Dario LORENZONI

Raffele NOCERINO

Marco PETTORALI

Riccardo XEFRAJ

newspaper

Task 2

Index

[Description of the system 4](#_Toc32496404)

[System requirements 4](#_Toc32496405)

[Functional requirements: 4](#_Toc32496406)

[Non-functional requirements: 5](#_Toc32496407)

[Main actors 6](#_Toc32496408)

[Use Cases 6](#_Toc32496409)

[Analysis Class Diagram 7](#_Toc32496410)

[Data model 8](#_Toc32496411)

[Classes 10](#_Toc32496412)

[Client’s classes: 10](#_Toc32496413)

[Server’s classes: 11](#_Toc32496414)

[Message’s classes: 13](#_Toc32496415)

[UML Diagram 15](#_Toc32496416)

[Server UML Diagram 15](#_Toc32496417)

[Client UML Diagram 16](#_Toc32496419)

[Application’s architecture diagram 17](#_Toc32496420)

[Scraping 18](#_Toc32496421)

[Scraper class: 18](#_Toc32496422)

[Scraping activity: 18](#_Toc32496423)

[Retrieving information from websites 19](#_Toc32496424)

[Workflow of scraping process 19](#_Toc32496425)

[Text Analyzer 19](#_Toc32496426)

[Mongo DB 20](#_Toc32496427)

[Server Worker 25](#_Toc32496428)

[ServerMain Class: 25](#_Toc32496429)

[ServerRequestListener Class: 26](#_Toc32496430)

[ServerWorker Class: 26](#_Toc32496431)

[ServerAsynchronousWorker Class: 26](#_Toc32496432)

[User manual 27](#_Toc32496433)

[Login 27](#_Toc32496434)

[Sign-in 28](#_Toc32496435)

[Working with the application 29](#_Toc32496436)

[Administrator case 29](#_Toc32496437)

[User case 31](#_Toc32496438)

## Description of the system

The main purpose of this application is to provide to the users an overview of the trending topics using statistic reports based on the articles published by the most famous Italian online newspapers. After the login, users can watch which are the most used words among the articles written in the last seven days, that are collected in a ranking ordered by the importance of the keyword (that depends on how many times a certain word appears in each article and on how many articles talk about that specific word). The users can also view the statistics of the trending keywords observing a Pie Chart in which they can find the first 25 important trending keywords (ordered by percentage). The system also provides a search engine for retrieving articles, in which the user can perform queries based on a specific keyword and on some filters (category of the article, author, newspaper from which it comes, geographical region related to the article). The system keeps track of the searches made by the user and, using that history of searches, shows him/her a customized list of suggested articles. An administrator can manage the users signed into the service, setting the rate at which articles are collected and analyzed and can force an asynchronous update of the articles stored into the database.

## System requirements

### Functional requirements:

* The user can sign in to the service filling a form;
* The user can login to the service after he/she signed in;
* The system shows the list of the trending keywords, ordered by their importance;
* The system shows a pie chart which represents the 25 most important keywords ordered by percentage of usage.
* The system provides a search engine to query the database of the articles by keyword;
* Query can involve the following filters: category of the article, author, online newspaper, geographical region;
* When the user moves the cursor over an article, the system shows the main keywords of the article and their occurrences;
* If the user clicks on an article, the system open the entire online article into the browser.
* The system shows to the user a customized list of suggested articles, based on its last searches;
* The administrator can login to the service;
* The administrator can view the list of all users;
* The administrator can set the period at which articles are download from the web and analyzed;
* The administrator can force in any moment a collection of new articles from the web and the subsequent calculation of the statistics;

### Non-functional requirements:

* The service must be reachable by the users at every time, independently of the state of the data to which he/she has access;
* Collection of articles from the online newspapers must not be invasive and cause them no harm;
* Users must experience a fast response of the system when they make queries in the search engine;
* Every time an error occurs, the system shows an error message with a short description of the error;

