Arduino
Facebook
Notifier

Intel Galileo Facebook Notifier

Militaru Cristian

Group 30431

Table of Contents

1 Problem statement

The problem requires the Intel Galileo board to connect to the Internet and check if there is any post on the user's profile in the last hour. If there is a new post, then the LED on the board is turned on.

2 Requirements

2.1. Hardware requirements

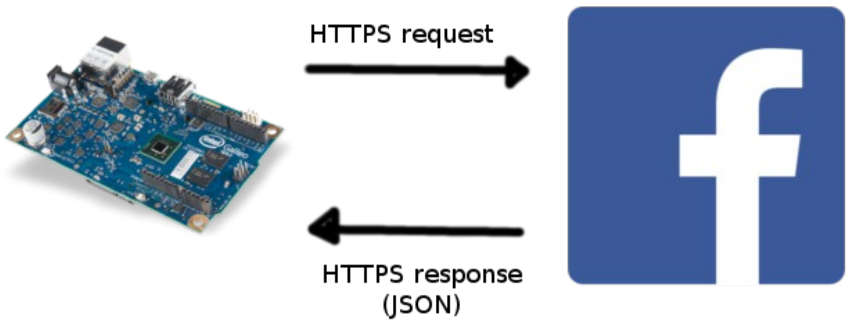
- Intel Galileo board with a connection to the Internet;
- A microSD card for storing the Linux operating system on the board;
- A computer connected to the Internet;
- An USB cable for serial communication between the computer with the board.

2.2. Software requirements

The software resources can be downloaded from the Intel's website: https://software.intel.com/en-us/iot/hardware/galileo/downloads. The installation steps are also available here: https://software.intel.com/en-us/get-started-galileo-linux-step1.

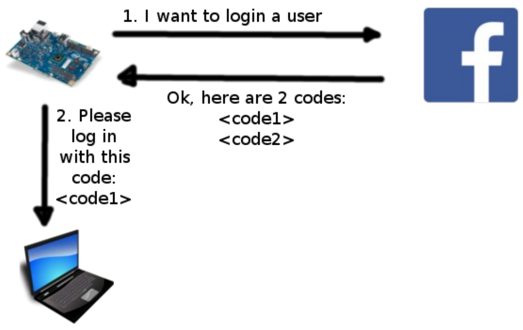
- Firmware updater (should be run once, for the board to work correctly);
- Arduino IDE for Intel Galileo (on the PC);
- The Intel Galileo Board microSD Card Linux Operating System Image (on the microSD card, for the board);
- Libraries: wolfSSL (more instructions will be provided later).

3 The Facebook API



The board can exchange information with the Facebook through a HTTPS based API. The board connects to the Facebook server and sends data about the required information. In this case, there are several steps to find if there is a post on the user's profile in the last hour. A high-level overview of these steps are:

- Login with a Facebook account – since the board has neither a keyboard nor a browser, the login process must use a computer [1]. The following sequence of actions are required for a login process:
  - The board tells Facebook it needs to login a user. For that purpose, two codes are returned by the Facebook API – a code that is shown to the user (through the Arduino serial console), and one that the board should store internally to refer to this login request.


1. I want to login a user
Ok, here are 2 codes: <code1> <code2>
2. Please log in with this code: <code1>

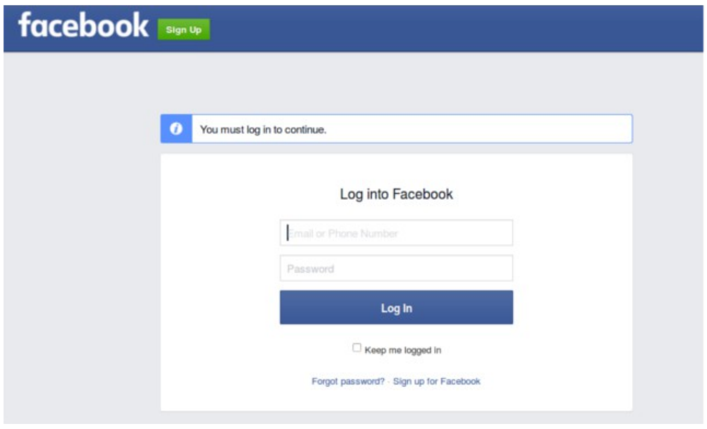
Facebook API request URL:

```
POST https://graph.facebook.com/oauth/device?
type=device_code&client_id=<APP_ID>&scope=<COMMA_SEPARATED_PERMISSION_NAMES>
```

