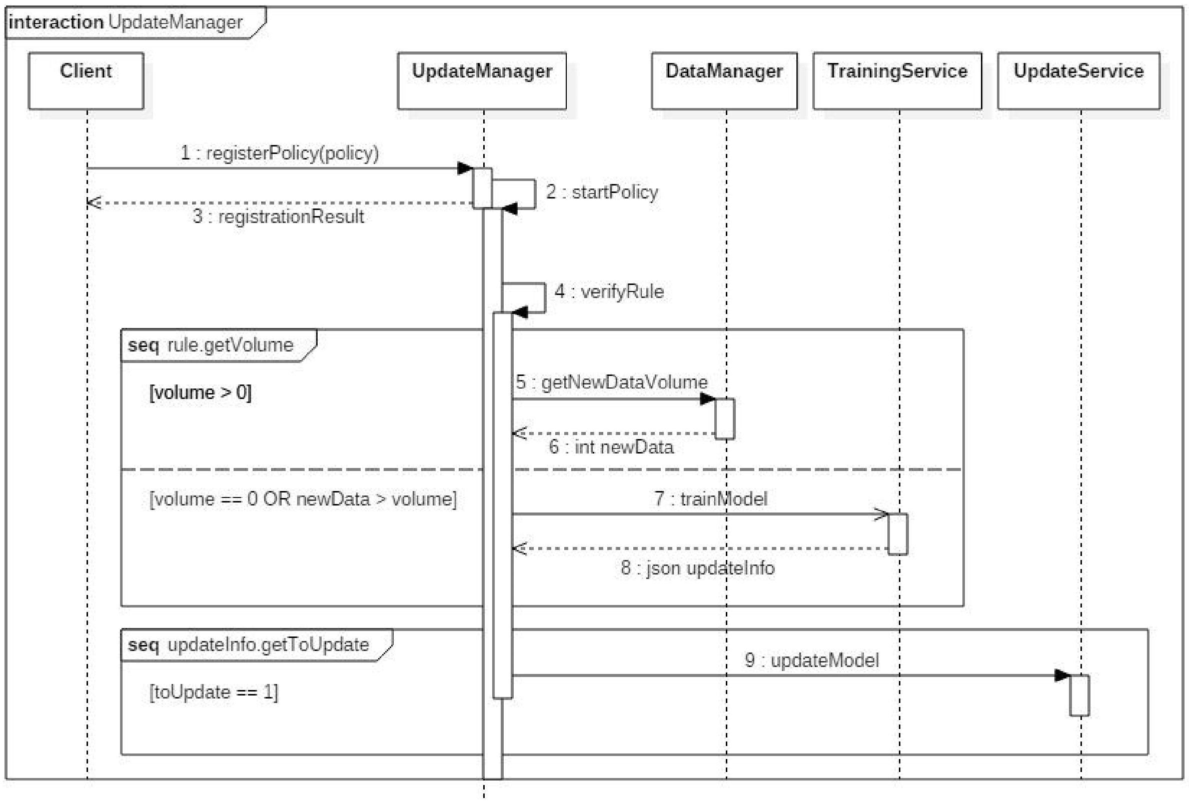
Example usage

URL	https://openreq.esl.eng.it/eng/openReq/trainWord2vec
Method	POST
URL params	None
Data params	{"url_input":"http://www.someurl.csv", "tweets":[{"message":"Posso essere chiamato da un operatore 3"}, {"message":"Ci sono appena andato a un negozio e o faccio una promozione o niente! Voglio pagare quando chiamo e non vo "}]}

3.24. analytic-back-end-update-manager

This microservice auto-updates other microservices. It takes as input a policy containing rules. Rules can be time- or time- and volume-dependent (e.g., based on the amount of new training examples available). When a rule is matched, the microservice requests a training microservice and, if required, updates the model.