## Main actors

* User
* Administrator
* Time

## Use Cases

Immagine che contiene testo, mappa

Descrizione generata automaticamente

## Analysis Class Diagram

Immagine che contiene screenshot

Descrizione generata automaticamente

## Data model

In our application we have used a document database, called MongoDB, that allowed us to obtain some important advantages. We have created a schema-less database obtaining high flexibility in storing information. Another important advantage that we have obtained, differently from a SQL db, is the possibility to store nested documents or arrays inside a single document. For example:

{

"\_id":"5e41c6de31fb29652f56d976",

"userID":"rafnoc",

"linkArticle":"http://www.ansa.it/lazio/notizie/2020/01/24/omicidio-sacchi-giudizio-immediato-per-sei-anche-per-anastasia-\_880f2403-28a3-4953-aa1e-942e3248c4f5.html",

"dateRead":"2020-02-10T21:10:54.558Z",

"filters":{"Keyword":"euro","Newspaper":"ANSA"}

}

This is an example of storing an object within another object.

In our database, called Article, we have created 3 collections of documents :

* **Articles**: in that collection we have stored the list of all the articles scraped from the 4 most important newspaper’s website. The structure of each article’s document is:

Immagine che contiene screenshot

Descrizione generata automaticamente

As we can see from the picture, in the article’s document we have stored an array of objects. That array allowed us to store inside each article its correspondent Text Analysis, which contains for each word of the article its related number of occurrences.

* **Search**: we have used the search collection in order to implement the functionality of recommended articles for each user. Every time a user clicks on an article, the application stores into the database some information that will be used in future to compute the recommended articles for the user.

Immagine che contiene screenshot

Descrizione generata automaticamente

Here inside the collection we have the field “filter” that is a nested document which contains all the filters used by the user in order to perform the search related to the viewed article.

* **User:** that collection is used to store all the information about the user registered to the application.

**Immagine che contiene screenshot

Descrizione generata automaticamente**

## Classes

### Client’s classes:

|  |  |  |
| --- | --- | --- |
|  | |  |
| Front-end | **AdminPaneGUI** | That class implements the admin control panel in which the administrator can manage the settings of the application. |
| **SignupPaneGUI** | That class implements the panel used by the users to register themselves to the application. |
| **LoginPaneGUI** | That class implements the panel used to implement the login procedure. |
| **MainPaneGUI** | That class implements the main panel for a normal user. Here the user can perform a search, view trending keywords and recommended articles. |
| **ArticlesOverviewTable** | That class implements the table which contains the Text Analysis of the related article. |
| **ArticlesTable** | That class implements the table which contains a list of articles. (recommended or result articles). |
| **SingleWordAnalysis** | That class represents a single result of an article’s Text Analysis. Contains a word and its related number of occurrences. |
| **Trend** | That class represents the basic element of TrendingKeywords table. Contains a trending keyword and its related Percentage of usage. |
|  | **TrendingKeywordsTable** | That class implements the table used to show the list of the most important trending keywords computed by the application. |
| **UserOverviewTable** | That class implements the table showed into the admin control panel containing all the user’s information. |
| Middleware | **ConnectionToServer** | That class is used to implements the communication, using sockets, with the server. |
| **MainClass** | That class represent the main class of the application. Contains the main method and the method for the initialization of the application. |
| **MessageReceiver** | That class represents a thread that is used to receive messages from the server. In this way the main thread ( GUI ) is not blocked. |

### Server’s classes:

|  |  |  |
| --- | --- | --- |
|  | |  |
| MiddleWare | **Scraper** | It contains all the method useful in order to perform the scraping activity. In particular the server can call the main method *scrape()* for the purpose of start a new scraping session. |
| **ServerAsynchronousWorker** | Implements the main thread for the communications with the clients, creating a socket on which requests to the server are listened. |
| **ServerMain** | ServerMain is the main class of the server side of the application. It initializes the main structures of the server, to launch the worker threads and does the scrape periodically. |
|  | **ServerWorker** | ServerWorker is a thread communicating with a specific user. It collects all the methods for the intercommunication with the client’s socket. |
| **TextAnalyzer** | That class contains the methods for analyzing a text and extracting its keywords and their number of occurrences within the text. |
| Back-end | **Article** | That class represents an article with its related information. Is used to store the information about the articles on the database. |
| **Filters** |  |
| **MongoDBManager** | That class contains all the method used to communicate with the database and used to perform the queries. |
| **User** | That class represents an user with its relat |
| **View** |  |