Response:

```
{
      "code":"7e4f651356978e572edc2daf8f1673c4",
      "user_code":"ZSM7T4IJ",
      "verification_uri":"https:\/\/www.facebook.com\/device",
      "expires_in":420,
      "interval":5
}
```

- The user enters the www.facebook.com/device website from a computer. If there is no one logged in, a login form appears.



facebook Sign Up

You must log in to continue.

Log into Facebook

Email or Phone Number

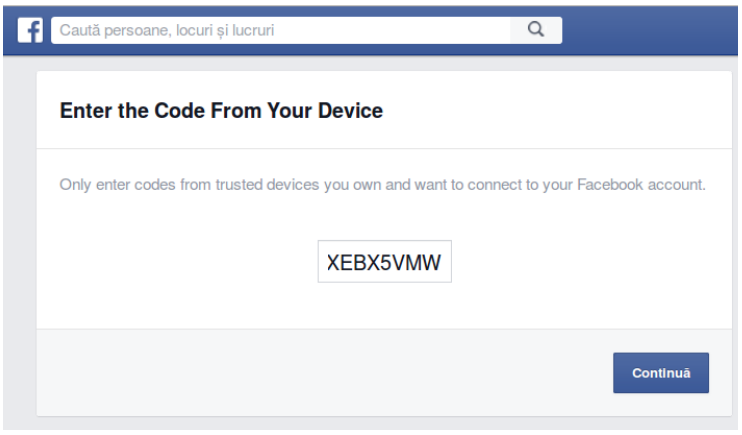
Password

Log In

Keep me logged in

Forgot password? · Sign up for Facebook

- Then the user and enters the code received from the board.



Caută persoane, locuri şi lucruri

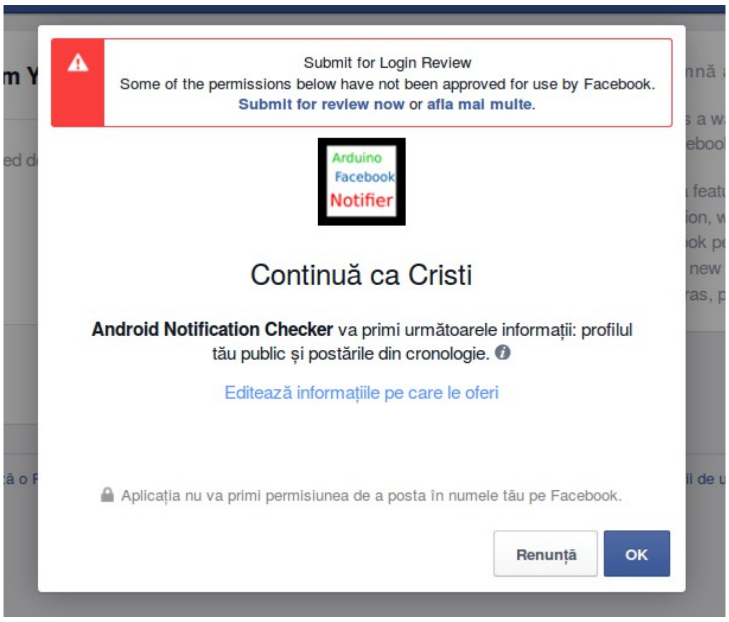
Enter the Code From Your Device

Only enter codes from trusted devices you own and want to connect to your Facebook account.

XEBX5VMW

Continuă

- Finally, the user grants the application the permission to access his/her posts.



Submit for Login Review
Some of the permissions below have not been approved for use by Facebook.
Submit for review now or **afla mai multe**.

