Application to simplify the extraction of data from Twitter

García Gracia, María Cristina

Curs 2017-2018

Director: CARLOS CASTILLO

GRAU EN ENGINYERIA INFORMÀTICA

Treball de Fi de Grau

Application to simplify the extraction of data from Twitter

María Cristina García Gracia

TREBALL FI DE GRAU

GRAU EN ENGINYERIA INFORMÀTICA

ESCOLA SUPERIOR POLITÈCNICA UPF

2018

DIRECTOR DEL TREBALL

Carlos Castillo

Acknowledgments

I would like to thank my supervisor Carlos Castillo for his guidance. I have learnt a lot during the project because of him and I have enjoyed developing it. Also thank you to my family and to my friends for all their support and love, and for always believing in me.

I owe this project to all them.

Abstract

Given the relevance of social networks today in this end-of-grade project, the Twitter API has been studied in order to develop a desktop application that allows to collect tweets through a search engine; the application, called "Search & Save for Twitter", connects to the Twitter API to download tweets, visualize them and save them in order to extract relevant information from them. This application aims to be easy to use for any user.

This report describes the motivation behind this work, the requirements of the project and the technologies used in it.

Resumen

Dada la relevancia de las redes sociales hoy en día en este trabajo de fin de grado se ha estudiado la API de Twitter para así poder desarrollar una aplicación de escritorio que permite recolectar de tweets mediante un buscador; la aplicación, llamada "Search & Save for Twitter", se conecta a la API de Twitter para así poder descargar tweets, visualizarlos y guardarlos para poder extraer información relevante de ellos. Esta aplicación tiene como objetivo ser fácil de utilizar para cualquier usuario.

En esta memoria se describe la motivación detrás de este trabajo, los requerimientos del proyecto, su desarrollo y las tecnologías utilizadas en él.

CONTENTS

Index of figures

1. MOTIVATION

1.1. Importance of the social media

Social media is one of the most important industries of all time, and that is because people have found in social networks a place in which they can relate and they are free to express themselves, share ideas, moments, etc. [1]. It is enough to contemplate what happens on the Internet in 1 minute in 2018 [2].

- 42,033,600,000 Facebook logins
- 159,840,000,000 Google searches
- 1,641,600,000,000 WhatsApp messages sent
- 8,078,400,000,000 emails sent

If we compare the data between different years, it is clear that each year more users join to different social networks, and that causes an increase of content:

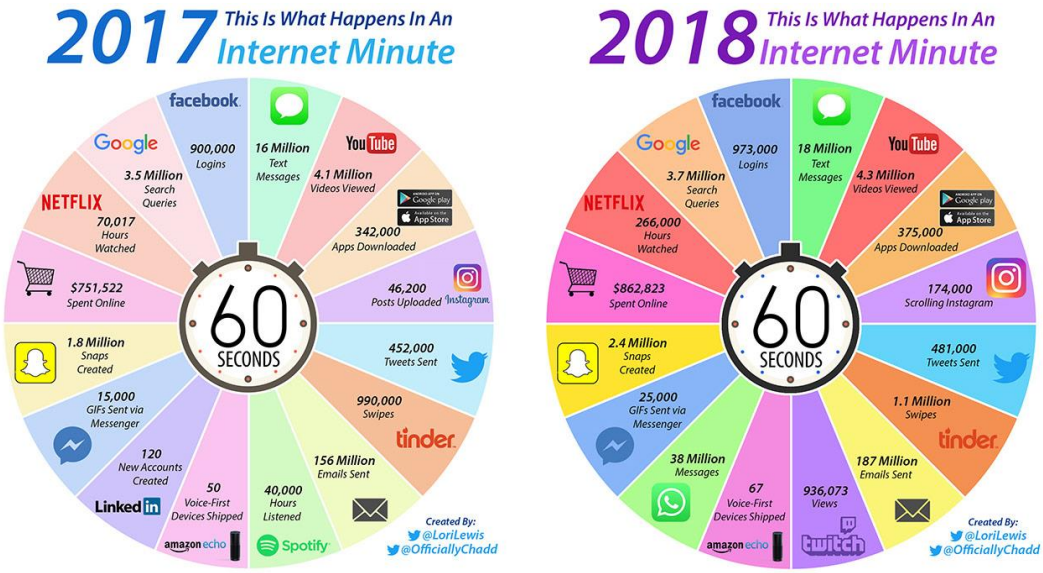


Figure 1. Image from: http://www.visualcapitalist.com/internet-minute-2018/, "This is what happens in an Internet Minute."

It is not only the exchange of experiences and opinions between users what makes attractive social media, but also its impact in the companies: nowadays almost every website has an account of Twitter, Facebook or Instagram available, among others, and that has produced a boom in online marketing and house brands (either businesses, movie or music stars, athletes...); social media allows brands to create a closer relationship with the consumer, which leads to happier consumers and sellers [3].

On the other hand, social networks, especially Facebook and Twitter, have revolutionised election campaign in the USA, since they allow receiving information about polls and organising campaigns or even fundraisers at no cost, and in a matter of seconds. Nowadays politicians want to be in the social media to be closer to their voters (see for instance Donald Trump, who is very active in the social network Twitter, publishing tweets almost every day) [4].

On the other hand, journalism is one of the careers that have undergone the most the impact of social media and that is basically because social networks are a type of "media": now journalists can interact more directly with listeners/followers, find news in a more efficient way with Twitter, or announce themselves through these social networks. But now more than ever it is important to confirm the sources to know whether a new is true or false. [5]

As it can be seen, it is a broad matter that affects many segments in the society, so it supposes a very interesting subject to work with; even so, as the previous figure shows, the amount of data that social networks generate is too large for concluding anything: therefore, studying new tools to collect and analyse these data was even more interesting.

1.2. Tools for social media analysis

After observing a comparison between the 10 best social media monitoring software [6], we can see, among other things, that the vast majority of the tools are large payment companies and many of them are intended to be used in mobile platforms or through browsers: this can be an inconvenience for those professionals who work on their own, who cannot afford a business plan or the tools that require them.

On the other hand, there is a problem with which you can find some of these professions that do not have algorithms: there is nothing more to remedy than trust the tools that complete them or that they do not, they extract the data in clear [5]. Finally, it should be noted that these monitoring software have many tools, which can cause confusion when using them because you do not know very well where to start

For all this, he came to the conclusion of wanting to use a simple tool to use, without the level of knowledge of the user being a difficulty, accessible and multiplatform, in order to provide more analysis options to all those professionals (journalists, doctors, nurses , engineers) that need to require it.

2. SOFTWARE REQUIREMENTS

In this chapter there are described the set of requirements that are considered essential in "Search and Save for Twitter". Each requirement is presented as a user story, simulating the prospects of the user and the necessary criteria to accomplish the requirement; finally they are accompanied with a mock-up with the desired result.

2.1. User stories

a) User story 1: Install the program

- As a user, I want to install "Search and Save for Twitter" in my computer. As a desktop application it requires to be installed in a computer in order to be used.

Acceptance criteria:

- The user requires the "Search and Save for Twitter" installer on his/her computer.

License Agreement:

In this screen the user accepts the General Public License (GPL) Agreement in order to continue with the installation process:

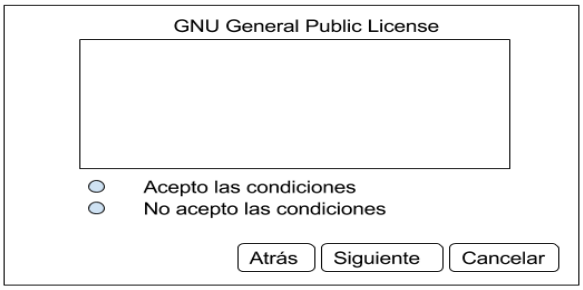


Figure 2. License Agreement

Destination location:

In this screen the user selects the installation location for the application and the amount of space that will require from the computer its installation.

Figure 3. Destination folder

Last steps of the installation:

In this screen the user can perform the last custom preferences, such as to create a desktop shortcut.

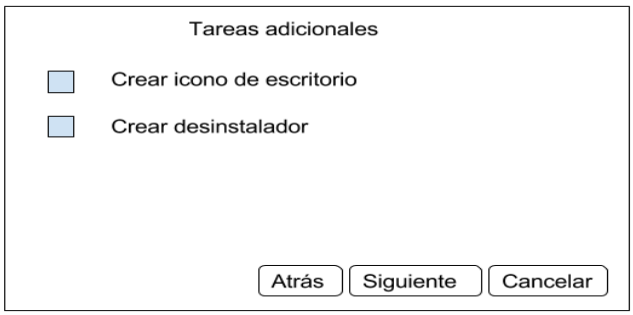


Figure 4. Last installation steps

b) User story 2: Login with Twitter in "Search and Save for Twitter

- As a user, I want to login with my Twitter user in the application. I will have two options: the first time I use the application I will register in it through my Twitter account; the second time, I will login directly into the application itself.

Acceptance criteria:

- The user must have a Twitter account.
    - If not, the user must sing up on Twitter first.

- The system must be able to connect with the Twitter authentication system.

- The system must be able to show to the user his/her basic account information, such as the username.

Login view:

In this screen the user has two login ways: "New user" in case it is the first time that uses the application, or "Existing user" in case that it has used the application before.

Figure 5. Login view mock up

Main window after login:

This is the main screen of the application after the successful login of the user:

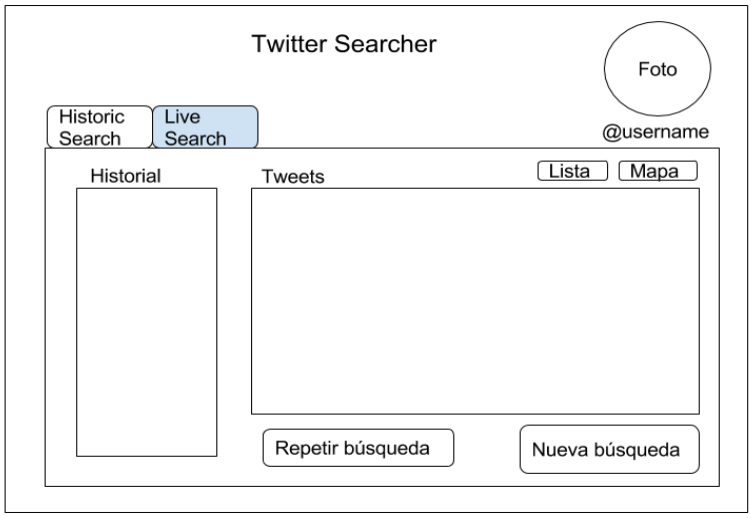


Figure 6. Main view mock up

c) User story 3: Search tweets

- As a user, I want to execute searches of tweets with "Search and Save for Twitter" based on a term of interest.

Acceptance criteria:

- The system must be able to connect to the Search API of Twitter in order to dive inside the historical records of Twitter.
- The user must be able to search tweets by a query.
- The user must be able to execute the same search in case he/she wants to add more tweets in it.
- The tweets must be stored in a database.

