Development of Distributed Applications
Project – Development of a distributed application integrating Bicing, Twitter and Telegram services

Bicing free slots notifier

November 2016

Davinia Hernández-Leo, David Llanos Santos

Objectives

- Build a distributed application interacting with multiple real interfaces (Bicing[1], Twitter[2] and Telegram[3]).
- Communicate two asynchronous processes.
- Work and learn real interfaces based on REST and JSON.

Description

The aim of this project is to implement a system that sends a Telegram message to user phones with the free slots of the Bicing stations the user has previously subscribed to. It also publishes in Twitter some global statistics of Bicing service. For this purpose, some external services will be used: Bicing, Twitter and Telegram.

The functionality of the application is as follows:

The main service is the **Bicing Free Slots Notifier (BFSN).** The requirements of this service are:

- Access to Bicing API periodically, every each 60 seconds, and cache the stations and their status in an internal variable.
- Allow users to subscribe to a list of Bicing stations by their phone number.
- Send a Telegram message to the users with the free slots of the stations they have previously subscribed to.

To accomplish these requirements, the BFSN must publish **a REST API** with two services producing and consuming messages in JSON format:

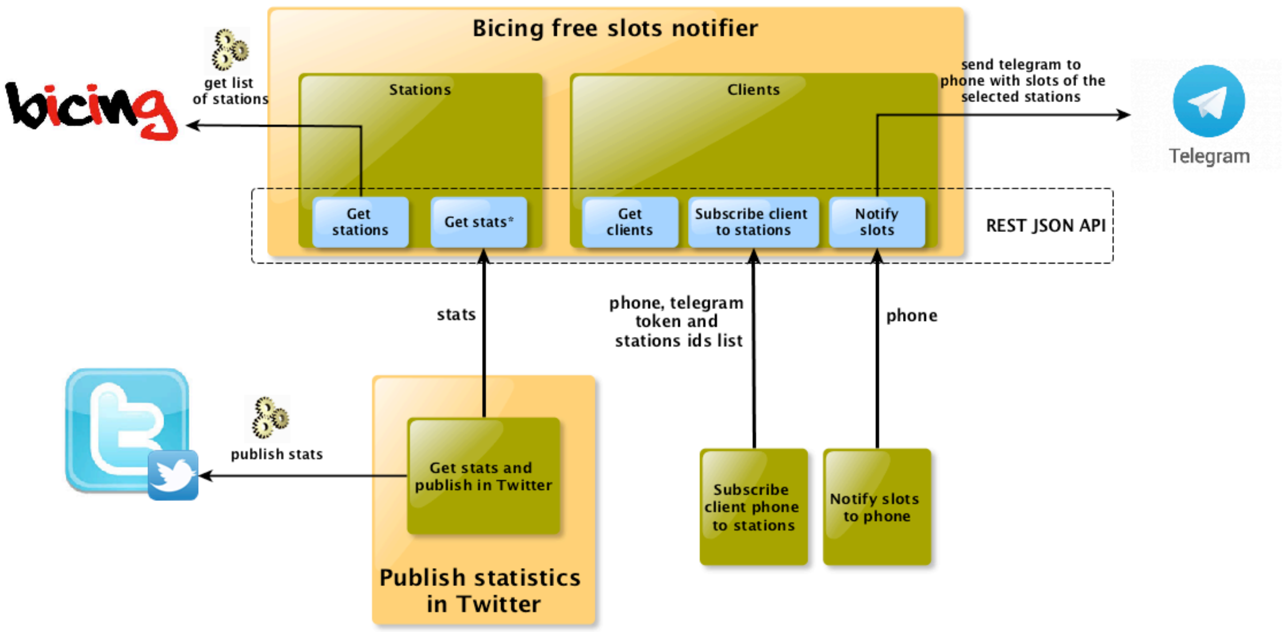
- **Stations service**, with two methods:
  - **Get stations:** Returns the stations list with the same information retrieved from the Bicing API. A request to this method must return the cached information and not make a request to the Bicing service. That is, this service will return the same data during 60 seconds, until a new asynchronous request is made by the BFSN to update the cached variable.
  - **Get stations statistics:** This method returns one or more statistics about the Bicing stations. There is not any requirement about a specific statistic, so you can propose your own ideas. Some examples: stations average occupancy, percentage of failed stations, etc.
- **Clients service**, with three methods:
  - **Subscribe client to stations:** This method is used to create users and subscribe them to Bicing stations. For this purpose, the service must receive the phone number of the client, the list of stations ids the client wants to be subscribed and the Telegram token to be able to send a Telegram message.
  - **Get clients:** Returns the list of clients subscribed to this service, including the Telegram Token of each user and the list of stations identifiers.
  - **Notify slots:** This method receives the phone number of the user and sends a telegram message with the free slots in each one of the stations the client previously subscribed to.