Continuă ca Cristi

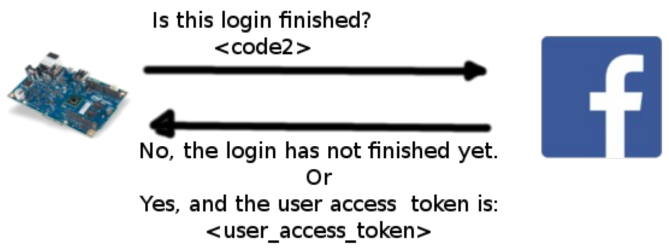
Android Notification Checker va primi următoarele informații: profilul tău public și postările din cronologie.

Editează informațiile pe care le oferi

Aplicația nu va primi permisiunea de a posta în numele tău pe Facebook.

Renunță OK

- In the meantime, the board is checking if the user has logged in, by sending the second code (the one stored internally, not the one shown to the user) to Facebook. If the user has not logged in yet, Facebook returns an error and the board waits for a few seconds before trying again. If the user has logged in, then an User Access Token [2] is provided. This is a character string that identifies the user that is logged in.


Is this login finished?
<code2>
No, the login has not finished yet.
Or
Yes, and the user access token is:
<user_access_token>

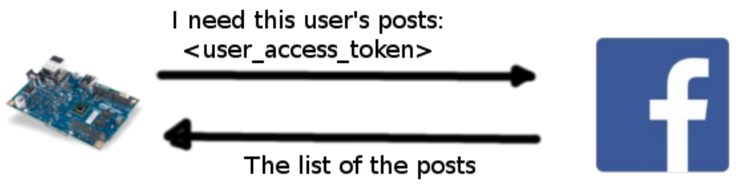
Facebook API request URL:

```
POST https://graph.facebook.com/oauth/device?
type=device_token&client_id=<APP_ID>&code=<LONG_CODE_FROM_FIRST_STEP>
```

Response:

```
{
      "access_token":"CAALYP...[more characters here]...EqwZDZD",
      "expires_in":5176762
}
```

- Retrieving the last post on the user's page. It is not possible to retrieve only the last post, but we can get a list all of them. Actually, the Facebook API divides the results into pages to reduce the traffic, so we will only receive a page consisting of a few posts. Apparently, the posts are sorted in descending order of the created time. Then we only need the first post in the list. The request from the board to Facebook must contain the User Access Token, so that Facebook knows whose posts it should return.



Facebook API request URL:

```
GET https://graph.facebook.com/v2.5/me/feed?access_token=<USER_ACCESS_TOKEN>
```

Response:

```
{
   "data": [
      {
         "message": "Post title",
         "story": "Post description",
         "created_time": "2016-02-27T11:56:17+0000",
         "id": "829903667155946_825599620919684"
      },
      {
         "message": "Another post title",
         "story": "Another post description",
         "created_time": "2015-12-29T00:22:44+0000",
         "id": "829903667155946_794293377383642"
      },
      ...[more similar entries]...
   ],
   "paging": {
      "previous": "https://graph.facebook.com/...[an URL to the previous page of posts]",
      "next": "https://graph.facebook.com/...[an URL to the next page of posts]"
   }
}
```

4 Goals

The program can be divided in two big sections: networking (connection to the database) and string parsing (for the JSON responses).

For the networking side, there is an web client implemented in the Arduino library. That allows connecting to the Facebook server via the HTTP protocol. A sample program that searches the term “arduino” on Google can be found on their website: https://www.arduino.cc/en/Tutorial/WebClient.

There is a problem: the Arduino library has no support for Secure Sockets Layer (SSL), which is a must when connecting to the Facebook API. Without SSL, all the requests to the Facebook API are errors (thus no data can be sent/received). Luckily, Intel Galileo has a Linux operating system alongside with the Arduino environment. This can power more advanced libraries, such as wolfSSL. Here is a guide for setting up the library:
https://software.intel.com/sites/default/files/managed/ab/55/Bringing%20SSL%20to%20Arduino%20on%20Galileo.pdf

Note: The Facebook server requires some cryptographic algorithms that are not installed by default. I got it to work by using the following command when configuring WolfSSL:

```
./configure --prefix=$HOME/wolfssl_try3  --enable-ecc --enable-supportedcurves
C_EXTRA_FLAGS="-DWOLFSSL_STATIC_RSA" --host=i586-poky-linux-uclibc
--target=i586-poky-linux-uclibc
```