Search engine view:

In this screen the user is able to perform searches of tweets.

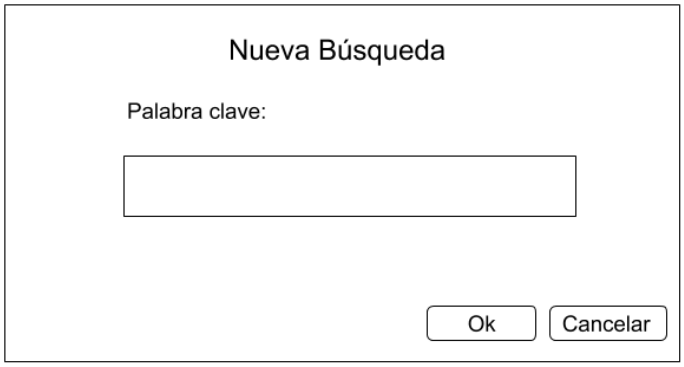


Figure 7. Search engine mock up

d) User story 4: View the search results

- As a user, I want to visualise a list with all the tweets of my search.

Acceptance criteria:

- The system must show a table with the list of tweets for that search; for each tweet it must show:
  - The date of creation.
  - The author.
  - The text.

- The tweets of the list must be ordered from recent to oldest.
- The user must be able to click on a tweet to see it in the browser.

Current search view:

This is a detailed view of the main window (figure 6) with the tweets of a specific search downloaded:

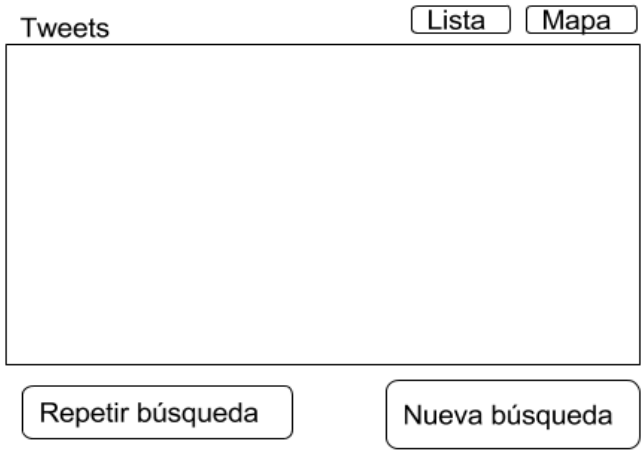


Figure 8. Current view

e) User story 5: View the search history

1. As a user, I want to visualise a list of all the searches that I have done.

Acceptance criteria:

- The system must show a table with the list of searches done by the user; for each search it must show:
    - The date in which the search was done.
    - The query of the search.
- The user must be able to click a search from the history search to see the tweets of the search in the view window.

History search view:

This is a detailed view of the main window (figure 6) with all the searches performed by the user:

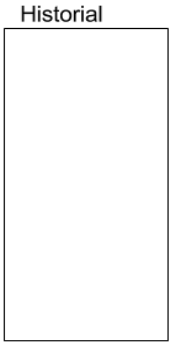


Figure 9. History search view

f) User story 6: Export tweets to a csv file

1. As a user I want to export my searches into csv files.

Acceptance criteria:

- The system must have buttons to allow the user to export a specific search.
- The system must allow the user to view the document after the exportation.

Export button:

For each search the user will be able to export all the content:

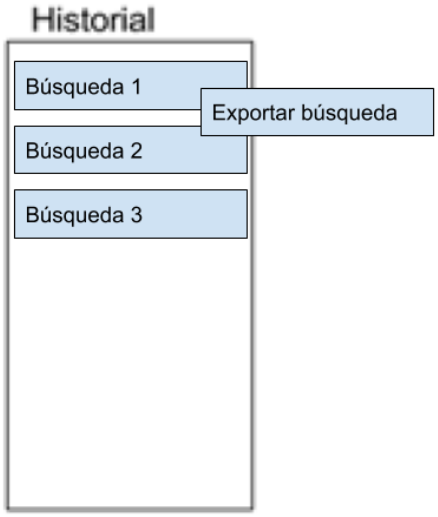


Figure 10. Export button

2.2. Data model

In this project a database was designed based on the Relational Model, which contains the following tables: a table with the user's information (user), a table with the information of each search (collection), and a table with the information of each tweet (tweet).

The *user* table contains basic information about the user who performs the searches in our application:

- **username**: is a string containing the name of the user in Twitter; with this the application identifies which is the user connected. This is also the primary key of this table.
- **access_token**: is a string containing one of the access keys that allow the application to connect to Twitter on behalf of the user. With this the user is able to do queries through the application.
- **access_secret:** is a string that keeps the other access key (the secret key) of the user in order to connect to the Twitter API and do queries.

The system allows to save information of multiple users, although there will be only one user connected per computer.

The *collection* table contains information about every search that the user does: the

- **collection_id**: an integer that identifies each search made by the user. This is the primary key of the table.
- **username:** this is a string containing the name of the user that performed the search; with this the system will be able to retrieve the searches done by this user every time he/she logs in the application. This is a foreign key referring to the previous table.
- **time_start**: this is a date that indicates the moment when the search started; with this the user will be able to see when was performed the search.

- **time_end**: which is a date that indicates the moment when the search has ended. This parameter of the table, unlike the other ones, does not have the "not null" constraint, because it is information that we update after the search is finished (the system stores information about the search at the beginning of it).
- **type**: a string value that describes the kind of search that has been done; this is useful for the system when it is able to performs different types of searches.
- **query**: a string that contains term that the user wants to search; this is the term by which the user distinguishes between different searches in the main menu of the application.

The *tweet* table contains information regarding the tweet that has been downloaded, most of them extracted directly from the Twitter API:

- **tweet_id**: is a long that identifies the tweet among others; this value is directly extracted from the Twitter API.
- **collection_id**: is an integer that indicates to which collection the tweet belongs.
- **author**: is a string which contains the name of the user that posted the tweet; in case the tweet is a retweet, the author corresponds with the one who made the retweet, not the one who posted it. The user will visualise this date through the main view of the application.
- **created_at**: a date containing the time when the tweet was created; the user will visualise this date through the main view of the application.
- **text**: a string containing the text of the tweet; the user will visualise this date through the main view of the application.
- **retweet**: a boolean telling if the tweet is a retweet or not; with this parameter the user will be able to use one of the visualisations mode incorporated in the application.
- **raw_tweet**: a representation in JSON of the downloaded tweet; the system stores this JSON to access rapidly to its content in case the system requires of any of them.

Following there is included the diagram of the database:

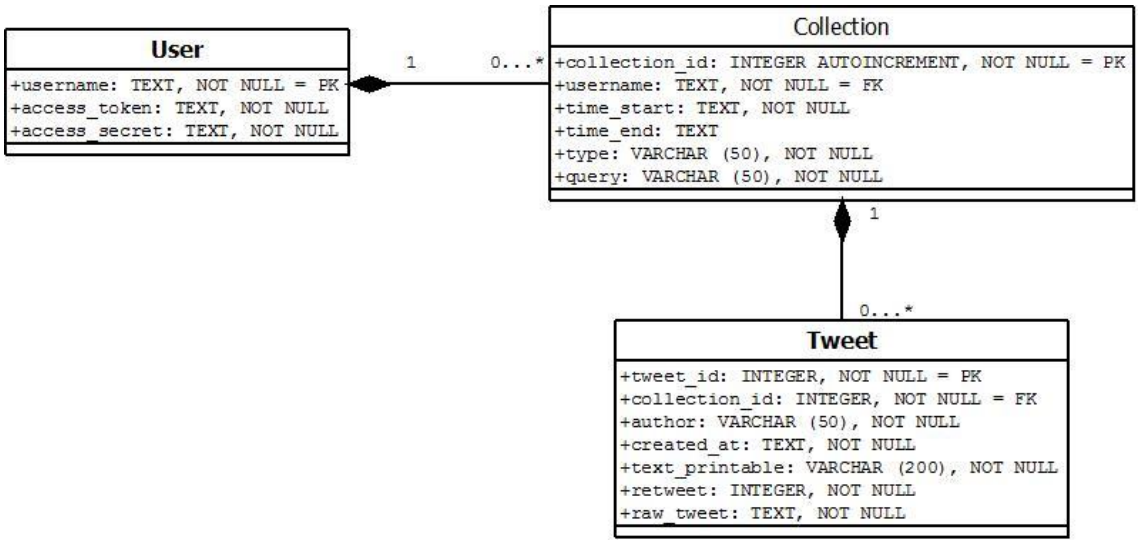


Figure 11. Data model

Here there are included the sql code for each table:

USER TABLE:

```
CREATE TABLE user (
      username            TEXT   PRIMARY KEY
                                 NOT NULL,
      access_token        TEXT   NOT NULL,
      access_secret       TEXT   NOT NULL
`
```

Figure 12. User table

COLLECTION TABLE:

```
CREATE TABLE collection (
      collection_id       INTEGER      PRIMARY KEY  AUTOINCREMENT
                                       NOT NULL,
      username            TEXT         NOT NULL
                                       REFERENCES user (username) ON DELETE CASCADE,
      time_start          TEXT         NOT NULL,
      time_end            TEXT,
      type                VARCHAR (50) NOT NULL,
      query               VARCHAR (50) NOT NULL
```

Figure 13. Collection table

TWEET TABLE:

```
CREATE TABLE tweet (
      tweet_id           INTEGER       PRIMARY KEY
                                       NOT NULL,
      collection_id      INTEGER       NOT NULL
                                       REFERENCES collection (collection_id) ON DELETE CASCADE
      author             VARCHAR (50)  NOT NULL,
      created_at         TEXT          NOT NULL,
      text_printable     VARCHAR (200) NOT NULL,
      retweet            INTEGER       NOT NULL,
      raw_tweet          TEXT          NOT NULL
);
```

Figure 14. Tweet table

3. SOFTWARE DEVELOPMENT

3.1. Introduction: Main technologies

In this chapter I will explain the development of each user story that it has been presented before, and which technologies were used for each one.

Based on the user stories and the database model that I presented previously, "Search & Save for Twitter" requires a connection to the Twitter API (first so that the user can authenticate to his/her Twitter account, and second to download the tweets), a database to store the relevant data (user, collection, tweet) and an interface to display all the collected data.

I decided to use *Java* to construct my application for two main reasons: firstly because I am very familiar with this language after programming with it several times in different projects during my bachelor, and because nowadays it is a widely used and present language (it is the most used language according to the TIOBE index on June 2018 [7]), therefore the number of libraries linked to it are very diverse.

The entire program is developed following the Model-View-Controller pattern (MVC) to ease the development, the code reuse and the readability of the code [8].

