# The sentimental bot

MCS Lab 2017

v0.2, 2017-05-19

## 1. Description of the exercise

Write a bot for the Slack instant messaging system [Slack→], [Slack API→].

The bot will join a Slack channel, listen to all messages on that channel, detect messages expressing sentiments and:

1. react by sending a message with the name of the sender and an emoji
2. keep a list of {user, sentiment} and send it if asked

### 1.1. React by sending a message with the name of the sender and an emoji

For example, Marco sends a message to the channel (not mentioning the bot):

*marco:* I am happy
*sentibot:* marco is 😊

### 1.2. Keep a list of {user, sentiment} and send it if asked

For example, Marco sends a command to the bot, by mentioning its name, followed by colons (`:`) at the beginning of the message:

*marco:* sentibot: sentiments
*sentibot:* marco is happy, alice is sleepy.

### 1.3. Sentiment detection

The detection of sentiments can be very basic: if alice sends a message `I am sad`, the bot will detect and store `{alice, sad}`.

### 1.4. Operational characteristics

Mandatory: to be able to see all the bots in action *together* in class, each bot must support the following:

- Must be able to connect and operate to any Slack team.
- Must be able to connect and operate on any Slack channel.
- Must support being assigned any name, since it must be able to operate correctly also in presence of other bots.

### 1.5. Details about the Slack API to use

If you read Slack Bot Users, you will see they mention two types of bots: *custom bots* and *app bots*. The simplest approach is enough: *custom bots*.

## 2. Logistics

Lab will be done in small groups of 3 to 4 people.

You can use as dependency existing Slack libraries for Erlang like [slacker→]. On the other hand you **cannot** use an implementation of a Slack bot and customize it.

In case of doubt: "can we use this library or not?" the best thing to do is to ask as soon as possible.

Archive the project (without BEAM files) with name `sentibot-NAME.zip` (where NAME is the name of the student representing the group) and send it via email to the teacher.

Project must have a `README.md` file containing names of the students of the group, any particular instructions to use the bot and any text you think is useful to better understand your project and its features.

Project must be buildable with `rebar3 compile`.

We will run the bots all together during the next lessons, so do not wait the last moment to start coding!

## 3. Getting started

Create the project with

```
$ rebar3 new app sentibot
```

This will create file `src/sentibot.erl`. You can use that file as the API entry point. For all the other files, use the format `sb_FUNCTIONALITY.erl`, for example `sb_slack.erl` for the Slack protocol, `sb_sa.erl` for the sentiment analysis, and so on.

You will need to learn about

- OTP behaviors (gen_server and gen_statem)
- OTP supervisors
- OTP applications

(See the books and online sites in the first slide deck).

## 4. Optional

We list here some optional features you might like to implement

### 4.1. Production-ready: logging

Use the [lager→] logging system.

### 4.2. Production-ready: admin controls

Provide some basic admin controls, such as:

- support receiving Erlang messages to
  - join a specific channel
  - dump state
  - control logging
  - …

### 4.3. Production-ready: metrics

Integrate [exometer→].

### 4.4. Sentiment analysis

Research about the field of [sentiment analysis→] and add this capability to the bot.

## 5. Change log

v0.1: Initial version
v0.2: Specify which Slack bot to build