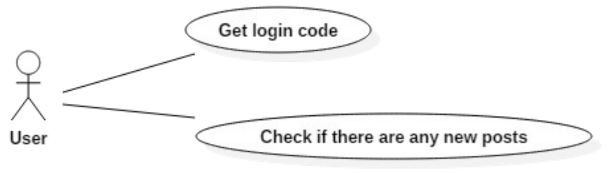
For the parsing side, the JSMN library (https://github.com/zserge/jsmn) can be used. It splits the JSON string into tokens, which can be objects, lists or atoms (mostly integers and strings).

In conclusion, the goal is to implement the following components:

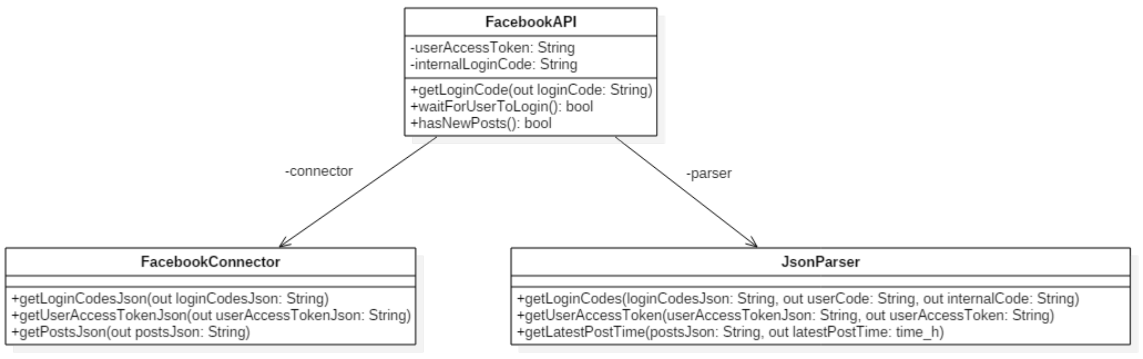
- A networking component, capable of HTTPS requests.
- A parsing component, for retrieving the relevant information from the JSON responses.
- A high-level component integrating those two.