The interface has been completely developed in JavaFX, a GUI library of Java for building cross-platform applications; this API library is built into the Java SDK, which allows programming the entire application in Java. For the development of the views of the interface I used JavaFX Scene Builder, a visual layout tool to design the interfaces quickly and easily: all the views of the application are managed through xml files (they correspond to the *.fxml files in the project) [9].

The IDE used to develop this project is Eclipse (Oxygen version).

3.2. Installation

The installer is configured with the Ant Build tool of Eclipse, a tool to generate installers of Java programs for several platforms such as Windows, Linux or MacOs; this tool extracts all the resources and all the java files of the project and it will generate the executable file.

Once the installer is created are added the installation steps described in the first user story of this document: a license agreement document, a destination folder configuration and the additional tasks such as creating a desktop shortcut. This is done with the "Inno Setup" software:

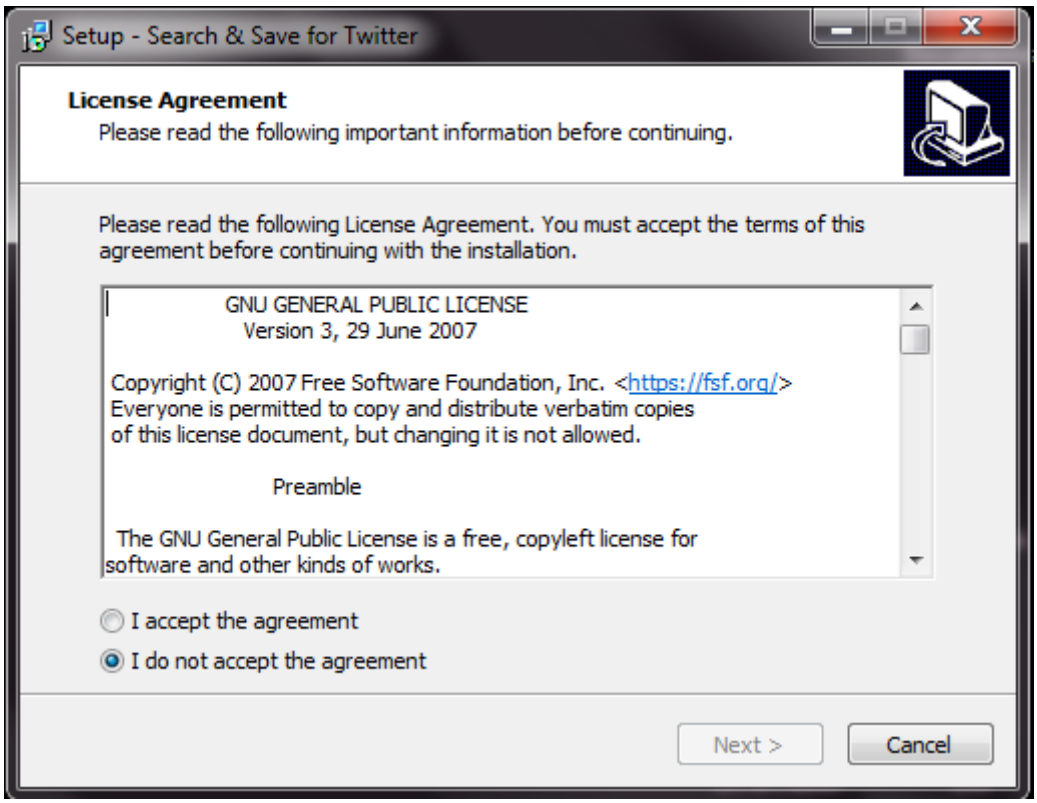


Figure 15. Licence agreement setup

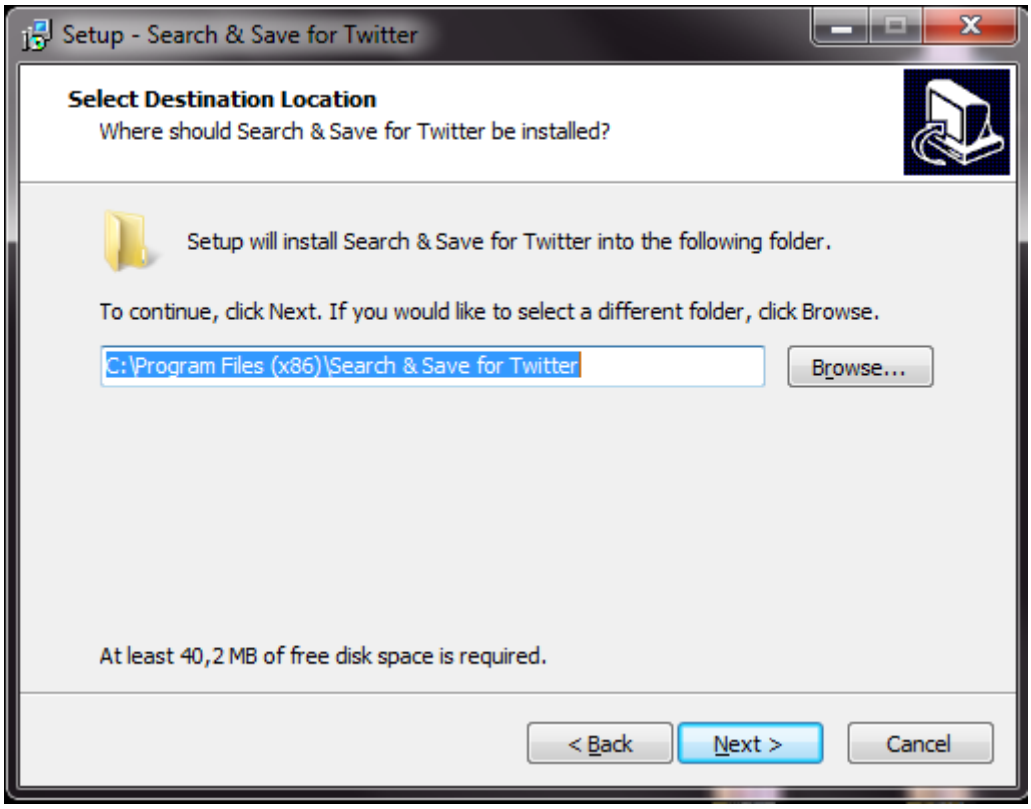


Figure 16. Destination folder setup

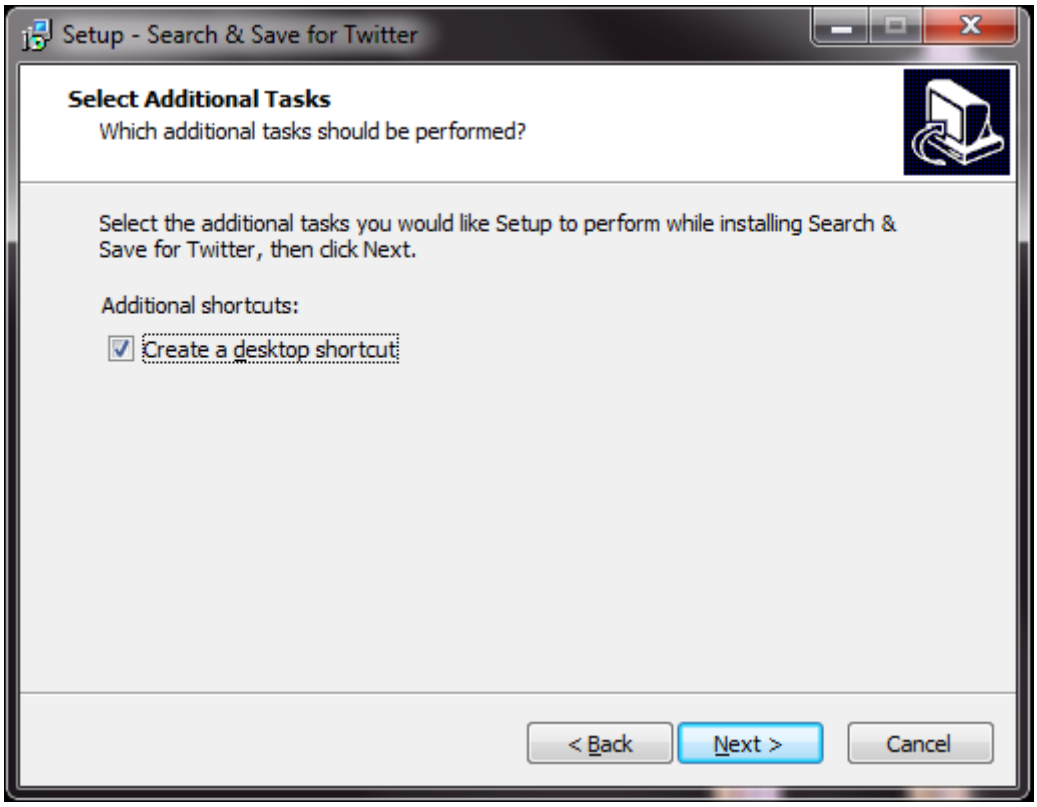


Figure 17. Last steps setup

This project has been tested in a Windows 7 and a Windows 10 machine.

3.3. Database setup

In order to manage all the collected data the application uses the database engine SQLite, a relational database management system. This is a widely spread software that suits very well with the requirements of our system because of the main features of this engine: it is **self-contained** (requires minimal support from the operating system, which permits to be used in many different systems), it is **serverless** (it is fully-integrated with the application that will access it, so it does not require a server to operate), it has **zero-configuration** (you just need to connect the JDBC driver with Eclipse to start using it) and it is **transactional** (it permits secure operations even after a crush of the application, operating system, etc.)[10].

Furthermore, the syntax is compatible with SQL, which permitted to learn fast how to use this engine, and also permits an easy interaction with Java through Java Database Connectivity (JDBC) API.

In this project we have 3 main classes to manage all the database info: *DatabaseDAO.java* (managing the creation of the database and the connection), *DBUserDAO.java* (regarding all the information about the user connected) and *DBCollection.java* (regarding all the information of each search; note that the searches done by the system are called "Collections" in the database).

The system will work with just one connection to avoid overloading the database: that is why we use a Singleton for the main database file, where we define the connection to access it, a configuration properties object (which it is explained later), and the path of the database, which will be defined when the instance is generated for the first time:

```
private static DatabaseDAO instance;
private Connection c;
private SQLiteConfig config;
private String databasePath;

private DatabaseDAO(String path) {
	databasePath = path;
}

public static DatabaseDAO getInstance(String path) {
	if (instance == null) {
		instance = new DatabaseDAO(path);
	}
	return instance;
}
```

Figure 18. DatabaseDAO.java

The general performance of the database when it is executed a query is as follows: first the system prepares a query, then it connects to the database, and finally we execute a statement object with the specific query.