Sequence diagram



Example usage

URL	https://openreq.esl.eng.it/eng/openReq/updateManager/registration
Method	POST
URL params	None
Data params	Policy: { "addressNewData": "dataManager/getNewData", "addressTraining": "trainingService/getTraining", "addressUpdate": "updateService/putNewModel", "ownerId": "eng", "policyId": "service001", "rules": [

```
    {
      "period": 6,
      "timeUnit": "HOURS",
      "startDateTime": "2018-05-24T00:00:00+02:00[Europe/Rome]",
      "volume": 1000
    }
  {
      "period": 1,
      "timeUnit": "DAYS",
      "startDateTime": "2018-05-24T00:00:00+02:00[Europe/Rome]",
    }
  ]
}
```

4. Challenges and solutions

In this section, we describe the challenges we encountered and the solutions selected during the development and testing phase of the microservices presented in Section 3.

4.1.Twitter API limitation

Challenges: We crawl data from Twitter using the official API which is limited by a specific number of requests in a certain time window[1]. Further, the API comes in different access levels and pricing: *standard*, *premium*, and *enterprise*—each of them allowing different search options, such as crawling data of the last seven days, 30 days, or the full archive[2]. As we cannot provide paid Twitter search API tokens to the public, we are limited to the *standard* search API. In future versions of the microservice, we will add the possibility add a custom API access tokens to allow *premium* and *enterprise* searches.

Solutions: We mitigate the API limit by enabling our microservices to crawl data in intervals (e.g., on a daily basis). Although we cannot go back in time, we are able to gather all recent data from the point in time in which the crawler is configured. In addition, we consider the API rate limit and pause the crawler in case it gets blocked; this solution allows us to have a continuous crawling process.

4.2.Creation of a gold standard of Tweet messages

Challenges: The creation of a gold standard of tweet messages (i.e., messages annotated with their correct class) used for classifying them as, for example, problem report or irrelevant is challenging because of the vast number of messages available. There is a trade-off between creating a representative sample (not domain-specific) and focusing on a specific domain.

Moreover, the gold standard is created through human annotations. This results in several challenges such as getting data annotated in the first place and having high-quality annotation (i.e., correct annotation).

Solutions: To address the challenge of getting annotated data, we employed crowed source annotation using the platform *figure eight* (formerly CrowdFlower)[3]. On this platform, people can annotate data for monetary compensation. Using such platform can lead to incorrectly annotated data due to the inexperience of the platform users. First, we mitigate this threat by providing written, peer-reviewed, and validated annotation guidelines explaining how to perform the annotation task, including several examples. Second, we asked questions to check if the crowd understood the annotation guide, and discarded annotators who did not. Third, each message is annotated by at least two persons, if they do not agree on the annotation it will be shown to a third person. We only consider messages having an agreement by at least two persons.

[1] https://developer.twitter.com/en/docs/basics/rate-limiting.html
[2] https://developer.twitter.com/en/pricing.html
[3] https://www.figure-eight.com/

4.3. Collection of implicit feedback

Backend Logging

Challenges: Logging microservice interactions, including those without human intervention, requires each microservice to implement an additional logging call to each microservice. The effort can possibly exceed the possibilities microservice maintainers, in particular when third parties want to extend OpenReq with additional microservices.

Solutions: The backend logging will be implemented within the webserver application (i.e., Apache HTTP Server) that dispatches the calls to all microservices hosted on the OpenReq infrastructure. By implementing the logger as an Apache module, we can capture all requests and responses without any effort by the microservice maintainers.

Frontend Logging

Challenges: The frontend interactions are logged by a custom library. The main challenges are compatibility, distribution, and reduction of the integration effort.

Solutions: To address the challenge of the compatibility, the logging library is implemented in Vanilla JavaScript and thus has no dependencies on other libraries. The library is distributed from within the logging microservice itself and therefore always available and up=to-date. The integration of the library only to import the script within the HTML.

5. Conclusion

In this document we positioned the Requirement Intelligence Engine and its services in the overall OpenReq architecture. Moreover, we defined each microservice behavior, interaction, and usage. Finally, we delineated the challenges we faced and how we addressed them.

For version 2 of the Requirements Intelligence Engine, we will improve the efficiency of the existing services. To that end, we will i) use the interaction information collected by the microservices, ii) devise new policies for the update manager component based on the experience and feedback gathered during the trials.

Based on the interviews with the case companies (see D1.2), we consider adapting the existing microservices to other similar sources of relevant data for requirements engineering, such as customers’ tickets and developers’ issues.