### Message’s classes:

|  |  |  |
| --- | --- | --- |
|  | |  |
|  | **ACKMsg** | That class implements the message sent by the server to the client in order to confirm the reception of the previous object. |
| **ArticleresponseMsg** | That class represents the message sent by the server to the clients containing a list of articles. That messages is sent after a search or at the beginning for the recommended articles. |
| **ChangePeriodMsg** | That class represents the message sent by the client to the server in order to communicate the changing of scraping period. |
| **ChangeSiteMsg** | That class represents the message sent by the client to the server in order to communicate the changing of scraping sites. |
| **ClientResponseMsg** | That class represents the message sent by the server to the client in order to communicate the list of users registered to the application. |
| **FindMsg** | That class implements the message sent by the client to the server when a user clicks on search button. That message contains the parameters of the search (keyword, filters). |
| **LoginMsg** | That class implements the message sent by the client to the server in order perform the login. In that message the client sent the username and password of the user. |
| **LoginResponseMsg** | That class represent the response to the loginMsg sent by the server to the client. That class contains a code which stands for the result of login operation. |
| **SignInMsg** | That class implements the message sent by the client to the server in order to perform the signIn phase. That message contains all the information specified by the user. |
|  | **SignInResponseMsg** | That class represent the response to the SignInMsg sent by the server to the client. That class contains a code which stands for the result of signIn operation. |
|  | **TrendResponseMsg** | That class implements the message sent by the server to the client at the opening of the application. That message is used to sent the list of all the trending keywords with their related percentage of importance. |
|  | **ViewMsg** | That class implements the message sent by the client to the server when a user clicks on an searched article. That message is used to sent to the server some information that will be used to compute recommended articles for the user. |

## UML Diagram

### Server UML Diagram

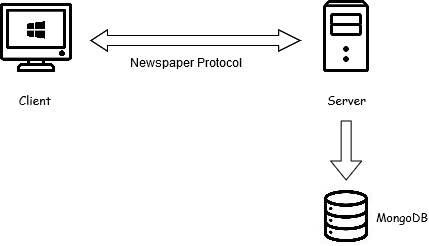
### 

### Client UML Diagram

Immagine che contiene testo

Descrizione generata automaticamente

## Application’s architecture diagram



Our application is based on the Client-Server paradigm. On the client-side we have the GUI used by the users to perform a search, view recommended or result articles, view trending keywords or the article’s text analysis.

All the information requested by the user are computed within a server. In particular the server does periodically a scrape of the 4 most important newspaper’s website, storing into the db all the collected articles.

After that the server computes for each saved article its related text analysis, which stand for calculating for each word of the article its number of occurrences.

Moreover the server communicates with the clients receiving some requests. The server receives the request of recommended articles, the request of trending keywords but also a search request. Once the server has received a request, elaborates a response and using a socket send the answer to the client. The communication between client-server is based on a specific protocol:

Immagine che contiene testo

Descrizione generata automaticamente

After that we can summarize the server in 4 parts:

* Scraper: the component of the server used to collect articles from the 4 most important Italian newspaper websites.
* TextAnalyzer: the component of the server which computes for each article its text analysis.
* MongoDBManager: the component of the server used to store the information into the document database.
* ServerWorker: the component of the server used to handle the communication with the clients.

## Scraping

### Scraper class:

The Scraper class contains all the method useful in order to perform the scraping activity. In particular the server can call the main method *scrape()* for the purpose of start a new scraping session.