5 Design

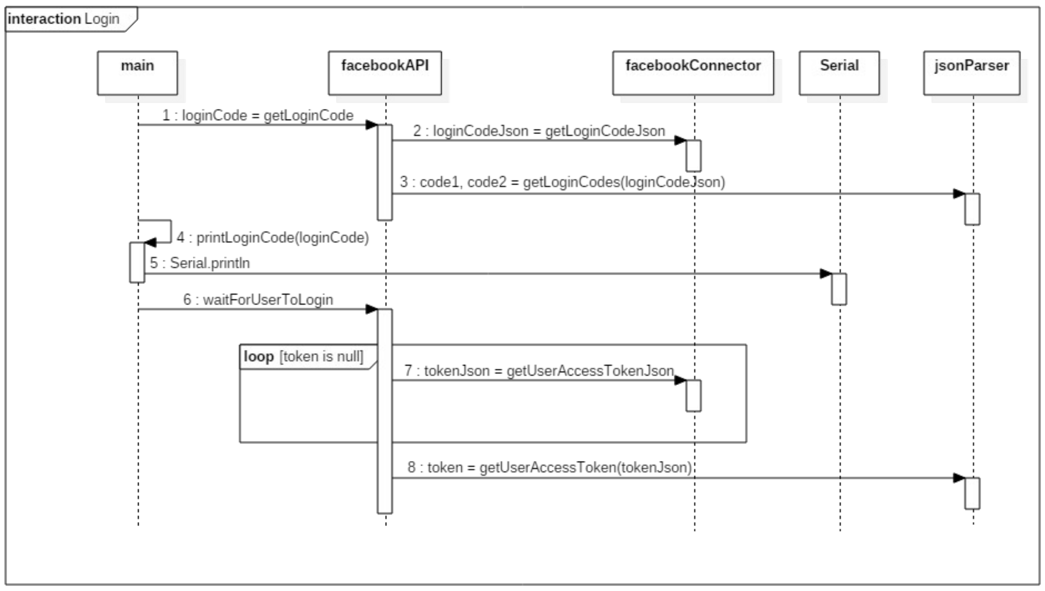
5.1. Use Case Diagram



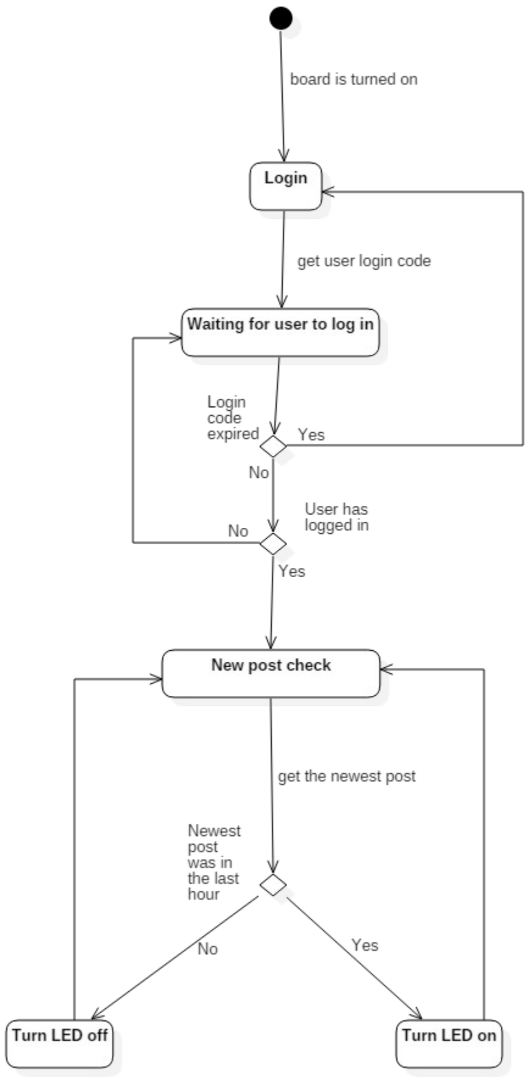
5.2. Class Diagram


FacebookAPI
-userAccessToken: String
-internalLoginCode: String
+getLoginCode(out loginCode: String)
+waitForUserToLogin(): bool
+hasNewPosts(): bool
-connector
-parser
FacebookConnector
+getLoginCodesJson(out loginCodesJson: String)
+getUserAccessTokenJson(out userAccessTokenJson: String)
+getPostsJson(out postsJson: String)
JsonParser
+getLoginCodes(loginCodesJson: String, out userCode: String, out internalCode: String)
+getUserAccessToken(userAccessTokenJson: String, out userAccessToken: String)
+getLatestPostTime(postsJson: String, out latestPostTime: time_h)

5.3. Login Sequence Diagram


interaction Login
main
facebookAPI
facebookConnector
Serial
jsonParser
1 : loginCode = getLoginCode
2 : loginCodeJson = getLoginCodeJson
3 : code1, code2 = getLoginCodes(loginCodeJson)
4 : printLoginCode(loginCode)
5 : Serial.println
6 : waitForUserToLogin
loop [token is null]
7 : tokenJson = getUserAccessTokenJson
8 : token = getUserAccessToken(tokenJson)

5.4. State Chart Diagram



6 Development

Let's start with the main functions of the Arduino language: **setup** and **loop**.

In the setup section, we prepare the application. For simplicity, the Facebook login process is done in the setup function (because it is required only once). Also, the current time is retrieved in here (it will be compared to the time of the last post on Facebook).

In the loop function, the program repeatedly requests Facebook to send the most recent posts of the current user. If the most recent post is within the last hour, then the on-board LED is turned on.