Another component of the application is the **Twitter publisher**. The requirement of this component is to make a request to the statistics API of **BFSN** periodically, generate a message with the response and publish it in a Twitter account created specifically for this project.

- Once this component has been started, the requests to the statistics API must be automatically and periodically made.
- The requests must not be made before the service API has updated the statistics.
- The response of the service must be processed in order to generate a human readable message. This message will be the one posted on the Twitter account.

[1] https://www.bicing.cat/
[2] https://twitter.com/
[3] https://telegram.org/

The following figure shows a diagram of the operation of the application:


Bicing free slots notifier
bicing
get list of stations
Stations
Clients
send telegram to phone with slots of the selected stations
Telegram
Get stations
Get stats*
Get clients
Subscribe client to stations
Notify slots
REST JSON API
stats
phone, telegram token and stations ids list
phone
publish stats
Get stats and publish in Twitter
Publish statistics in Twitter
Subscribe client phone to stations
Notify slots to phone

- Appendix sections should be followed to build the application.
- As a recommendation, the **BFSL** service should be implemented first. This process can easily be tested with the public REST API without implementing the Twitter publisher.

Project assessment

The assesment criteria of the project are:

- **Functionality of the application.** (25% of the mark)
- **Intermediate demo.** It will take place during the the third project session (21st and 24th of November). (15% of the mark)
- **Final demo.** It will take place during the final project session (1st and 2nd of December). (25% of the mark)
- **Structure, comments and quality of the source code.** Fault management of the application. (15% of the mark)
- **Technical report.** Maximum 4 pages. The report should include the following sections: (20% of the mark)
    - Design: Describe the design of the system, ie, how it works and what parts it has: what twitter accounts are used, what news channels are scanned, how relevant words are chosen, etc.
    - Document the REST API of BFSL: Method URL, parameters, examples of request and response with the JSON, etc.
    - Implementation: Details of how you have implemented the most relevant parts of the project using a class diagram.
    - Execution: How to run the application.
    - Conclusions.

Intermediate demo: During the third project session (21st and 24th of November), each group will make a **brief demo of the following methods of BFSL API:**

- Get stations, get stations statistics, subscribe clients to stations and get clients.
- Use any tool (web browser, poster, etc.) to make HTTP requests for these API methods.

Project delivery: Each group will deliver a single zip file including the eclipse projects with the source code and the technical report. This file will be uploaded to the activity created in Aula Global.

The file name will be the following: `dad_project_NIAstudent1_NIAstudent2.zip`

The deadline to deliver the project outcome is 30th November 2016, 20:00.

Marks behind the minimum required to pass the course will be published on 5th of December.

Appendix

Run scheduled tasks

Quartz[4] is a Java library to schedule tasks execution. You can use this library to run repetitive tasks: get tweets from Twitter, get news from Google or add hot topics found to Pocket.

Use maven to manage Quartz dependencies (as explained in Seminar 2):

```
        <!-- Quartz library -->
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz</artifactId>
            <version>2.2.1</version>
        </dependency>

        <!-- SLF4j library -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.6.1</version>
        </dependency>
```

The following sample code can help you to build a quartz job running a scheduled task:

- The first step is to create a **job** that implements the Quartz Job interface, and therefore the method `execute()`. The code inside the method will be the application logic to be executed periodically.

```
public class ScheduledTestJob implements Job {

    public void execute(JobExecutionContext context) throws JobExecutionException {
        System.out.println("Executing ScheduledTestJob at " + new Date());
    }
}
```

- In the application main you must define the job to run the periodic task and the trigger with the planning setup.

```
            // Specify the job' s details..
            JobDetail job = JobBuilder.newJob(ScheduledTestJob.class)
                    .withIdentity("testJob").build();

            // Specify the running period of the job
            Trigger trigger = TriggerBuilder.newTrigger()
                    .withSchedule(SimpleScheduleBuilder.simpleSchedule()
                    .withIntervalInSeconds(30).repeatForever())
                    .build();
```