The first time is executed the application it creates the complete database; here there are included the tables that we specified before and their execution:

USER TABLE EXECUTION:

```
public void createUserTable() throws DatabaseWriteException {
	Statement stmtU = null;
	try {
		stmtU = c.createStatement();
		stmtU.executeUpdate(userTable);
	} catch (SQLException e) {
		throw new DatabaseWriteException("There was an error creating the users
table.",e);
	}
}
```

Figure 19. User table creation

COLLECTION TABLE EXECUTION:

```
public void createCollectionTable() throws DatabaseWriteException {
	Statement stmtC = null;
	try {
		stmtC = c.createStatement();
		stmtC.executeUpdate(collectionTable);
	} catch (SQLException e) {
		throw new DatabaseWriteException("There was an error creating the
collections table.",e);
	}
}
```

Figure 20. Collection table creation

TWEET TABLE EXECUTION:

```
public void createTweetTable() throws DatabaseWriteException {
	Statement stmtT = null;
	try {
		stmtT = c.createStatement();
		stmtT.executeUpdate(tweetTable);
	} catch (SQLException e) {
		throw new DatabaseWriteException("There was an error creating the
tweets table.",e);
	}
}
```

Figure 21. Tweet table creation

The connection of the application to the database is as follows: first we need to ensure that the foreign keys and the constraints specified in the "collection" and "tweet" tables work (we define this with the "config" property of the database); finally the system will check the connection with the database path provided at the beginning of the execution (if the database does not exists, it will generate a new file in the provided path):

```
public void connect() {
	config = new SQLiteConfig();
	config.enforceForeignKeys(true);
	try {
		Class.forName("org.sqlite.JDBC");
		c = DriverManager.getConnection("jdbc:sqlite:"+databasePath,
config.toProperties());
	} catch (Exception e) {
		System.err.println(e.getClass().getName() + ": " + e.getMessage());
	}
}
```

Figure 22. Database connection

3.4. User login

The first window that will appear in the application is the following one:

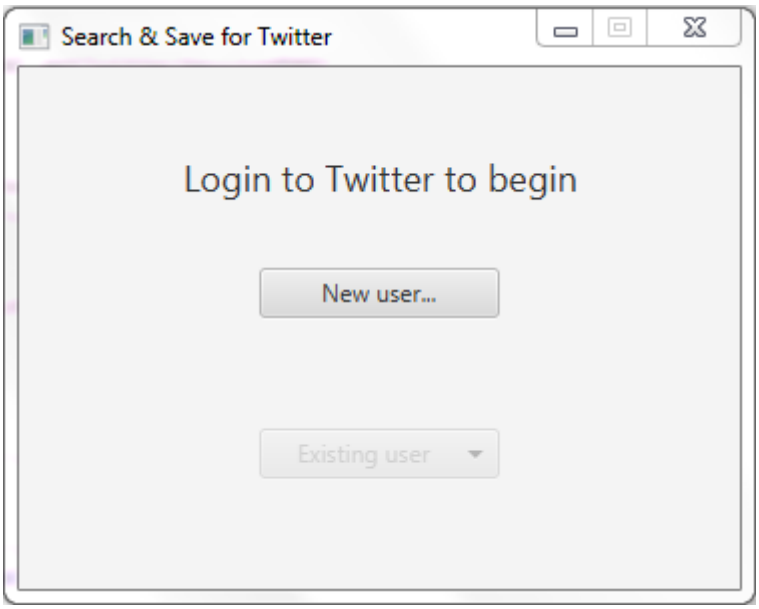


Figure 23. Login view

At this point the user has two options:

a. New user → Login when it is not registered in the application.
b. Existing user → Login when it is registered in the application.

It is necessary to ensure a secure communication between the application and Twitter in behalf of the user to guarantee that there will be no data leakage or other security problems: for this we are using OAuth (Open Authorization), a standard open source protocol which allows secure authentications through applications and web services; this protocol provides to the users an easy way to connect with third-party applications without compromising sensitive information, such as passwords, emails, etc.

The actors of this protocol are the *consumer* or *client* (Search & Save for Twitter"), which is the one who wants to connect to Twitter on behalf of a user, the *resource owner* or *end-user*, the user which gives access to its account to a client and the *service provider* or *API* (Twitter), which contains the user information (*resource server*) and which grants access to the client after the user approval (*authorization server*) [11].

Twitter has got their own implementation of the OAuth protocol to enable third-party applications to connect to their services (it is based on the *Client Credentials Grant* flow of the OAuth 2.0 framework) [12]:

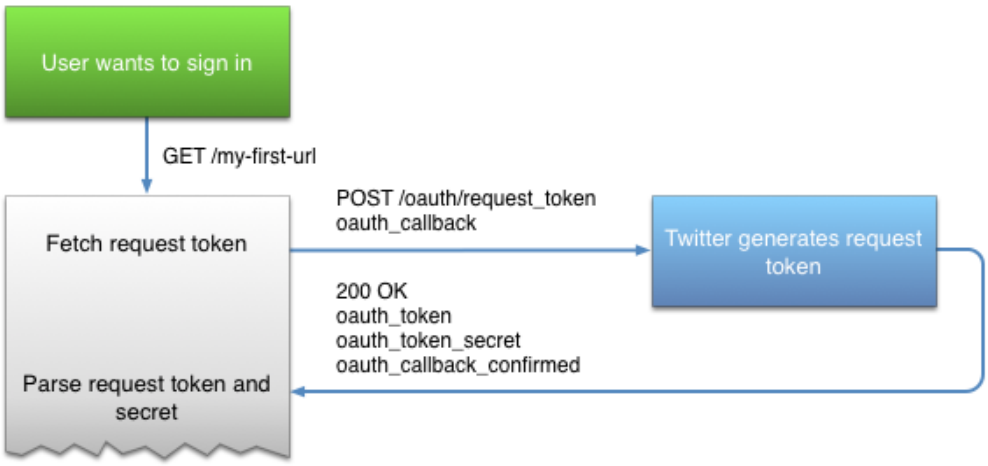


Figure 24. Image from: https://dev.twitter.com/web/sign-in/implementing. Sign in process

Before starting the authentication process, the application does the following:

First it determines a single user for the application, since only one user per machine will access: for this, as has been done with the database, the user's information is handled by a Singleton (DBUserDAO.java): he defines the connected user and the connection generated by the database (which we retrieve from the DAO of the database):

```
public static DBUserDAO instance;
private String user;
private Connection c;

private DBUserDAO() {
	c = Main.getDatabaseDAO().getConnection();
}

public static DBUserDAO getInstance() {
	if (instance == null) {
		instance = new DBUserDAO();
	}
	return instance;
}
```

Figure 25. DBUserDAO.java

Secondly it determines a single Twitter session, accessible from the entire system, which will remain active from the moment the user logs in until the session closes: the management is similar to that of the database and the user, through the TwitterSessionDAO.java file, containing basic information about the twitter connection, such as the instance and its callback_url (where the user is redirected after a succesful authentication):

```
private static TwitterSessionDAO instance;
private Twitter twitter;
private String callback_url;

private TwitterSessionDAO() {

}

public static TwitterSessionDAO getInstance() {
	if(instance == null) {
		instance = new TwitterSessionDAO();
	}
	return instance;
}
```

Figure 26. TwitterSessionDAO.java

Lastly it configures the Twitter session with the client data (in this case, the application): for this, the Singleton of Twitter contains the following method, which configuring the keys of the client (consumer key and consumer secret) and the callback_url of the application; after obtaining this data through the Twitter Applications Management portal, the system gets access to them through a properties file of this project:

```
public Twitter startTwitterSession() {

	AppProperties appProps = new AppProperties();

	try {
		appProps.loadFile("client.properties");
	} catch (IOException e) {
		e.printStackTrace();
	}

	ConfigurationBuilder builder = new ConfigurationBuilder();
	builder.setOAuthConsumerKey(appProps.getValue("consumer_key"));
	builder.setOAuthConsumerSecret(appProps.getValue("consumer_secret"));
	Configuration conf = builder.build();
	TwitterFactory factory = new TwitterFactory(conf);
	Twitter twitter = factory.getInstance();
	setTwitterInstance(twitter);
	callback_url = appProps.getValue("base_callback_url");
	return twitter;
}
```

Figure 27. Setting Twitter session

Now we will see the first case: a user which uses the application for the first time.

NEW USER

The process of creating a new user consists of the following: firstly, the application requests a token to the Twitter API in order to authenticate and access it; secondly, the application redirects the user to the authentication system to log in with his account; finally, the application collects the access tokens provided by Twitter, connects to its API and redirects the user to the main page of the application:

1. The system requests an authorization to access to Twitter: for this, first the client requests a request token, which is used to request permission to the Twitter API to connect to it:

```
public boolean createRequest(Twitter twitter, DBUserDAO dbu) throws NetworkException,
AccessException {

   this.dbu = dbu;
   this.twitter = twitter;

   try {
       requestToken = twitter.getOAuthRequestToken(
                      Main.getTwitterSessionDAO().getCallbackUrl());
   } catch (TwitterException e) {
       if (e.getStatusCode() == UNAUTHORIZED) {
          throw new AccessException("401: Unable to get the access token. Please check your
credentials.", e);
       } else {
          throw new NetworkException("You do not have internet connection. Please check it out
before continue",
                                     e);
       }
   }

   return true;

}
```

Figure 28. Request token process

2. Once the request token has been obtained, the application opens a WebView (a JavaFX element that opens web pages as if it were a browser) showing the Twitter authentication page (this is contained in the request token that is has requested):

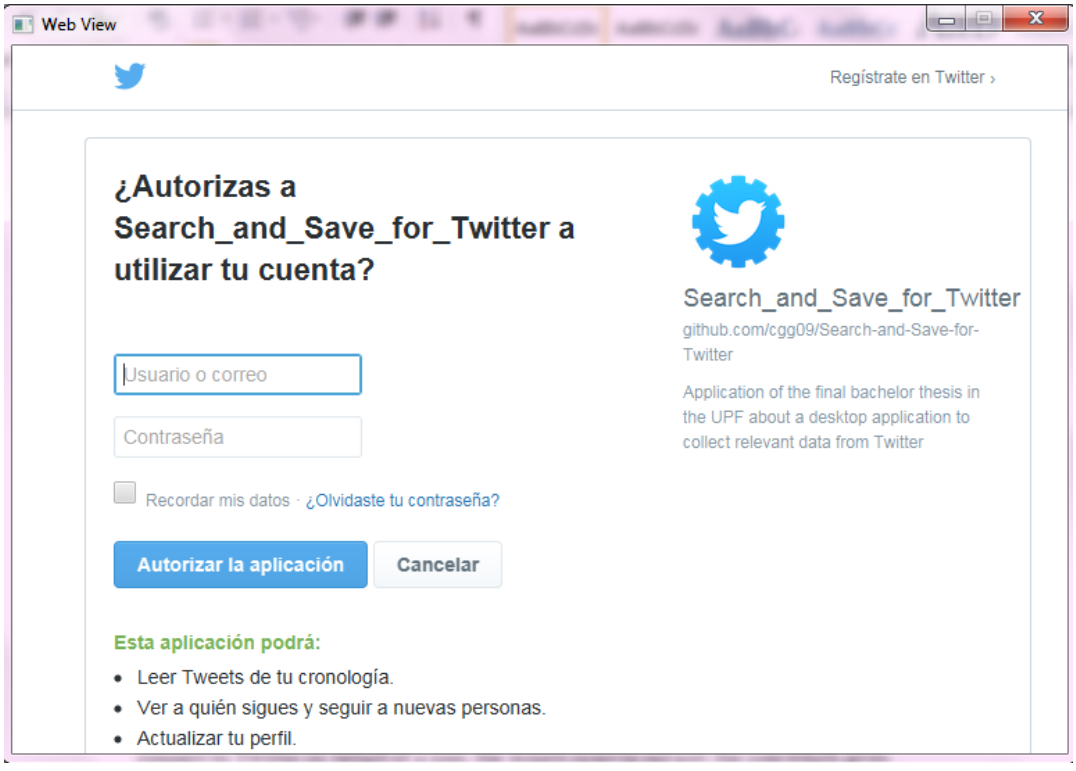


Figure 29. Twitter authorisation page

Through this WebView with the Twitter authentication portal, the user authenticates himself with his data; once done, the system redirects the user to the callback_url that we have defined (in this case, a web address on the server of localhost). The reason why this has been done is because the application is not found on any server nor will it operate in a real browser.

