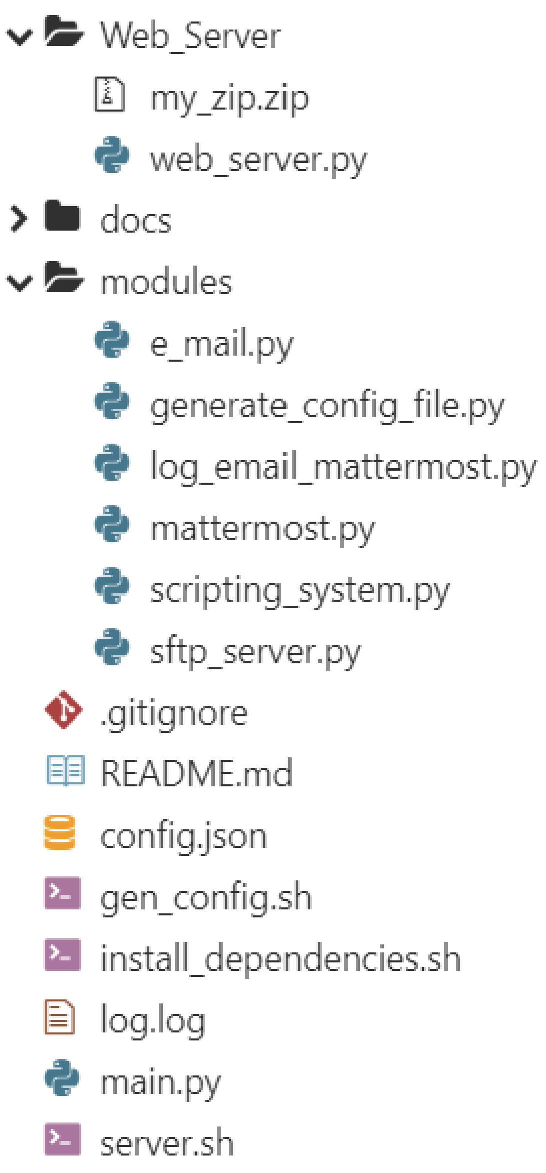
# Scripting System

## Summary

## Project Architecture

This project contains 3 folders and the root folder.



```
Web_Server
    my_zip.zip
    web_server.py
docs
modules
    e_mail.py
    generate_config_file.py
    log_email_mattermost.py
    mattermost.py
    scripting_system.py
    sftp_server.py
.gitignore
README.md
config.json
gen_config.sh
install_dependencies.sh
log.log
main.py
server.sh
```

In the folder named “Web_Server”, there are files linked with Web server. In fact, there is python script that launch web server. The mount point for this server is chosen to be the folder Web_Server. Thus, in this tree, files enabled on web server are “web_server.py” and “my_zip.zip”, on the ground root for web server is “/Scripting-System/Web_Server/”.

In the folder docs, there are all PDF documents such as technical document and installation document.

The folder named “modules” contains python script used by “main.py” script.

- “e_mail.py”: script that configures and manages e-mails send. Called by “log_email_mattermost.py” module.
- “generate_config_file.py”: script to regenerate configuration file template.
- “log_email_mattermost.py”: script that manages all logs, such as log file, call “e_mail.py” module and manages Mattermost notification by calling “mattermost.py” module.
- “mattermost.py”: manages Mattermost notification. Called by “log_email_mattermost.py” module.
- “scripting_system.py”: manages the first part of the program, it means it gets configuration, requests file on web server and operates on zip file, such as compares dates, unzips, creates a .tgz archival …
- “sftp_server.py”: manages the second part of the program. It creates a connection with sftp server check archival and send .tgz file to sftp server.

All files in “modules” folder are files unused by user, it is useless and risky to print them in the root folder of the project.