Our aim is to create first and then update a Database that store the information about the article published from the major Italian newspaper web-site (la Repubblica, il Corriere, Sky TG 24 and ANSA, even if the code was arranged in order to make adding new newspaper website easy). This is made in a structured way, distinguishing the authors, the title, the text, and etcetera.

### Scraping activity:

The scraping activity can be divided in two separated parts: retrieving the link of the new article and analyze the content of the article found.

Twitter’s article links retrieval:

In order to perform this first part, we choose to retrieve the link through Twitter for several reason the main ones are:

* The tweet’s structure is fixed, while the homepage of the newspaper websites are a bit messy and of course to analyze a fixed structure in order to fine a string that match your requirement is easier.
* Every tweet has an associated id, so it’s also easier search the find the link of the new article pushing aside the other ones.
* Twitter accounts of the major newspapers are constantly updated
* Finding the article link through Twitter, and more in particular through its specific API, reduces the request to the newspaper websites, and this is a good news because we have always the risk of being banned form the website because of an high number of request!

The already mentioned used API is *Twitter4j*;in particular the main class used is *Status* that represent one single status of a user.

Every time that a new scraping session start a new instance of *Twitter* class is created by the method *getTwitterFactory()* and for every newspapers’ Twitter account we try to retrieve the id associated to the last Tweet scraped (that is stored in a file called lastTweetId). If the file doesn’t exist, or we don’t have information about the account to scrape, we chose to take into consideration the last 1000 tweets posted by the user. Otherwise, if we have information about the last tweet scraped, we retrieve all the tweets up to this one.

After retrieving the new tweet, we store the id of the more recent tweet of each Newspaper in the file lastTweetId in order to guarantee that an article isn’t analyzed twice. At the end we have some tweets from which using the method *getArticleLink()* we extract the link of the article that eventually are related to that tweets.

### Retrieving information from websites

As regard the article analysis the code, organized in methods that are called using the convention *scrape+NewspaperName(),* could become messy because of the various possible structure of the web page. In order to retrieve much information as possible, during the design phase we used a log file that stores the part of the Article that our program wasn’t able to find with the associated link.

In order to perform a fair scrape, but trying to reduce as much as possible the time required at the same time, we decided to send a request to each website every 30 seconds.

### Workflow of scraping process

The behavior of the program is the following: assuming that we are analyzing 4 newspaper, and we have at least one article that is available and is waiting to be analyzed for each newspaper, we begin finding a link of an article associated to the newspaper number 1. Then, we make a request to that web page and we analyze it. We repeat these operations for each newspaper, and finally the program stops and wait for 30 seconds.

This operation is repeated while there are no more article left.

