SERVER PROJECT MAIL

NIVESSE Charles – LEDOUX Alexis – MARCHAL Baptiste – MEROLLA Mathis

Table des matières

[SQLite DATABASE 1](#_Toc59035845)

[Creation 1](#_Toc59035846)

[Connection and methods 2](#_Toc59035847)

[Mail Server 3](#_Toc59035848)

[Connection 3](#_Toc59035849)

[Recuperation 3](#_Toc59035850)

[Sending 3](#_Toc59035851)

[Interface 3](#_Toc59035852)

# SQLite DATABASE

## Creation

First, we create the SQLite database. Here, we need 3 tables :

* email: a table with all the information about each account.

Columns:

* + Address: the email address of the account.
  + Password: the password of the account.
  + Service: the name of the mail service (gmail, outlook..)
* mail: a table with all the mails from each account.

Columns:

* + Sender: the email address of the sender.
  + Receiver: the email address of the receiver.
  + Body: the message.
  + Date: the date of the mail.
  + Subject: the subject of the mail.
* log: a table with all the log of connection to the mail server.

Columns:

* + Type: the type of connection (login, send, receive..)
  + Ip: the ip address of the machine.
  + Date: the date and time of the connection.
  + Service: the name of the mail service (gmail, outlook..)

## Connection and methods

Now that the database is created, we need a bunch of methods to manipulate it.

First, we need to establish the connection to the database. We create a “create\_connection” method which will connect python with the database file (here names “pythonsqlite.db”).

We use this call to sqlite3 to connect it :



Now that we are connected to the database, let’s create each method.

1. DROP METHODS

We create drop methods to delete data stored in each table. Here we created 3 general drop methods (drop\_email, drop\_all\_mail and drop\_log) and a particular drop method which only drop the mails of a specified account (drop\_mail). This last method will be used when we recover each mails from a server (we’ll see later).

1. INSERT INTO DB METHODS

We create insert methods to insert data in each table of the database. Here we create 3 general methods (insert\_email, insert\_mail and insert\_log).

1. SELECT FROM DB METHODS

We create select methods to select data when needed. Here we create 3 general methods (select\_email, select\_mail and select\_log)

1. SORTING METHODS

We create sorting methods to sort the mails. Here we create 3 sorting methods:

* + select\_mail\_order\_sender: order by sender.
  + select\_mail\_order\_receiver: order by receiver.
  + select\_mail\_order\_subject : order by subject.

1. SAVING MAILS TO A FILE METHOD

We create a method to export every mails to an external file. We call the method mail\_to\_file. If the file does not exist yet, it’ll create it. Otherwise, it’ll overwrite the current one.

1. UPDATE LOG METHOD

Finally, we create a method to update the log table. We call it update\_log and we’ll use it everytime we have a connection to a mail server.

1. COMPLEMENTARY METHODS

We then have some leftover methods that we use in some scenarios:

* + Select\_id\_email: return the id of an account.
  + Email\_exist: used to check if the email is already in the database.

# Mail Server

## Connection

## Recuperation

## Sending

# Interface

For the interface we have the interface.py which contain the main menu of the app.

It works like this, when you launch the app you begin with a series of choice (in the console) for testing our different function. It’s built like this:

* 1. SEND MAIL
     1. Send by outlook
     2. Send by gmail
     3. Return
  2. SYNCHRO MAIL
     1. Synchro outlook
     2. Synchro gmail
     3. Return
  3. LOCAL :
     1. Check your mails
     2. Check logs
     3. Save mails on a file
     4. Return
  4. QUIT

choice ‘a’, ‘b’, ‘c’ and ‘d’ are made with the function choice()

‘a’ use choice\_send\_mail() to give the choice for ‘I’, ‘ii’, ‘iii’ (in the code it’s ‘a’, ‘b’ and ‘c’)

‘b’ use choice\_synchro\_mail() to give the choice for ‘I’, ‘ii’, ‘iii’ (in the code it’s ‘a’, ‘b’ and ‘c’)

‘c’ use choice\_local() to give the choice for ‘I’, ‘ii’, ‘iii’, ‘iv’ (in the code it’s ‘a’, ‘b’, ‘c’ and ‘d’)

By the way every one of these ‘choice’ functions have a validation system so that there won’t be any error coming from the fact that someone put a wrong string.

For choice\_send\_mail():

* If ‘a’ is chosen, we call the function send\_outlook() from ConnectionMailServer.py to send an outlook’s mail.
* If ‘b’ is chosen, we call the function send\_gmail() from ConnectionMailServer.py to send a gmail’s mail.
* If ‘c’ is chosen, we return to choice()

For choice\_synchro\_mail():

* If ‘a’ is chosen, we call the function recup\_dataoutlook() from ConnectionMailServer.py to register the information of a outlook address to the database.
* If ‘b’ is chosen, we call the function recup\_datagmail() from ConnectionMailServer.py to register the information of a gmail address to the database.
* If ‘c’ is chosen, we return to choice()

For choice\_local():

* If ‘a’ is chosen, we call the function read\_mails() from Read\_mails.py to read all the mails from the email address
* If ‘b’ is chosen, we print the result of the function select\_log() from SQL\_retriever.py to check all the logs
* If ‘c’ is chosen, we call the result of the function mail\_to\_file() from SQL\_retriever.py to write all the data from the mails of the address mail to a file name mail.txt
* If ‘d’ is chosen, we return to choice()