In order to collect the authentication data, since the callback_url itself is not real, the only moment of the execution in which we can obtain them is when the page cannot be loaded: for this, a "Worker" has been created, a concurrency object of JavaFX that can be used to track loading progress in order to detect any loading failure of the callback_url; then is when the verification url is retrieved; this url contains the authentication tokens that correspond to the user who has connected to Twitter:

```
public void retrieveTokens(Browser browser) {

webEngine = browser.getWebEngine();
webEngine.getLoadWorker().stateProperty().addListener(new ChangeListener<Worker.State>()
{
	@Override
	public void changed(ObservableValue<? extends javafx.concurrent.Worker.State>
	observable, javafx.concurrent.Worker.State oldState,
	javafx.concurrent.Worker.State newState) {

		if (newState.equals(javafx.concurrent.Worker.State.FAILED)) {
			String location = webEngine.getLocation();
			if(
			location.startsWith(Main.getTwitterSessionDAO().getCallbackUrl())) {
				String callbackURLWithTokens = location;
				browser.closeBrowser();
				try {
					verifyTokens(callbackURLWithTokens);
				} catch (AccessException | NetworkException e) {
					e.printStackTrace();
				}

			}
		}

	}
});

}
```

Figure 30. Retrieve tokens process

3. Finally, the system collects the authentication tokens from the url recovered in the previous step, verifies that they are correct (if this is the case, it would close the previous WebView) and requests the access token to Twitter (in order to connect to Twitter in user name) using the request token and the oauth verifier extracted from the url. Finally, the application saves in the relevant user database: the user name and the Access token, divided into token and token_secret, in order to access later in the application without having to request permission from Twitter every time.

```
public void verifyTokens(String callbackURL) throws AccessException, NetworkException {

       String oauthToken;
       String oauthVerifier;

       Map<String, String> urlMap = getQueryMap(callbackURL);
       oauthToken = urlMap.get("oauth_token");
       oauthVerifier = urlMap.get("oauth_verifier");

       if (oauthVerifier != null &&
((requestToken.getToken().toString().equalsIgnoreCase(oauthToken)))) {
          webEngine.getLoadWorker().cancel();
          try {
              accessToken = twitter.getOAuthAccessToken(requestToken, oauthVerifier);
          } catch (TwitterException e) {
              if (e.getStatusCode() == UNAUTHORIZED) {
                     throw new AccessException("401: Unable to get the access token. Please
check your credentials.", e);
              } else {
                     throw new NetworkException("You do not have internet connection. Please
check it out before continue", e);
                            }
                     }
              }

          try {
              dbu.saveLogin(twitter.verifyCredentials().getScreenName(),
       accessToken.getToken().toString(),
              accessToken.getTokenSecret().toString());
          } catch (DatabaseWriteException | TwitterException e) {
              e.printStackTrace();
          }
          Main.showSearch();

       }
```

Figure 31. Verify credentials process

Finally the application will show the Main view, which is explained in section 3.5. of this document.

EXISTING USER

This case is present when there is at least one user registered in the system:

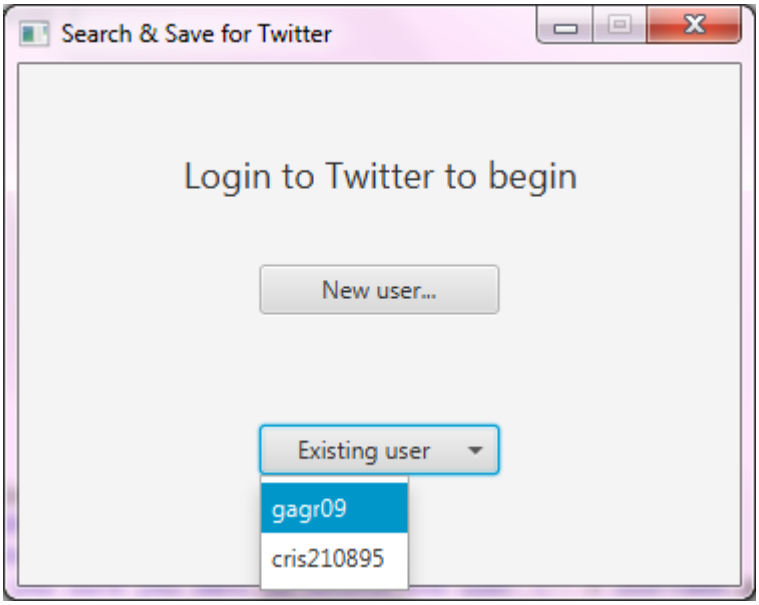


Figure 32. Login with users registered

Here the system just needs to extract the information from the database of the selected user and verifies that the tokens provided by Twitter are still correct (more details in section 3.8.) and then it will have access to the main view of the program:

```
public boolean retrieveSession(Twitter twitter, String user, DBUserDAO dbu) throws
NetworkException, AccessException {

	this.dbu = dbu;

	String token = null;
	try {
		token = dbu.getUserData("access_token", user);
	} catch (DatabaseReadException e) {
		e.printStackTrace();
	}
	String secret = null;
	try {
		secret = dbu.getUserData("access_secret", user);
	} catch (DatabaseReadException e) {
		e.printStackTrace();
	}
	AccessToken at = new AccessToken(token, secret);
	twitter.setOAuthAccessToken(at);

	try {
		twitter.verifyCredentials().getId();
	} catch (TwitterException e1) {
		if (e1.getStatusCode() == UNAUTHORIZED) {
			throw new AccessException(e1.getErrorMessage(), e1);
		}
		throw new NetworkException("You do not have internet connection. Please
check it out before continue", e1);
		}
	return true;
}
```

Figure 33. Login process of an existing user

3.5. Main window

When the user finishes the login the main window of the application will be displayed:

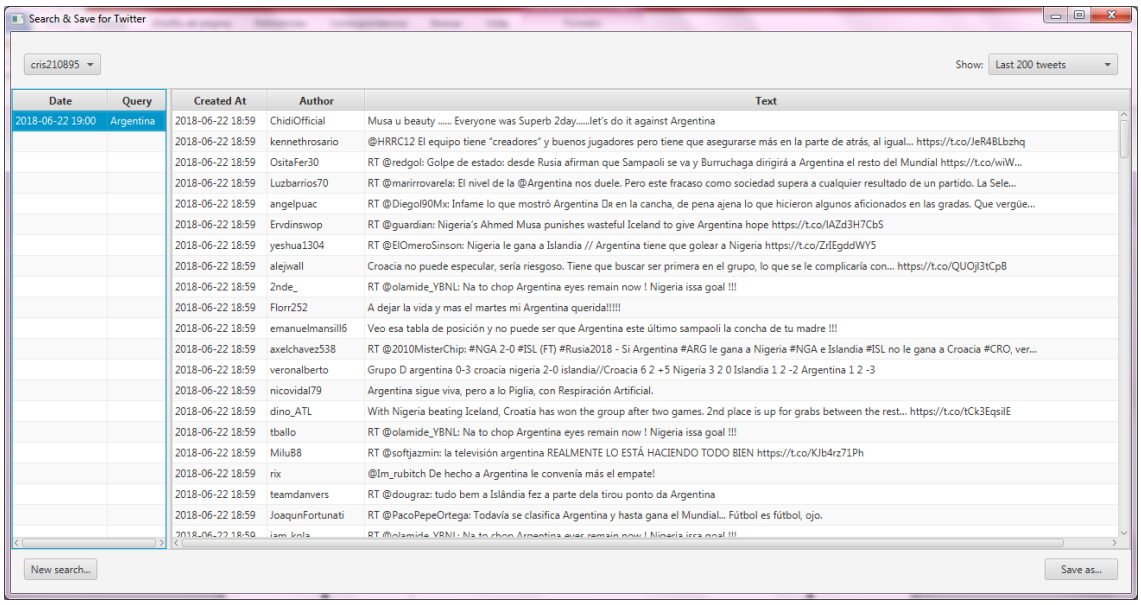


Figure 34. Main view

From here the user is able to perform the main actions of the application: search, view and save or export; this section in particular is focused in the visualisation functionalities, while the search and the exportation will be explained later.

In this view will find the following: there are two buttons on the top left and on the top right, two tables and two buttons on the bottom right and on the bottom left.

The top left button is a button to manage the user: here is shown the username of the user that logged in, and it shows also a dropdown menu with two options: log out, in case the user wanted to sign off the application, and delete user, in case he/she wants to delete its user of the application.