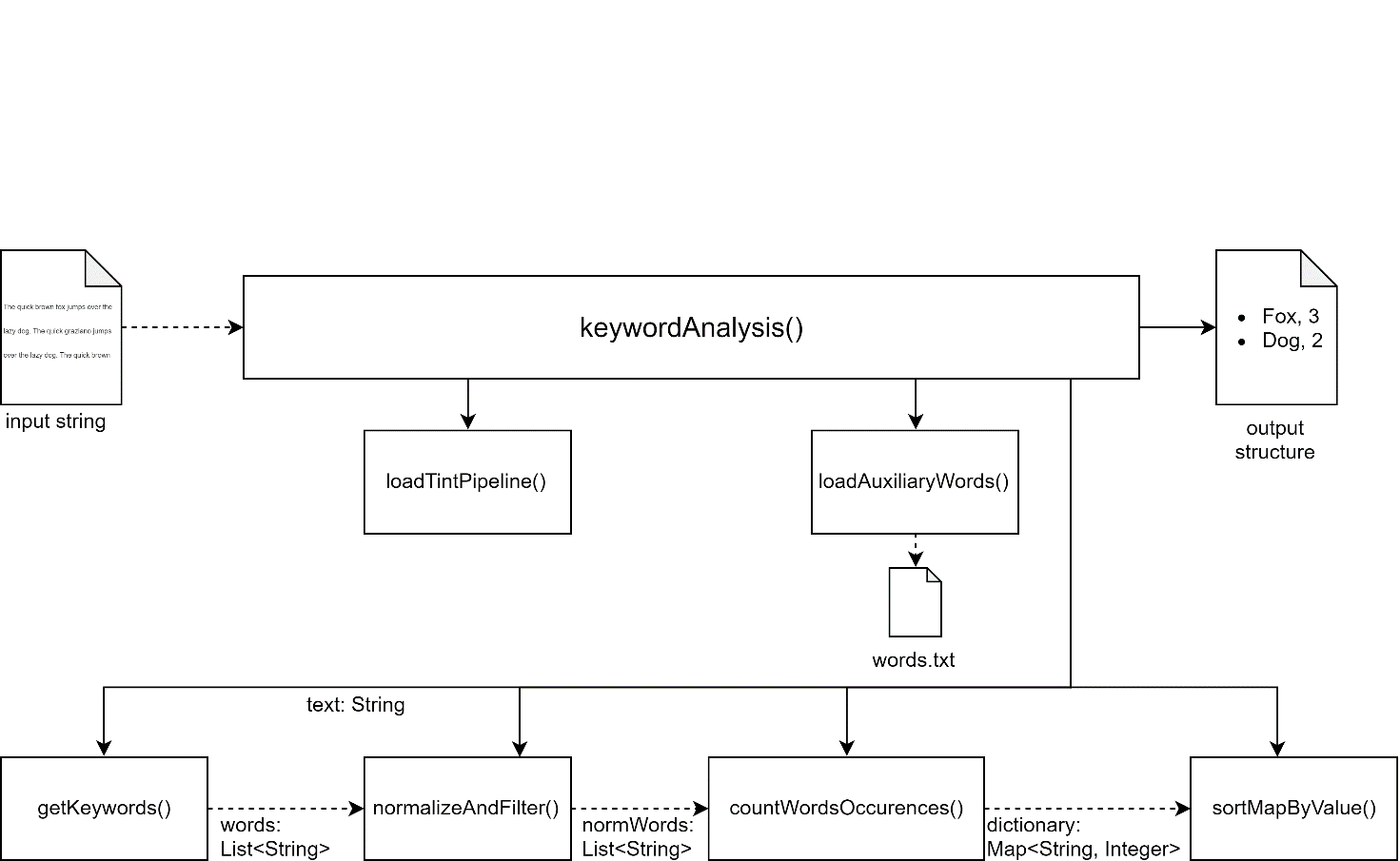
## Text Analyzer

TextAnalyzer class contains the methods for analyzing a text and extracting its keywords and their number of occurrences within the text. Keywords are proper or common nouns, and adjectives: each verb, adverb and other parts of the speech are not considered, since they don’t define the content of an article.

This class contains only one public static method, that is keywordAnalysis(). It takes a string as a parameter, and it will call some other functions to return the list of all the keywords, together with their occurrences, sorted by number of occurrences.

TextAnalyzer exploits a NLP (Natural Language Processing) framework, called Tint (The Italian NLP Tool), based on Stanford CoreNLP. This framework provides useful methods for the recognition of the role of a word within a sentence, and it’s designed for the Italian language.

A scheme representing the function calls for keywordAnalysis() is the following:



## Mongo DB

As we said before the database used in our application is MongoDB. We have used a document database in order to exploit some advantages. High flexibility and high-speed access to data are the most important characteristic of our database.