The following sections will present how the components of the application accomplish the sub-tasks of the application:

- Connection
    - HTTP connection
    - HTTPS connection
    - Proxy support
- HTTP response parsing
    - Header parsing
    - Content parsing
        - Construction of the content from chunks
        - JSON parsing
- Time arithmetic

6.1. Connection

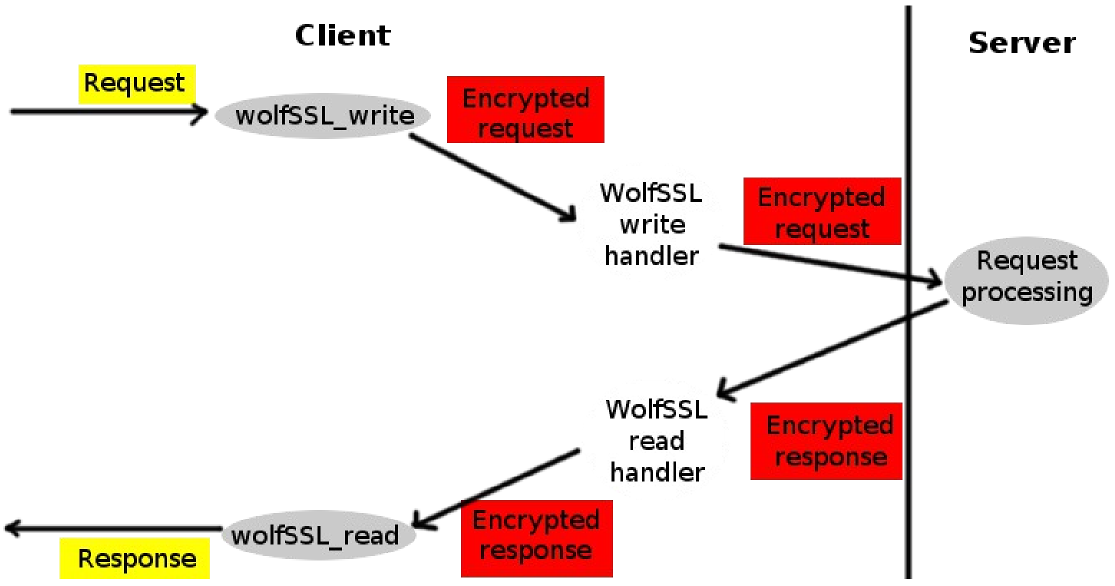
The HTTP/HTTPS connection is implemented in the *SSLConnector* class. It is a wrapper around the *EthernetClient* library provided by Arduino. The SSL Connector has a higher-level of abstraction, providing the following functionalities:

- Sending requests via HTTP and reading the response. This is done by simply using the *EthernetClient* library (like here). A special use-case is when there is a network proxy between the client (the Galileo board) and the server (in our case, the Facebook API server and the time API). The “proxy” sub-section will give further details in this case.

- Sending requests via HTTPS and reading the response. WolfSSL comes to rescue in this case. It has a simple interface: instead of sending the HTTP request by using *EthernetClient*, we give the **request** to WolfSSL as a parameter to the *wolfSSL_write* function, which will **encrypt** the message and send the encrypted message as a parameter to a handler defined by us. Our handler uses the *EthernetClient* library to send the encrypted data to the server.

  The **server** processes the request and returns an encrypted response through the connection (again, using *EthernetClient*).

  The response is collected in a buffer when the user calls the *wolfSSL_read* function. The receiving and collecting the response is done inside a handler (WolfSSL calls our method to read the encrypted data). The handler returns the encrypted response to WolfSSL, which **decrypts** it, and returns it.



WolfSSL function calls and data flow

- Luckily, proxies have nice HTTP interfaces. If the client sends the proxy a request which looks like this: "`CONNECT <servername>:<port>`", then the proxy will forward all the information that follows to the server without any interference, as if it would not exist. Therefore, the request to any server will consist of the following steps:

  1. Connect to the proxy;
  2. Send the CONNECT request to the proxy;
  3. Send the actual request to the server.

  This way, the proxy has no idea what data is being sent. There may be other options available, but this one seems the simplest.

  For HTTP requests, there is an alternative. Instead of connecting to the server as if there would be no proxy (and send the request "GET /some/relative/path"), connect to the proxy and send an absolute request to the server ("GET http://servername/some/relative/path"). Therefore, instead of sending.

6.2. HTTP response parsing

After the response is received, we need to retrieve the information we need from it. A response consists of three parts:

1. The status line
2. The headers
3. The content

The **status line** looks like this:

```
HTTP-Version Status-Code Status-Phrase
```

For example:

```
HTTP/1.1 200 OK
```

The response code is important for checking if there was any error received from the server.