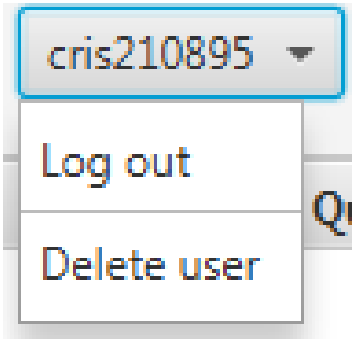


Figure 35. User settings menu

If the user clicks Log out, the application will close the window and the user will be redirected to the previously shown login page:

```
@FXML
private void signOut() {
	currentStage.close();
	Main.showLogin();
}
```

Figure 36. Sign out

Instead, if the user wants to delete the user it will be shown this warning message, asking him/her to confirm this operation:

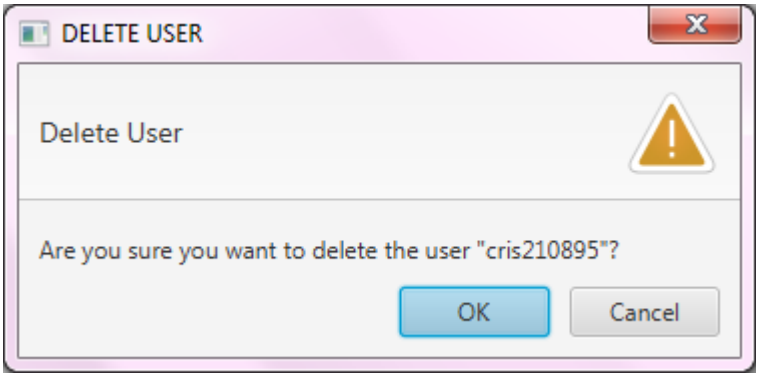


Figure 37. Delete user warning

Once pressed OK the application will delete the corresponding user by accessing to the database. Here is the code corresponding to this process:

```
@FXML
private void deleteUser() {
	Alert alert = new Alert(AlertType.WARNING);
	alert.setTitle("DELETE USER");
	alert.setHeaderText("Delete User");
	alert.setContentText("Are you sure you want to delete the user \"" +
	username.getText() + "\"?");

	ButtonType buttonTypeOk = new ButtonType("OK", ButtonData.OK_DONE);
	ButtonType buttonTypeCancel = new ButtonType("Cancel", ButtonData.CANCEL_CLOSE);

	alert.getButtonTypes().setAll(buttonTypeOk, buttonTypeCancel);

	Optional<ButtonType> result = alert.showAndWait();
	if (result.get() == buttonTypeOk) {
		try {
			Main.getDBUserDAO().deleteUser();
		} catch (DatabaseWriteException e) {
			e.printStackTrace();
		}

		username.setText("");
		currentStage.close();
		Main.showLogin();
	}
}
```

Figure 38. Delete user process

Here in this code it is described the delete action on the database:

```
private String delUser = "DELETE FROM user WHERE username = ?";

public void deleteUser() throws DatabaseWriteException {
	PreparedStatement psdt = null;
	try {
		psdt = c.prepareStatement(delUser);
		psdt.setString(1, user);
		psdt.executeUpdate();
	} catch (SQLException e) {
		throw new DatabaseWriteException("There was an error deleting the user.", e);
		}
	}
```

Figure 39. Delete user query

In this case, unlike the previous one, the deleted user will not be shown in the menu "Existing user".

On the other hand, due of the foreign keys configuration of the database, there will be deleted with the user all its relative information, such as the performed searches and their corresponding tweets.

The rest of the buttons of the view will be explained later, because the top right is related with the tweet view, and the bottom ones are related with the search and the exportation (New search and Save as). Here below will be detailed explained the interactions between the user and the tables of this view:

3.5.1. Search history

It is shown in the table of the left, and it is the one that contains all the searches that the user has done: it has two parameters, the starting date of the search (time_start) and the key word of the search (query); the table is sorted by default by the date. Different actions can be executed in any element of the table (search) by clicking the right or left button:

If the right button is clicked, the tweets of the search will be displayed in the Current view. On the other hand, clicking the left button will unfold a menu with secondary options, history options: the user will be either able to repeat the search, export it, or delete it:

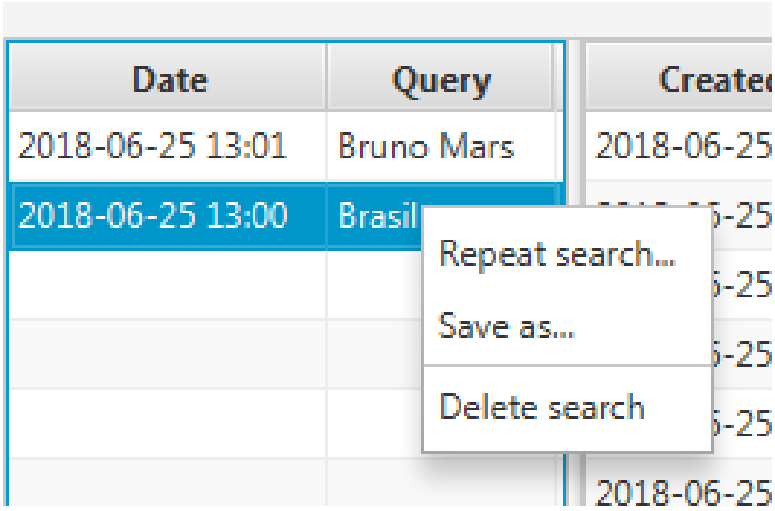


Figure 40. History search options menu

The functioning of the first two ones is very similar to the search and exportation: it was decided to add the first option in this menu to facilitate repeating the search for users in case they want to have it updated in a staggered manner; the second option was added basically to allow the user to have an immediate access to the exporting method when examining in detail that table in particular.

The third one, delete the search, works similarly to the previously explained option of deleting a user: first of all, a warning message is displayed; then, the information of the collection is searched in the database and deleted along with the corresponding tweets. Once the search is erased, it will be no longer displayed in the Main view:

3.5.2. Current view

It is shown in the table of the right, and it is the one that displays all the tweets relating to a specific search: the displayed parameters are the creation date of the tweet (date), its author (author) and its text (text).

An option to visualise the tweet in the browser has been added in the table: in order to do so, the user has to double-click on the tweet itself, and the user's default browser will be opened and will display the tweet:

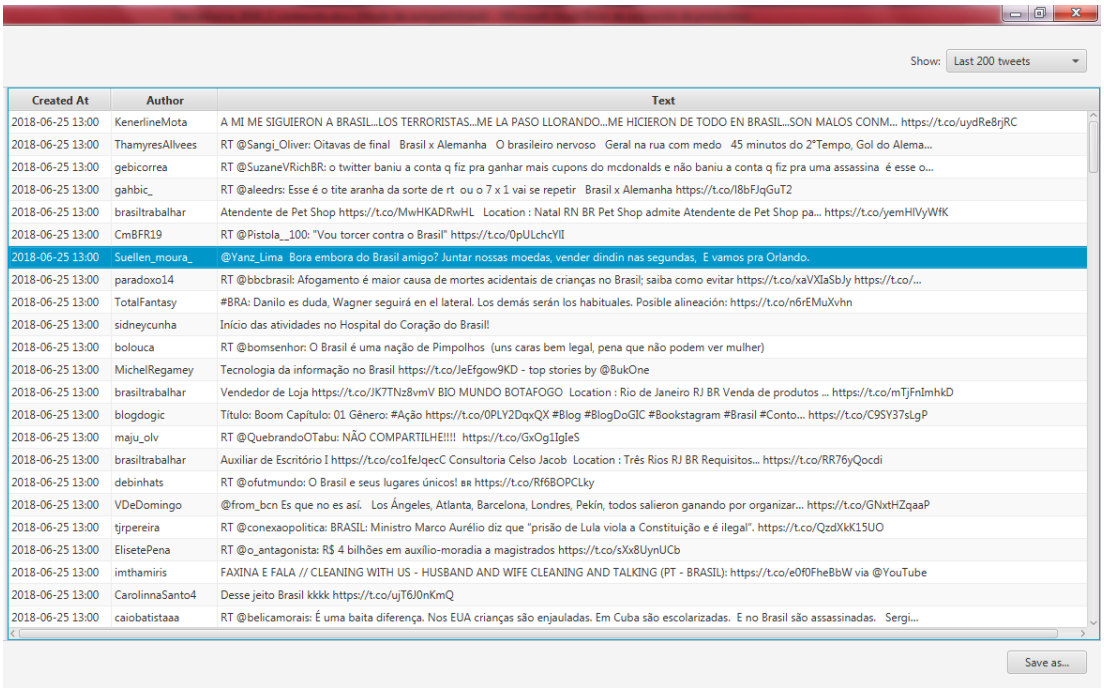


Figure 41. Current view

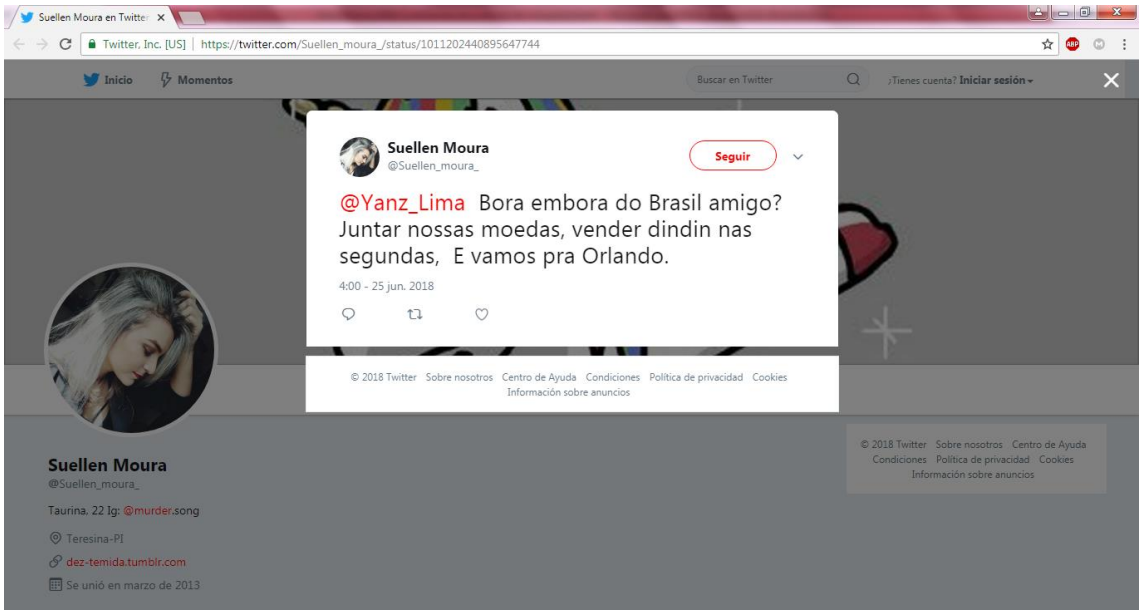


Figure 42. Tweet displayed on browser

Double-clicking on the tweet generates the tweet's url that will display it in the browser; this is done thanks to the name of the tweet's author and the tweet's identifier, and both are collected and saved in the "author" and "tweet_id" parameters of each tweet that is downloaded.

```
// show a selected tweet in a browser
currentSearch.setRowFactory(cRow -> {
	currentRow.setOnMouseClicked(event -> {
		if (event.getClickCount() == 2 && !currentRow.isEmpty()) {
			DisplayableTweet rowTweet = currentRow.getItem();
			HostServices host = Main.getInstance().getHostServices();
			host.showDocument("https://twitter.com/" + rowTweet.getAuthor() +
			"/status/" + rowTweet.getId());
		}
	});
	return currentRow;
});
```

Figure 43. Tweet displaying code

On the other hand, in this view the user can make use of the button of the top-right corner: it allows filtering the tweets that will be displayed in this table. There are three different filters: visualise "last 200 tweets", "all tweets (except RTs)", and "all tweets":

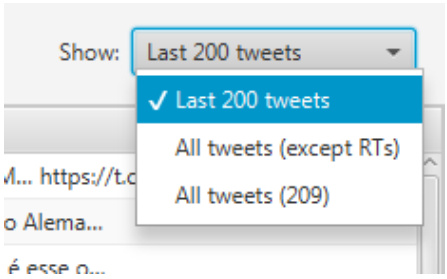


Figure 44. Filter menu

The first option displays the most recent 200 tweets of the search: this is especially useful when the search contains too many tweets, or when it has been repeated several times, since it allows visualizing easily the last downloaded tweets. This is default option when the tweets of a search are displayed.