In fact we have used the MongoDB indexes reducing the number of reads before obtaining the correct answer:

 Indexes

public static void createIndexes() {

MongoCollection<Document> collection =database.getCollection("Article");

BasicDBObject obj = new BasicDBObject();

obj.put("Topic", 1); //

obj.put("Date", -1);

collection.createIndex(obj);

obj = new BasicDBObject();

obj.put("Keywords.keyword", 1);

obj.put("Date", -1);

collection.createIndex(obj);

collection = database.getCollection("Users");

obj = new BasicDBObject();

obj.put("userID", 1);

collection.createIndex(obj);

}

From an analysis we have individuated the heavy read operations on our Database in particular:

The user authentication is an operation done with an elevate frequency

Write: Once an user is saved in our Database no more write operation is required for that user

Read: Assuming that on average an user access to our service 2 times a day

Assuming 200 users are using our service every day

Statistics write vs read in one month

+ 200 writes on the user collection

+ 200\*2\*32 = 12800 read operations

MongoDB Results :

Query:

db.Users.find({"userID":"RiccardoXe"}).explain("executionStats")

Without Index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 1,

"executionTimeMillis" : 1,

"totalKeysExamined" : 0,

"totalDocsExamined" : 1602,

…

}

With index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 1,

"executionTimeMillis" : 0,

"totalKeysExamined" : 1,

"totalDocsExamined" : 1,

…

}

Usually users are interested in specific topics.

Taking the example above, if 100 users access to our service twice a day, with a great probability they will make a search by the tag they are interested in

Statistics write vs read

10/20 articles for a specific topic per day assuming 4 major topics

+ 20\*4=80 write per day

+ 2\*200=400 read operation by topic per Day

Obviously, this index is more useful in a scenario where an elevated number of users use our application every day.

Query

db.Article.find({"Topic":"Serie A"}).explain("executionStats")

Without Index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 161,

"executionTimeMillis" : 21,

"totalKeysExamined" : 0,

"totalDocsExamined" : 22720,

With Index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 161,

"executionTimeMillis" : 0,

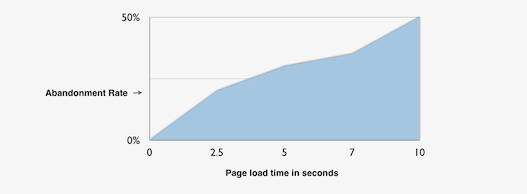
"totalKeysExamined" : 161,

"totalDocsExamined" : 161,

Keywords

Our service offers a specific table where one user can find the trending keyword of a chosen period. When an user select one of the keywords, a query find all the documents containing it. Assuming that users are very likely to read articles that contain trending different keywords every day.

Even if this operation is balanced between write and read, out main objective is to ensure fast response time to our users! If a user has to wait more than 2/3 second for a response is less prone to keep using our application.



Immage: <https://www.relentlesstechnology.com/wp-content/uploads/2012/02/site-speed-visitor-patience1.png>

Statistics for one day

Assuming that 50 articles are written containing 10 significant keyword each

and 200 users are using our service every day and each one search for the top 5 trending keywords we have

+ 50\*10=500 writes

+ 200\*5=1000 reads

Query

db.Article.find({"Keywords.keyword":"prestito"}).explain("executionStats")

Without Index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 66,

"executionTimeMillis" : 279,

"totalKeysExamined" : 0,

"totalDocsExamined" : 22720

With Index

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 66,

"executionTimeMillis" : 0,

"totalKeysExamined" : 66,

"totalDocsExamined" : 66,

Using MongoDB, we still had the possibility of performing aggregate queries. In this way the server was able to perform complex queries and sending the interesting results to the client. In particular the aggregate queries done by our application are:

retrieveUsersInformation()

This simple aggregation query is used to retrieve the UserID and the number of researches done for each user to allow the administrators to have an overview of the most active users

results = collection.aggregate(Arrays.asList(

new Document("$group", new Document("\_id","$userID")

.append("value", new Document("$sum", 1))),

new Document("$sort", new Document("value", -1))));

calculateTrendingKeyWords()

This aggregation query selects the articles from the queryDate to the current date. From each document the keywords are taken and grouped.

For each keyword the occurrence on each article are summed and saved in "Occur" and the number of Article in which the keyword is present is saved in "NumberOfArticles".