- Finally, it is necessary to create a **scheduler using the job and the trigger to run the task.**

```
            // Schedule the job
            SchedulerFactory sf = new StdSchedulerFactory();
            Scheduler sched = sf.getScheduler();
            sched.start();
            sched.scheduleJob(job, trigger);
```

Accesing Bicing API

Bicing is a service that spreads bicycles and bike stations all over the city for the inhabitants to use. It publishes an API with the status of all stations in http://opendata.bcn.cat/opendata/en/catalog/TRANSPORT/bicing/
This service, available at http://wservice.viabicing.cat/v2/stations, gives access to the stations, location (lat-long coordinates), address (street and number), a list of the nearest stations, station status, number of parking spaces and the number of bikes, mechanics and electrics, available.

The service returns a JSON object with the list of stations, so the recommendation to parse the json returned by the service is to create two classes with the same structure as the response:

[4] http://quartz-scheduler.org/

```
	public class Stations {
		private List<Station> stations;
```

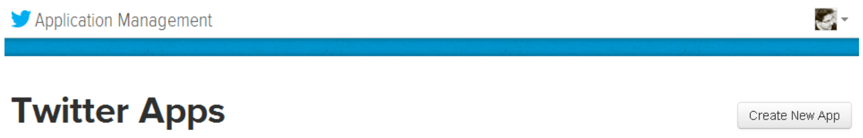
```
	public class Station {
		private String id;
		private String type;
		private String latitude;
		private String longitude;
		private String streetName;
		private String streetNumber;
		private String altitude;
		private String slots;
		private String bikes;
		private String nearbyStations;
		private String status;
```

Publish status in Twitter account

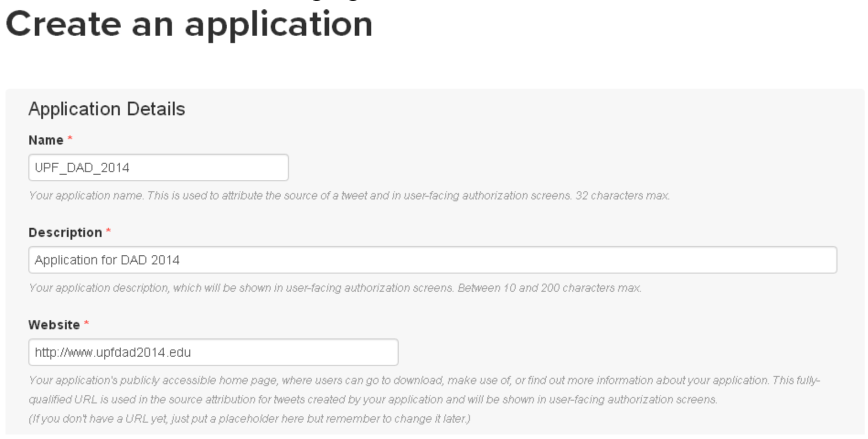
To use Twitter API, you must create a new application within the developer portal.

- Developers portal: https://dev.twitter.com/
- Create an application on Twitter. https://dev.twitter.com/apps

The following image shows the control panel for Twitter developers:



Create a new application as shown in the following figure:


Create an application

Application Details

Name *

UPF_DAD_2014

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Application for DAD 2014

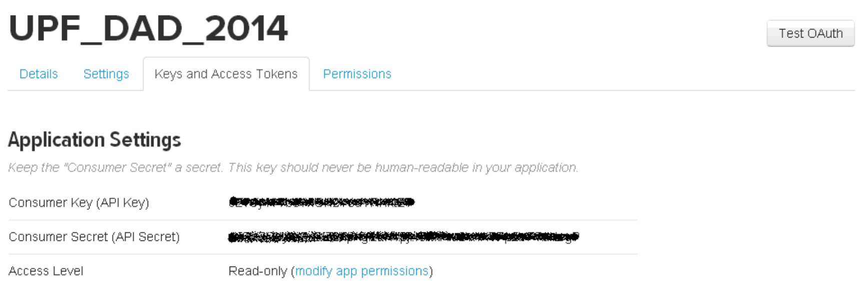
Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

http://www.upfdad2014.edu

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

After creating the application, Twitter will have generated the Consumer Key and Consumer Secret for your application. These two keys are needed to make requests to the Twitter API.