The second option of this filter menu displays all the search's tweets that are not a retweet: this is mainly useful in case different users have retweeted the same and, as a consequence, a text is repeated in different search results. It is possible to know whether a tweet is a retweet or not by consulting the downloaded tweets attribute "getRetweetedStatus" (which is explained in more detail in section 3.6.).

Finally, the third option allows visualising all the downloaded tweets: besides, the number of downloaded tweets is displayed next to this option every time that the user does a search, as it is shown in the figure 44.

It should be noted that some downloaded tweets do not show the complete text, but they finish with an ellipsis: this is related to the API that has been used in this project and the character limit allowed in a tweet.

Since September 2017 Twitter doubled the original character limit, from 140 to 280, because they had verified that with some languages, such as English, the limit of 140 characters was easily overcome, thus complicating the users' experience [13]; leaving aside if this change is beneficial or not for the platform, what is clear is that it would be very

difficult for Twitter developers to adapt the entire API to this change: that is why the decision is made to keep the attribute as it is original of the text with a limit of 140 characters and add two new attributes to the Tweet Object, "truncated" and "extended_text"; On the one hand, truncated is a Boolean that indicates whether or not the tweet exceeds 140 characters; on the other, extended tweet is a string that contains the complete text of the tweet in case truncated is equal to true (in this case, in the "text" field, ellipses will be added).

It is important to take into account that although this is included in the main object of the entire Twitter API, the information to which the developer has access is variable according to the specific API being used; in this particular case these two new attributes are only accessible from the payment APIs (Premium and Enterprise) and Streaming API, among others. Because in this project we have focused on the Standard Search API, we do not have access to the full text of the tweet, so that tweets exceeding 140 show at the end of it the ellipses. This at first glance can be an inconvenience when working with these tweets, although it is compensated to the users by just double-clicking the tweet to view it in the browser, as explained earlier in this section.

In the following section is explained in more detail what Twitter API has been used for this project.

3.6. Search

This functionality can be carried out through the two buttons described in section 3.5 .: the "New search" button in the main view and the "Repeat search" button contained in each history search; Although the two buttons perform exactly the same search, we will first describe how they prepare the system to perform the search, and then go into detail about the search.

In case the user chooses the first option, the search engine view will be shown:

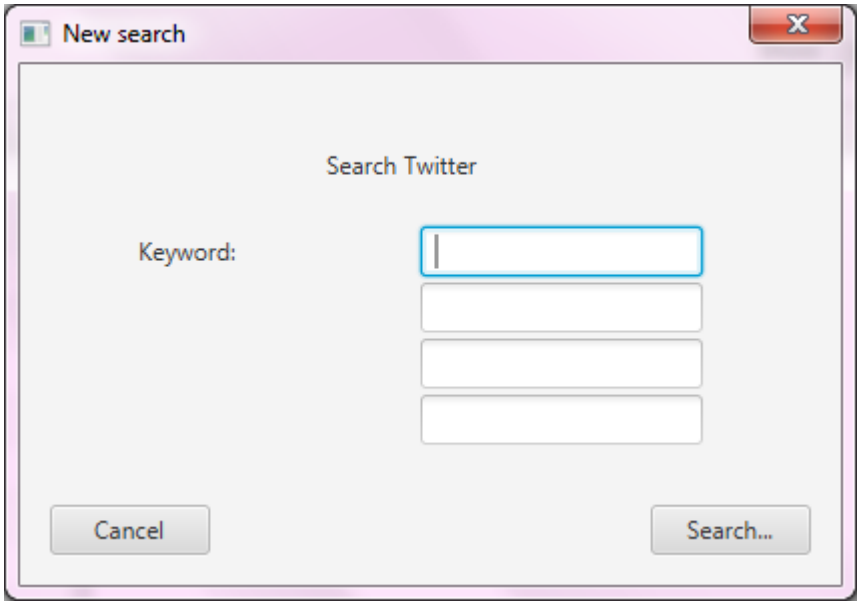


Figure 45. Search engine

In this window are provided various text fields to the user to write the keywords of interest; In principle, for the search to work, it is enough for the user to type in at least the first text box: in case of using any of the other three, the query would be constructed by chaining the content of each text box through ORs (the API of Twitter allows working with different operators, and among them this was interesting for the project).

Once the query is settled, the search will proceed creating a new collection on the database; each collection is defined by an object of the *DBCollection.java* class, which contains methods to save, extract and/or delete information from a search. Each of them will work mainly with data described in the collection table of section 2.2.

Here is included the main code of the search main engine:

```
do {
	try {
		queryResult = Main.getTwitterSessionDAO().getTwitter().search(query);
	} catch (TwitterException e) {
		if (e.getStatusCode() == UNAUTHORIZED) {
			throw new AccessException(e.getErrorMessage(), e);
		} else if (e.getRateLimitStatus().getRemaining() == 0) {
			Integer time = e.getRateLimitStatus().getSecondsUntilReset();
			time = time / 60;

			Alert alert = new Alert(AlertType.WARNING);
			alert.setTitle("RATE LIMIT FAILURE");
			alert.setHeaderText("Rate Limit searching tweets");
			alert.setContentText("You have exceeded rate limit. You have to wait "
			+ time.toString() + " seconds before continue searching.\n Please note
			that your current search will not be saved on the system.");
			alert.showAndWait();
		} else {
			throw new NetworkException(
				"You do not have internet connection. Please check it out
				before continue", e);
		}
}
```

Figure 46. Search code

After this process the search is saved in the database and then showed in the Main view.

In case the user wanted to use the second option, the system proceeds to recover the search in question generate the search directly, since with this option you want to provide the user with a fast and efficient way to repeat a previous search without the need to rewrite the query, thus avoiding confusions and errors.

The element of the Twitter API from which is extracted more information is the Tweet Object, a JSON that contains data related to a tweet. The most relevant attributes for the application are the following:

- Id → the unique identifier of the tweet to distinguish one from the other.

- Created_at → the date of creation of the tweet.
- User → is an API object that contains information relative to the user who has published the tweet; in our database this corresponds with the "author", to distinguish it from the user who downloads the tweet.
- Text → text from the tweet.
- Retweet_status → in case the tweet is a retweet, this attribute will contain the original tweet; otherwise, it will be null. This attribute is useful in the display mode "All tweets except retweets", described in section 3.5.2. of this document.

Any of the attributes of a tweet can be null (except the last one) since they are the ones that contains the essential information of the tweet [14].

On the other hand, investigating about the different APIs that Twitter has, for this project the API of major interest is the Search API which allows to the system to do historic searches: in particular this system gets access to the Standard Search API (we have chosen this mode to ensure an open source product due the others are paid).

3.7. Export

Through this functionality the user will be able to export tweets from one or more searches that he has made. The user can perform this action in two ways:

- Using the history options menu button, described in section 3.5.1.
- Using the button in the lower right corner, described in section 3.5. It is important to take into account that for this button to work, a history search must first be selected.

The exportation process was done with the "Commons CSV" library of "Apache Commons" was used, a project belonging to Apache that generates and maintains Java components [15].

The first thing that the application will do is generate the name by default of the file (later the user can change it if he wishes). Then, the application will access the database to be able to extract all the tweets related to the search and save them as ResultSet:

```
public ResultSet exportTweets() throws DatabaseReadException {
		PreparedStatement pstmt = null;
		ResultSet rsExp = null;
		try {
			pstmt = c.prepareStatement(exportCol);
			pstmt.setInt(1, id);

			rsExp = pstmt.executeQuery();

			return rsExp;
		} catch (SQLException e) {
			throw new DatabaseReadException("There was an error while
exporting the tweets from the database.", e);
		}
	}
```

Figure 47. Export tweets code

In the next step of the configuration the user can choose the name, format and destination folder of the file: for this, JavaFX includes the FileChooser library to standardize

"plataform file dialogs" in Java; first of all it is defined what type of dialogue will be opened, if to save or to open files, the title and the directory name of destination, among others:

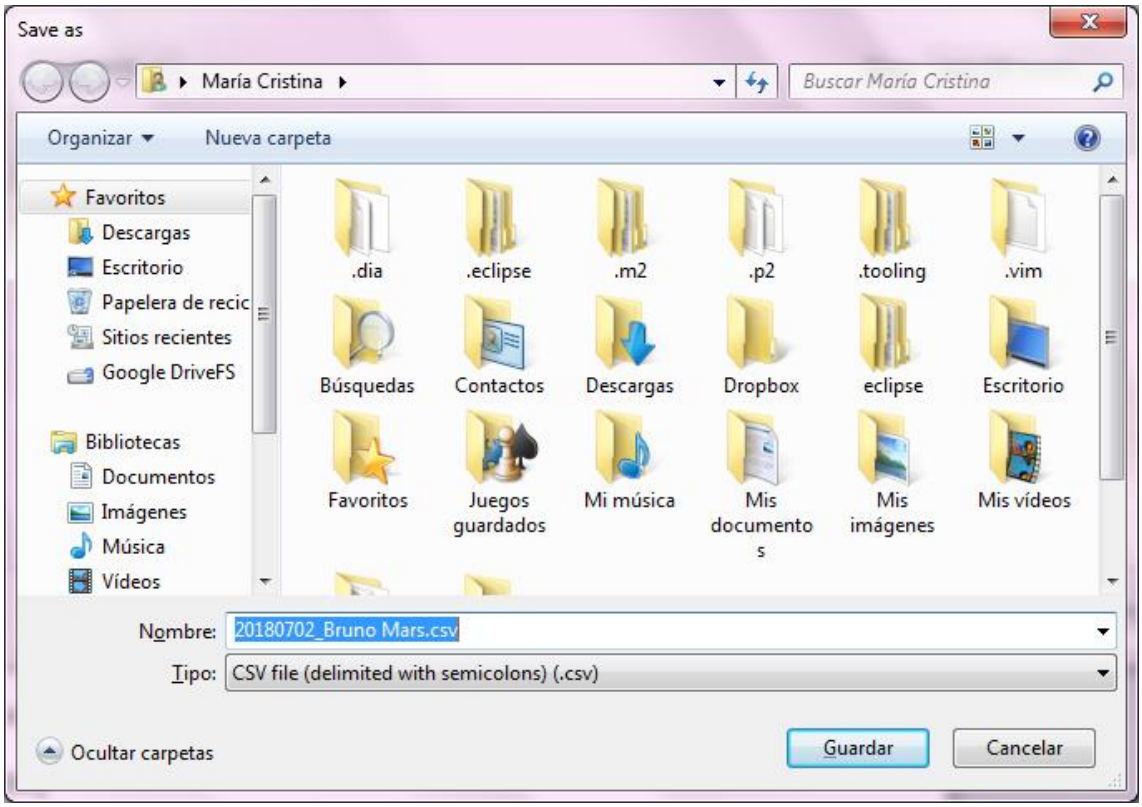


Figure 48. Export dialog

Once the dialog has been opened and the file with the desired preferences saved, the csv file will begin to be generated.