The headers are pairs of the form “Attribute-Name: Value”. For example:

```
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

Most of the headers are ignored. The only exception is "`Transfer-Encoding: chunked`". If we see this header, it's going to be a long night. Actually, it's not such a big deal. It says that the content is split into chunks. The following will explain more about chunks.

The response is in JSON format. JSON is an encoding that can represent: objects, lists and atoms.

Let's start with atoms. In JSON, there are three kinds of atoms: numbers, booleans and strings. In our application, all atoms are treated as **strings** (the number `1234` will be represented as `“1234”`, the boolean `true` will be represented as `“true”`).

The lists represent **enumerations** of elements, enclosed between square brackets: [...]. The elements can be atoms, objects, or even other lists. We encountered a list when retrieving the Facebook posts:

```
{"data":
      [
           {"story":"Post story","created_time":"2016-04-
23T08:38:05+0000","id":"..."},
           {"message":"Post msg","story":"Post
story","created_time":"2016-02-27T11:56:17+0000","id":"..."}
           ...
```

```
        ]
}
```

Here, the “data” field is an array of objects.

The **objects** are key-value pairs enclosed in curly brackets: `{ key1: value1, key2: value2, …}`. The key is a string, and the value can be any element (atom, list or other object). An example of object can be seen above. All the data is enclosed in an object, containing the “data” field, whose value is a list. The list contains objects. Each object in the list describe a Facebook post.

A **chunked** response is only a little bit different: the response is not in one piece. The response consists of more chunks. To help parsing them, the response has the following format:

```
CHUNK_1_SIZE

chunk_1 data

…

CHUNK_N_SIZE

chunk_n data

0
```

The chunk sizes are **hexadecimal** numbers that express the number of bytes of the following chunks.

The chunks are used for optimization: the server does not have to give all the content at a time. It can process a part of the response and send it as soon as it is ready. Thus, the client does not have to wait for the server to complete the full request.

In this project, the chunks are collected in a character buffer, and the full content is returned.

6.3. JSON Parsing

The JSMN library offers a low-level API to parsing JSON. It splits a JSON string (as the ones from above) into **tokens**. Each token is a **sub-string** of the full JSON that has one of the following types: object, list or atom. For example:

{ “name”: “Cristi”, “age”: 21 }

The tokens in this case are:

OBJECT (the “{” character)

    ATOM(“name”) ATOM(“Cristi”) ATOM(“age”) ATOM(“21”)

For objects and lists, the JSMN library returns also the number of contained elements. For atoms, the JSMN library gives the start index and end index in the full JSON string.

In this project, the JSMN library is extended with a higher-level interface. The *JsonParser* class makes it possible to parse JSON elements and retrieve specific fields, in case of objects, or a specific element, in case of lists.

6.4. Time arithmetic

In order to check if the last Facebook post was recent, we need time functions. The time is represented as UNIX timestamps (long integers, representing the time that has passed from the epoch – 01/01/1970 – in seconds). The current time is computed using the timezonedb API (we get the current time at the start of the application), and the millis() Arduino function (to see how many seconds have passed since the moment of time retrieval). The time of the last Facebook post is parsed from a string having this format: “2016-04-23T08:38:05+0000”, and converted in an UNIX timestamp. Next, we need to add the GMT offset (the time given by Facebook is in GMT+00), which is retrieved from the timezonedb API at the same moment when the time is fetched.

Now, the “recent” check is trivial: subtract the Facebook post timestamp from the current timestamp, and if it is smaller than 3600 (one hour), then the post is recent.

7 Testing

Testing the connections is done manually, in environments with proxy and without proxy in the network.

For JSON parsing, there is a small test for the tokenizer inside the JsonParse.cpp file.

8 References

Parsing time strings:
http://linux.die.net/man/3/strptime

Time API:
https://timezonedb.com/api

Intel Galileo WolfSSL setup:
https://software.intel.com/sites/default/files/managed/ab/55/Bringing%20SSL%20to%20Arduino%20on%20Galileo.pdf

Arduino http client:
https://www.arduino.cc/en/Tutorial/WebClient