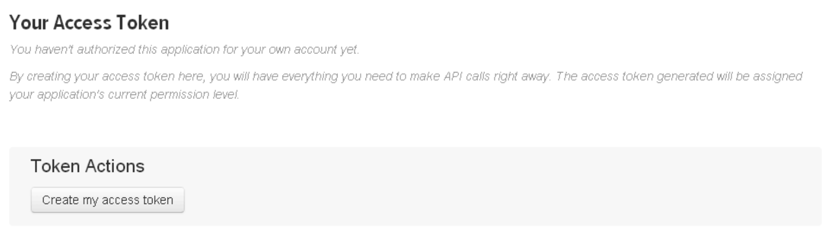
UPF_DAD_2014

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	[redacted]
Consumer Secret (API Secret)	[redacted]
Access Level	Read-only (modify app permissions)

Besides these two keys, is necessary to generate an Access Token to make requests on behalf of a user without knowing her password. To do this, we can use our own Twitter user and generate an Access Token for our account. On the same screen, below, you can create the access_token.

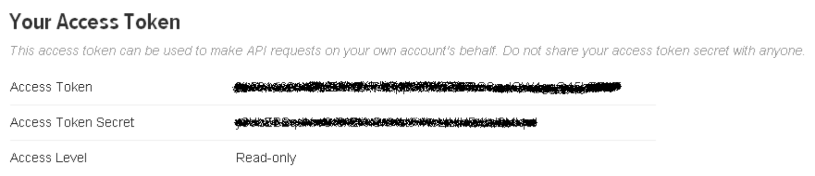


Your Access Token

You haven't authorized this application for your own account yet.

By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.

Once generated, a screen like the following appears:



Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	[redacted]
Access Token Secret	[redacted]
Access Level	Read-only

To make requests to the Twitter API, use the Twitter4J[5] library. Use maven to manage the dependency.

```
        <!-- Twitter4j -->
        <dependency>
            <groupId>org.twitter4j</groupId>
            <artifactId>twitter4j-core</artifactId>
            <version>4.0.2</version>
        </dependency>
```

The following code shows an example retrieving tweets from a channel:

```
        TwitterFactory factory = new TwitterFactory();
        Twitter twitter = factory.getInstance();
        twitter.setOAuthConsumer("CONSUMER_KEY", "CONSUMER_SECRET");
        twitter.setOAuthAccessToken(new AccessToken("ACCESS_TOKEN", "ACCESS_TOKEN_SECRET"));

        Status status = twitter.updateStatus("Status string");
```

To be able to publish messages in Twitter, it is needed to add a phone number to give "Read and write" access level to your application. Once the phone number has been added and the application created, the phone number can be deleted.

Sending items using Telegram bots

Telegram[6] is a messaging app (cloud-based instant messaging service). Telegram clients exist for both mobile and desktop systems. Users can send messages and exchange photos, videos, stickers and files of any type.

Documentation:

- Telegram bots: https://core.telegram.org/bots
- Telegram bots api: https://core.telegram.org/bots/api

To send a message using Telegram bots you must follow the next steps:

- Create a new bot using the BotFather.
    - Create a new chat with the BotFather, open this url with the Telegram app or the Telegram web client: https://telegram.me/botfather
    - Write `/newbot` in the chat and choose a name and a username for it.
    - Save the token generated for this bot
    - Open a new chat clicking on the link of the BotFather chat or opening this url with the Telegram app or the Telegram web client: https://telegram.me/username_of_the_bot
- Once the chat with the new bot has been created, we need to get the id of the chat to be able to send messages to it using the API. To get the id of the chat, make a request to https://api.telegram.org/botTOKEN/getUpdates substituting TOKEN with your token and save the property: `result.message[0].chat.id` If the output is empty, you have to write at least one message in the chat.
- With the TOKEN and the chat ID, we will be able to send messages to the chat with the API. You can use this example:
    - Create the Message object we must send. As in seminar 3, remember adding constructors, setters, getters and toString methods.

[5] http://twitter4j.org/en/index.html
[6] https://telegram.org/

```
	public class Message {
		private long chat_id;
		private String text;
```

- Then, create a client to send an HTTP POST request with that object in the body:

```
Message message = new Message(chatID, "Message");
WebTarget targetSendMessage = client.target("https://api.telegram.org").path("/botTOKEN/sendMessage");
String response = targetSendMessage.request()
	.post(Entity.entity(message, MediaType.APPLICATION_JSON_TYPE), String.class);
```