First of all, the delimiter chosen by the user will be defined (extracted from the description of the chosen format): depending on the country, the predetermined delimiter that generates the csv is different and, therefore, the visualization of the data is different; that is why two delimiters are provided so that the user can store the data in the format that is most comfortable for him: the semicolon and the comma.

Once done, the tweets are saved in the csv file as follows: first a filewriter is generated in order to generate the file; Next, we define the format in which the csv file will be written:

in this case we have chosen Excel, since it is the default program to use files in csv; here we also define the delimiter and the header:

Finally, we iterate the ResultSet to extract the data that we want to save:

- The id of the tweet (tweet_id).
- The date of creation of the tweet (created_at).
- The tweet's author (author).
- The tweet's text (text).
- The url of the tweet; it is obtained in the same way as described in section 3.5.2.

```
try {
	FileWriter fileWriter = null;
	fileWriter = new FileWriter(file);
	CSVFormat format = CSVFormat.EXCEL.withDelimiter(del).withHeader("tweet_id",
	"created_at",	"author", "text_printable", "url", "raw_tweet");
	CSVPrinter csvPrinter = new CSVPrinter(fileWriter, format);

	try {
		while (tweetsExp.next()) {
			String url = "https://twitter.com/" + tweetsExp.getString("author") +
			"/status/" + tweetsExp.getInt("tweet_id");

			csvPrinter.printRecord(tweetsExp.getLong("tweet_id"),
		tweetsExp.getString("created_at"),
			tweetsExp.getString("author"), text, url,
		tweetsExp.getString("raw_tweet"));
		}
	} catch (SQLException e) {
		csvPrinter.flush();
		csvPrinter.close();
		throw new DatabaseReadException("There was an error while exporting the tweets
from the database.", e);
	}
```

Figure 49. Print records

Once the file is saved, a dialog appears in which the application confirms that the file has been saved successfully and gives the user the option to open the file at that moment; if you press OK the computer automatically opens the file in Excel.

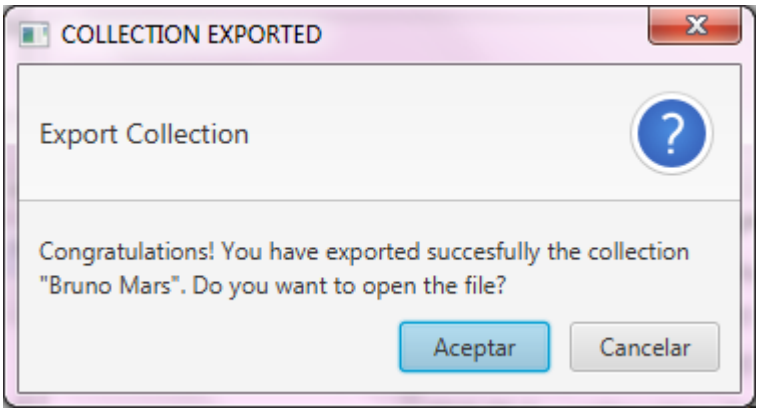


Figure 50. Expoting confirmation

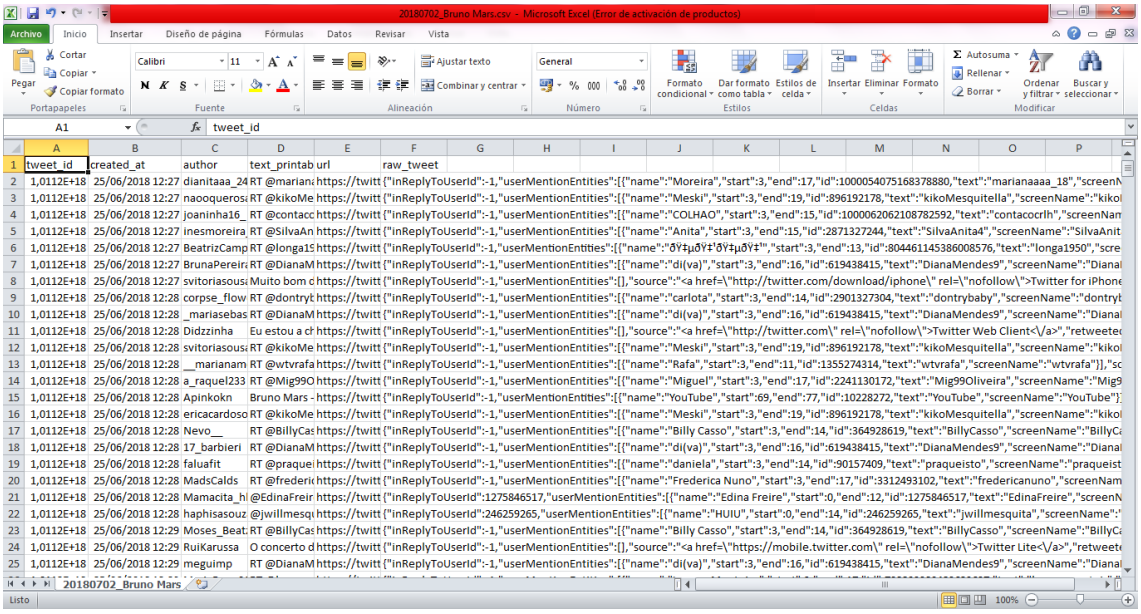


Figure 51. Excel exported

3.8. Response to database and network failures

This section explains how the errors in the project have been handled to make the user experience more enriching, being problems that can be easily solved:

CONNECTIVITY FAILURE:

The connectivity problems are mainly due to failures on the connection with the network: at the moment in which a user is connecting to the Twitter API (to authenticate or to collect tweets) and is disconnected from the internet, a warning message will appear alerting the user that he has no connection and that he should check it immediately:

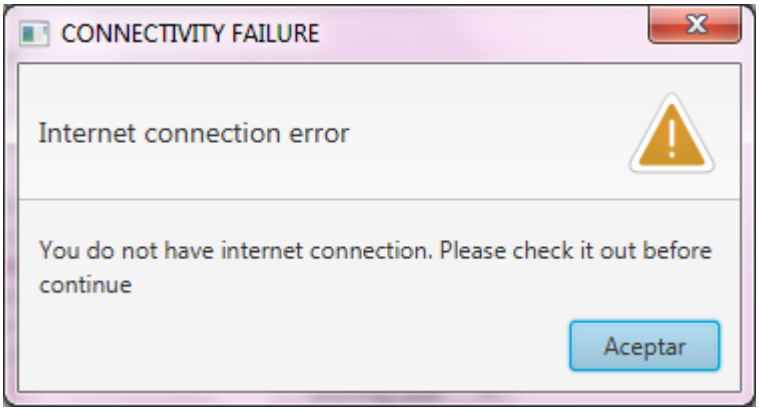
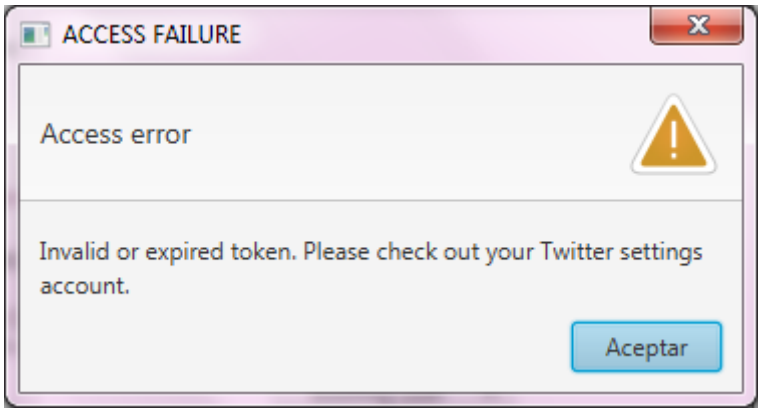


Figure 52. Connectivity failure

ACCESS FAILURE:

The access error happens when the tokens that the user uses to authenticate, either to log in or to search, are erroneous, either because they are incorrect or because they have expired; In any case, the most effective solution is for the user to check the settings of his account: this way he will be able to renew the access tokens and will be able to connect again to the application.


ACCESS FAILURE
Access error
Invalid or expired token. Please check out your Twitter settings account.
Aceptar

4. CONCLUSIONS

4.1 Conclusions

During the development of this project I learned a lot about the potential that exists in the analysis of data in social networks, what can be expected from society and how it will help us in the future.

On the other hand this project has been a great challenge, since I have discovered many new technologies that I had not used before (JavaFX or SQLite), and also it has allowed me to expand my knowledge about those I already knew (Java or Twitter4j).

In particular, JavaFX has turned out to be a very powerful tool for the development of desktop applications, both for all the possibilities it offers and for the ease that it has meant learning it.

On the other hand, SQLite has also been a tool that has turned out to be very powerful for the project by providing a serverless database, something that interested us a lot for the project; add that by also having the SQLite Studio tool, the management of the data has been extremely easy.

Finally, Twitter4j is a very powerful library with many options to get started on the Twitter APIs, with the slight disadvantage that being a non-Twitter project may find it difficult to have the code updated.

4.2 Future work

As future work it will be interesting to do usability tests to become aware of how users perceive this application and thus achieve their full potential.

On the other hand, something that may enrich the application for the future is to implement different search methods: in particular, it would be very interesting to add the collection of tweets that are being published live, in order to get tweets at the moment and thus complement the current search; For this purpose, the Twitter Streaming APIs, designed to access this data, should be studied in depth.

At the end of this document it has been included the link of the Github repository with the code of this project [16].

Bibliography

[1] https://www.simplilearn.com/real-impact-social-media-article

[2] http://www.visualcapitalist.com/internet-minute-2018/

[3]https://www.iprospect.com/en/gb/blog/why-social-media-is-essential-for-brand-marketing/#

[4] https://www.thoughtco.com/how-social-media-has-changed-politics-3367534

[5] https://irishtechnews.ie/how-social-media-has-changed-journalism/

[6] https://marketing-software.financesonline.com/compare/

[7] https://www.tiobe.com/tiobe-index/

[8] https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

[9] https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#A1141718

[10] http://www.sqlitetutorial.net/what-is-sqlite/

[11] https://developer.twitter.com/en/docs/basics/authentication/overview/application-only

[12] https://dev.twitter.com/web/sign-in/implementing

[13] https://blog.twitter.com/official/en_us/topics/product/2017/tweetingmadeeasier.html

[14] https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object.html

[15] http://commons.apache.org/

[16] Github repository of the project: https://github.com/cgg09/Search-and-Save-for-Twitter