This informations are used to compute the formula for the trending keywords of the specified period. The trending keywords are the first 500 keywords ordered by value.

Value of the keyword = (Number of total occurrences) \* (Number of Article Containing Keyword)^2

This formula gives more importance to the quantity of Article in which a specific keyword is found than the total number of occurrences. This is needed to avoid to giving importance to keywords present in fewer articles but with an elevated number of occurrences.

//ValueOfKeyword=NumberOfArticles^2\*Occ

results = collection.aggregate(Arrays.asList(

new Document("$match", new Document("date", new Document("$gt",

queryDate))),

new Document("$unwind", "$Keywords"),

new Document("$group", new Document("\_id", "$Keywords.keyword")

.append("Occur", new Document("$sum", "$Keywords.Occ"))

.append("NumberOfArticles", new Document("$sum", 1))),

new Document("$project",new Document("\_id",1)

.append("Value", new Document("$multiply",indexes) )),

new Document("$sort", new Document("Value", -1)),

new Document("$limit",500)));

suggestedArticles(User u)

This aggregation query retrieve the most used “filters” by an user in a specific period of time (form queryDate to the current Date) the top three filters retrieved are used to find articles suggested for the user

results = collection.aggregate(Arrays.asList(

Aggregates.match(and(eq("userID",u.userID),gte("dateRead",queryDate))),

new Document("$group", new Document("\_id", "$filters")

.append("value", new Document("$sum", 1))),

new Document("$sort", new Document("value", -1)),

new Document("$limit", 3)));

## Server Worker

The server worker class is used to manage the communication with the clients, in particular has to handle the reception of messages from the clients, obtain the correct answer and after send the answer to the clients.

### ServerMain Class:

ServerMain is the main class of the server side of the application. It has the task to initialize the main structures of the server, to launch the worker threads, and to call the back-end methods for the articles’ text analysis and for forcing a scrape round (collecting articles from all sites backwards in time until the last scraped article).

When the server is launched, mongoDB indexes are created first thing. Then the listener thread ServerRequestListener and the scraping thread ServerAsynchronousWorker are created (more details in the following paragraphs).

It contains the scrapingPeriod variable, which determines the cool down period for the scraping process. It’s accessed in a thread-safe way, so that its modifications due to admin requests are concurrent with its readings.

trendingKeywords is a structure representing the top 10 trending keywords in the reference period and their frequency. It’s updated after each scraping round, and sent to each user that logs in to build the trending keyword chart.

When an admin wants to force a scraping round, the scrapeNow() method is called. It at first checks that no scraping threads are already running; if it is, this method returns with an error code. If no scraping threads are active, this method creates a ServerAsynchronousWorker object, and calls its round() method: this will start a round of scraping.

Moreover, each time a scraping round is completed, the articleTextAnalysis() method is called, which invokes the back-end methods for the retrieval of stored articles and the methods for text analysis collected in TextAnalyzer class.

### ServerRequestListener Class:

ServerRequestListener is the core of the multi-threaded server architecture. It implements the main thread for the communications with the clients, creating a socket on which requests to the server are listened. Each time a new connection is accepted, a new thread (ServerWorker) is created.

### ServerWorker Class:

ServerWorker is a thread communicating with a specific user. It collects all the methods for the intercommunication with the client’s socket.

Each time a connected user sends a request to the server, his correlated thread receives it and, according to the command, calls a specific method of ServerWorker class to handle it.

This class contains a method for each action provided for by the communication protocol; these methods are not represented in the scheme above, because we opted for a compact version of this diagram to make it more readable.

### ServerAsynchronousWorker Class:

ServerAsynchronousWorker is a thread that periodically works for updating the database of the system. In particular, each ServerMain.scrapingPeriod seconds, it does a scraping round (collecting all the articles until the last one seen during last round), it calls the methods for updating the articles present in the database by applying them their text analysis, and it finally invokes the method for calculating the new trending keywords.