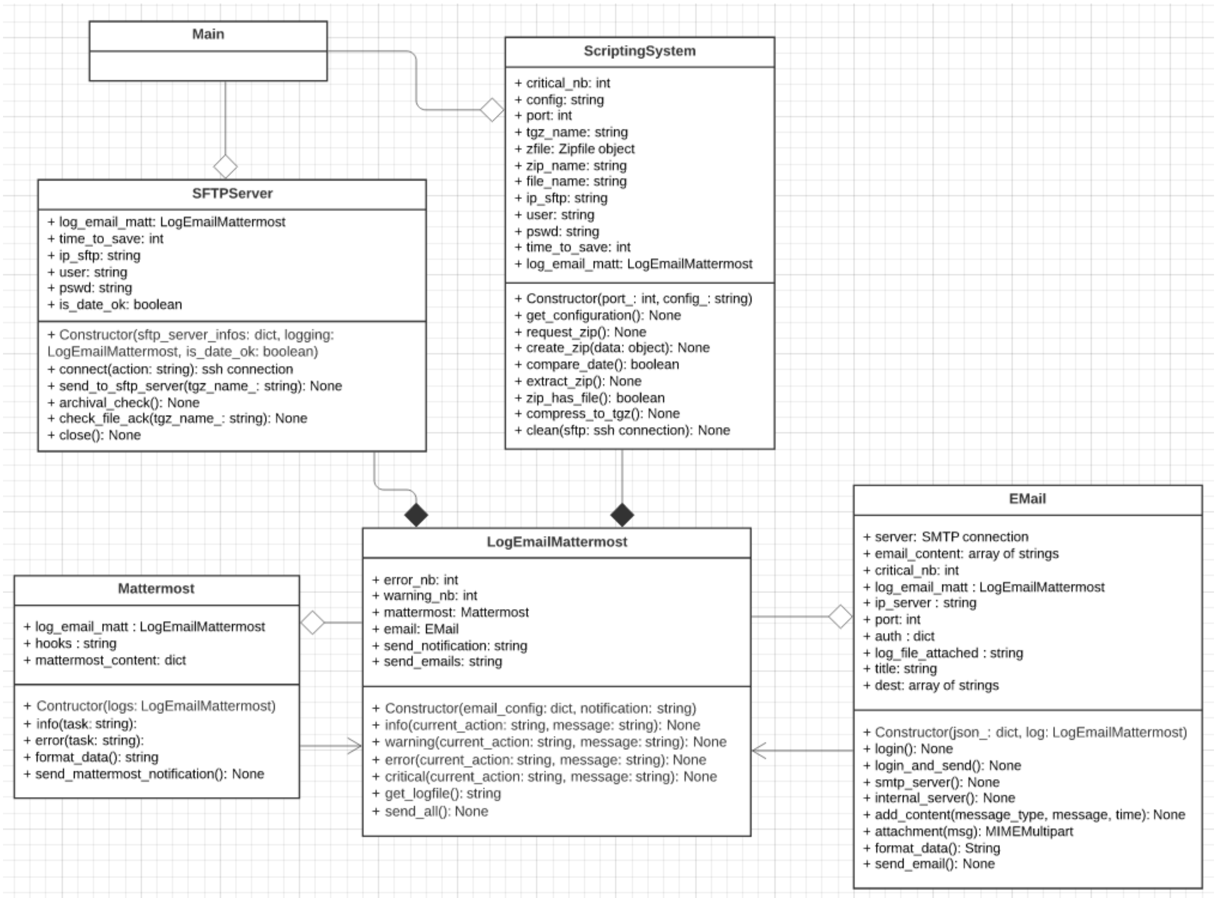
The root folder contains bash files, log file, configuration file and “main.py” script.

- “.gitignore”: standard gitignore file for my project, do not upload __pycache__ …
- “README.md”: classic README presents the project.
- “config.json”: is a file that can be modified by user. This one will configure program. This file is read by main.py script to pass configuration to program. I choose a json format, to minimize the risk of error in the program. This way configuration returns only 2 errors type: JSONDecodeError if file is not at json format or if a compile error occurred and KeyError if user forget to specified and key such as “ip” …
- “log.log”: it is a standard log file to print infos, warnings, errors and criticals (such as configuration reading with JSONDecodeError or KeyError).
- “main.py”: main of the program that call modules and concatenate everything in the same file.

- "gen_config.sh": bash file to execute modules/generate_config_file.py. User does not deal with python file but only with bash file to regenerate configuration file template.
- "install_dependencies.sh": bash file to install pip dependencies and create cron. I choose to create a file such as this one to simplify configuration for user. When user downloads project, user has just to run this file and everything is configured. If any problem occurred, he has possibility to install manually as well. By this way, all files are locked and user cannot modify files. Ha cannot improve code but he cannot break it too.
- "server.sh": launch web server at the good mount point.