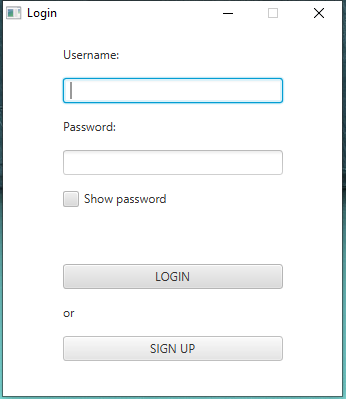
Since a scraping round can be forced by the admin, only one scraping process must be done at a time: this implies the presence of a semaphore (‘working’ boolean variable) that is set to true each time the round() function is called, and it is set to false when the round is completed. Before invoking this function, this variable is checked: if ‘working’ is true, then the round function returns immediately.

Accessing to working variable is thread-safe, due to the presence of a synchronization on the same variable; this is important to avoid situations in which two processes attempt to change the value of this variable.

## User manual

### Login

When the application is launched, the system will display a login form. The user has to put in its username and password and then to click on “LOGIN” button. If desired, the user can also see its password as plain text by clicking on the “Show/Hide Password” radio button.



### Sign-in

If the user doesn’t have credentials for the application, he can register himself to the service pressing the button SIGN IN. He has to insert his First Name, Last Name, Date of Birth, email and the desired username and password. After pressing the button REGISTER, the system will show a message which will explain the result of the registration process.

Immagine che contiene screenshot

Descrizione generata automaticamente

### Working with the application

### Administrator case

If the user is the administrator of the application, the main page showed will be the administrator control panel in which the admin can manage the setting of the application.

Immagine che contiene screenshot

Descrizione generata automaticamente

Here the admin can see the list of all the users registered and their related information.

Immagine che contiene screenshot

Descrizione generata automaticamente

The admin can also decide from which newspaper sites the application has to do the scraping of the articles. In particular he has to click on the checkbox of the desired websites and after that he has to click on the “APPLY” button.

Immagine che contiene screenshot

Descrizione generata automaticamente

The admin can manage also the scraping period of the application. He can specify in the “Scraping period” textfield the desired interval of time ( in minutes ). After the decision of the period, he has to click on the “UPDATE” button.

Instead of specifying a new scraping period, the admin can also force the scraping operation clicking on the “SCRAPE NOW” button. In this way the application receive the scraping command and starts to retrieve the articles from the specified sites.

Immagine che contiene screenshot

Descrizione generata automaticamente

### User case

Once the application has loaded the main page, the user can visualize two main tables.

Immagine che contiene screenshot

Descrizione generata automaticamente

The first table contains the trending keywords of the last week. In particular the user can click on one of those keywords. After that, the application will insert the selected word into the “Search Keyword” textfield. The user can start the search of the articles related to the keyword, specifying or not some filters, clicking on the “SEARCH” button.

Immagine che contiene screenshot

Descrizione generata automaticamente

Immagine che contiene screenshot

Descrizione generata automaticamente

The second table contains a list of recommended articles for the user. That list is based on the history of the user’s view. The user can scroll that list seeing the information about the articles. Moving the cursor on one of those articles, the user will see in the below table (Single Article overview) the article’s text analysis which contains the list of all the article’s word and their number of occurrence.

Immagine che contiene screenshot

Descrizione generata automaticamente

If the user clicks on a recommended article in the table, the application will open in the browser the newspaper web page containing the entire article.

The user has also the possibility to create his own search. In particular he can specify the desired keyword in the “Search keyword” field and specifying also some filters if he wants (like authors, City, Newspaper). After clicking the “Search” button the application will show the Result table which will contain the results of the user’s search.

Immagine che contiene screenshot

Descrizione generata automaticamente

The user can scroll that list seeing the information about the articles. Moving the cursor on one of those articles, the user will see in the below table (Single Article overview) the article’s text analysis which contains the list of all the article’s word and their number of occurrence. If the user clicks on a article in the Result table, the application will open in the browser the newspaper web page containing the entire article.