I choose to put all files used by user in the root folder to be easily accessible by this one. This way, user does not need to search across many files to find configuration file for example. All other files used in the background by program are in a specific folder not accessible by standard user (see rights and chmod).

## Project Structure


Main
ScriptingSystem
+ critical_nb: int
+ config: string
+ port: int
+ tgz_name: string
+ zfile: Zipfile object
+ zip_name: string
+ file_name: string
+ ip_sftp: string
+ user: string
+ pswd: string
+ time_to_save: int
+ log_email_matt: LogEmailMattermost
+ Constructor(port_: int, config_: string)
+ get_configuration(): None
+ request_zip(): None
+ create_zip(data: object): None
+ compare_date(): boolean
+ extract_zip(): None
+ zip_has_file(): boolean
+ compress_to_tgz(): None
+ clean(sftp: ssh connection): None
SFTPServer
+ log_email_matt: LogEmailMattermost
+ time_to_save: int
+ ip_sftp: string
+ user: string
+ pswd: string
+ is_date_ok: boolean
+ Constructor(sftp_server_infos: dict, logging: LogEmailMattermost, is_date_ok: boolean)
+ connect(action: string): ssh connection
+ send_to_sftp_server(tgz_name_: string): None
+ archival_check(): None
+ check_file_ack(tgz_name_: string): None
+ close(): None
LogEmailMattermost
+ error_nb: int
+ warning_nb: int
+ mattermost: Mattermost
+ email: EMail
+ send_notification: string
+ send_emails: string
+ Constructor(email_config: dict, notification: string)
+ info(current_action: string, message: string): None
+ warning(current_action: string, message: string): None
+ error(current_action: string, message: string): None
+ critical(current_action: string, message: string): None
+ get_logfile(): string
+ send_all(): None
Mattermost
+ log_email_matt : LogEmailMattermost
+ hooks : string
+ mattermost_content: dict
+ Contructor(logs: LogEmailMattermost)
+ info(task: string):
+ error(task: string):
+ format_data(): string
+ send_mattermost_notification(): None
EMail
+ server: SMTP connection
+ email_content: array of strings
+ critical_nb: int
+ log_email_matt : LogEmailMattermost
+ ip_server : string
+ port: int
+ auth : dict
+ log_file_attached : string
+ title: string
+ dest: array of strings
+ Constructor(json_: dict, log: LogEmailMattermost)
+ login(): None
+ login_and_send(): None
+ smtp_server(): None
+ internal_server(): None
+ add_content(message_type, message, time): None
+ attachment(msg): MIMEMultipart
+ format_data(): String
+ send_email(): None

UML Diagram of Scripting system program

## Explanations

I choose to separate my program in different modules as described in the architecture.

## First part: ScriptingSystem

First, class main call “ScriptingSystem” object. This permits to get configuration set by user in config.json through get_configuration() method . Each error (JSONDecodeError & KeyError) when user wrote in config.json is cached by the program and return an info in log file.

Then, if configuration was correctly set, program request zip file on web server. This request is set in request_zip() method. In this method I used urrlib3 library to create a get request. This library use SSL certificate to authenticate on web server. Transmitted data are encrypted, that secure the connection. The program catches all errors can happened during request such as Timeout, Error 404 …

If program received a 200 status code response from the server, then it can unzip zip file through extract_zip() method. If data is not readable, then program will notice that in log. To operate on zip file, I choose to use “zipfile” library which is very efficient and easy to use.

Know the program has to check if the downloaded zip has the dump file and if this one was modified.

To do this, there is a method called “zip_has_file()” that return true if dump file in zip file and else false. Now, if zip contains dump file, the program has to check if it was modified today. To do this the method “compare_date()” return a value to know if the last modification date of dump file was today or not.

Now the program has to compress dump file into .tgz format and with the name YYYYDDMM of the current date of processing. This operation is done in method called “compress_to_tgz()”. In python, there is a library called tarfile can compress files into tar format. Unfortunately, I could not fit with this library so I decided to tar dump file via terminal. To do this, I use os.system() method and operate standard command to tar file: “tar -czf “<my_path>/<my_tar> “my_file”””.

- -c: Create an archival
- -z: compress archival with gzip to have a .tgz archival.
- -f: location of archival

Compression was resolved with terminal, then program cannot have a precise reason of an error is this one occurred while compressing. Program just knows an error occurred.

Then, the last method “clean(sftp)” allows to clean everything, such as unwanted files or something else. Its closes sftp connection too.

Now in the main, program zip has dump file and this one is extracted. The program is not going to use ScriptingSystem object anymore because there is no more connection with web server or things to operate on zip and dump file.

## Second part: SFTPServer

For my program, I choose to use SFTP connection. This type of connection uses just one port instead of FTPS connection that uses many. It uses SSH connection that is why user has to create public/private pair keys to use this program. Moreover, connection is always secured, and ssh connections permit to do a lot of operations. The harder in ssh connection is to manage keys. FTPS connection is very hard to combine with firewalls on the ground this kind of connection requires a DATA channel that uses many ports.

To do this, let's use pysftp python library. This library is simple to use and powerful.

Let's instantiate a SFTPServer object by passing it information about connection, such as server ip, username and password to connect to and time to store files on server. These information were collected in ScriptingSystem class in method "get_configuration()". Moreover, program passes Boolean about dates to SFTPServer constructor too. Constructor make sure all information are available and automatically created a new connection with server via "connect()" method.

Connect method manage ssh keys. In fact, it read the current known hosts. If server public key is not known, then create a connection and automatically add its rsa key to known hosts. Then each time program runs, server is a known host, so data is encrypted. Moreover, this class catches all exception about connection and report it in logs.

Now connection is established, so program can check archival. SFTPServer class get all files on server and check at their name (date of compression). If file is expired, then remove it from sever.

Script can now send tgz file to server. If dump file was modified today then post it on server via "send_to_sftp_server()" method. This method uses pysftp library to move to "TSE-INFORX" folder and posts file on server and close the connection. If local/remote path does not exist, then report it in logs. To be sure, file was correctly posted, an ACK is send and check on server the file by recreating a ssh connection. Error are reported too in logs.

## LogEmailMattermost

In the specification we had to implement options to send e-mail an Mattermost notification in addition of traditional logs. Logs report all infos, warnings and errors in log file and permit to easily identify the answer. The aim of Mattermost notification and e-mails is to print a recapitulation of the daily process.

To manage these 3 entities, I decided to create a general class to implement information in log file, e-mail and Mattermost notification with just one method.

For example, method "info()" print info in log files, e-mail and notification.

This object is created at the beginning of ScriptingSystem class. This way, the same object is passed to SFTPServer object, and so for the all duration of the program, logs do not change.

When it is instantiated, it automatically creates a logging object (from logging library already installed with python3), and Email object and a Mattermost object with configuration passed in parameters.

Method "info()" add info type in log file, call "info()" method of Mattermost class and add "info" content to e-mail content by calling "add_content()" of Email class.

Method "warning()" add warning type in log file and add content to e-mail content "warning" by calling "add_content()" of Email class.

Method "error()" add error type in log file, call "error()" method of Mattermost class and add "error" content to e-mail content by calling "add_content()" of Email class.

Method "critical()" only add critical type in log file, o the ground e-mails and notification are not send. This method is only called when there is a reading configuration error such as JSONDecodeError or KeyError.

"get_logfile()" method return the log file name which is "log.log". The method is used by Email class to attach log file to e-mails. This method can be used as a function but it is a problem in files inclusion. In fact, LogEmailMattermost class is include in Email class, and python does not accept recursive inclusions, so I let this method as a method.

The last method "send_all()" is called at the end of the whole script to report the number of error and warnings in log file, and to send e-mails and notification following user configuration. This method call "send()" methods of EMail and Mattermost classes.

## E-mail: Email class

This class is instantiated in LogEmailMattermost class and permits to send e-mails via internal server or smtp server configured by user in configuration file. This class uses smtplib and ssl library to send e-mails. I use theses library on the ground e-mail sender need a smtp connection which contains SSL certificate authentications.

In the constructor of Email, pass e-mail configuration and current LogEmailMattermost object to write information in all logs. So it is such as a loop, LogEmailMattermost writes to Email and Email writes to logs.

From configuration, e-mail is built. If smtp server is not like this "192.168.1.2" or "smtp.mail.yahoo.com", then program uses internal mail server to send e-mails, else it uses smtp connection.

When LogEmailMattermost object calls EMail "send()" method, this class choose internal server or smtp server. If smtp server is configured then it calls "login_and_send()", which calls "login()" method to authenticate on server (with email and password) and to send e-mails to all destination e-mail addresses. If user choose to attach log file, then a MIME item is created. To attach log file, name of this one is required, this is why there is a "get_logfile()" method in LogEmailMattermost. Log file is encoded to good format and attached to e-mail. Before to b sent , data goes in "format_data()" method that modify data to be sent.

## Mattermost notification: Mattermost class

Mattermost object is instantiate in the LogEmailMattermost constructor. The same way Email class is instantiated, LogEmailMattermost passes itself as a Mattermost parameter. This way Mattermost can interact with logs.

In this class, we use a small library "request" to make a simple request on TSE Mattermost server.

When object is instantiated, all components are initialized.

"info()" method add the current task to dictionary of tasks, and add an info pictogram to state dictionary.

"error()" method add the current task to dictionary of tasks, and add an error pictogram to state dictionary.

Method "format_data()" prepare data to be send on server.

The last method “send_mattermost_notification()” use “requests” library to post notification with hook URL. Every error is returned the more precisely in logs. This method is called when LogEmailMattermost call “send_all()”.

## Bash files

### gen_config.sh

This file in root folder call “modules/generate_config_file.py” and execute it. It permits to create a shortcut for user to regenerate configuration file template.

### install_dependencies.sh

Program to set up all.

```
python3 --version

# Upgrade python3 with the last version.
sudo apt-get install --upgrade python3

# Install Pip if it is not installed yet.
# Pip is a dependencies manager. It simplifies libraries imports.
sudo apt-get install python3-pip

# Check pip is correctly installed.
python3 -m pip --version
```

Upgrade python3 to be sure to have the last version enabled. Install pip on computer. Pip is a packages manager. It simplifies dependencies inclusions. To add a dependency with pip, download it with appropriate command and you can include it in all scripts.

So, now install dependencies:

```
python3 -m pip install email_validator
python3 -m pip install pysftp
python3 -m pip install urllib3
```

Modify rights for standard user after downloading project.

```
sudo find . -type f -iname "*.sh" -exec chmod 511 {} \;
sudo find . -type f -iname "*.py" -exec chmod 511 {} \;
sudo find . -type d ! -name "." -exec chmod 511 {} \;
# Config file can be change by anyone.
sudo find . -type f -iname "config.json" -exec chmod 777 {} \;
# Log file can be read only.
sudo find . -type f -iname "log.log" -exec chmod 744 {} \;
```

Now standard user can just write in config.json, read log file and just execute all.

Create cron file and fill it with correct scheduling.

```
{ crontab -l -u $USER; echo "0 23 * * * cd $PWD; python3 main.py"; } | crontab -u $USER -
```

Cron file is created and the program wrote "0 23 * * * cd $PWD; python3 main.py" in it. This line says to execute the main file of the project each day at 23h. The 0 mins all 0 minutes (between 0 and 59), the 23 means at the 23rd hour of the day and all * are day of the month between 1 and 31 (here * means all days), the month between 1 and 12 (* means all months) and last * for the day of the week between 0 and 6 (here * means all). So, every day at 23h, the program runs the following command "cd $PWD; python3 main.py". It means to come to Scripting-System folder directory ($PWD return the current folder of Scripting-System when operation occurred. It means folder cannot change location or re-execute install_dependenecies.sh). and to execute main.py.

Check cron is created:

```
if sudo test -f "/var/spool/cron/crontabs/$USER"
then
        echo "Successfully crontab created in /var/spool/cron/crontabs."
        echo "Logs for this crontab are in /var/log/syslog."
        echo "To restart service, type sudo service cron restart/start."

        # Launch cron service
        sudo service cron start
else
        echo "Error: Cron file not created. Please create it manually as described in users' doc annexes."
fi
```

So this files install all dependencies automatically and create cron.

### server.sh

Launch web server on port 8000. The mounted point is "Web_Server". This is a shortcut to simplify use for user.