1. Coding report

*Class Mail:*

* Variable within class object:
  + **sender**: a string to store the sender's email address.
  + **cmd\_list**: a list storing the command data extracted from a received email.
  + **attachment**: a string to store the path of the return file (in case there is one)
  + **body**: a string to prepare the answer for the email sent back to the requester.
  + **subject**: a fixed string of the email title.
  + **email\_message**: a MIMEMultipart object to compress the email content that will be sent.
* Member functions of the mail class:
  + The `**fetch\_mail**` function is designed to retrieve unread emails from a Gmail inbox with the subject "Mail Control". It logs into the mail server, selects the inbox, searches for unread emails with the specified subject, and processes each email to extract the sender and a command list using the `**decode\_mail**` function.
  + The `**send\_mail**` function is designed to send an email using the SMTP protocol via the Gmail SMTP server (*smtp.gmail.com*). It establishes a connection, logs in using predefined global constants (*USERNAME* and *APP\_PASS*), sends an email to a specified recipient (sender), and prints a message indicating the successful mailing operation.
  + The `**screenshot**` function captures a screenshot of the user's PC, attaches it to an email, and populates the email message with relevant information. The function utilizes an external function, `**capture\_SS**`, to obtain the screenshot file path. It also incorporates auxiliary functions, **current\_time**, and **set\_info**, to include a timestamp and set additional information in the email.
  + The **webcam** function captures an image using the computer's webcam, attaches it to an email, and sets up the email message with relevant information. The function utilizes an external function, **capture\_webcam**, to obtain the webcam capture details. It also includes auxiliary functions, such as **set\_info**, to configure the email subject and additional information.
  + The **keylogger** function initiates a keylogging operation, configures email information, and prepares an email with details about the keylogging duration. Leveraging the external function **key\_logger**, it captures the keylogger log file path and the specified duration for keystroke recording. Additionally, the function incorporates **set\_info** to establish the email subject and initialize the email message object. The resulting email body contains information about the program recording keystrokes for a specific duration. The keylogger file is attached to the email for the sender's retrieval.
  + The **logout** function composes an email body indicating that the user has been logged out of their PC and attaches this message to the email. The function utilizes the **send\_mail** method to send the email, notifying the sender of the logout forward. Subsequently, the function employs the *os.system("shutdown -l")* command to initiate a system logout.
  + Similarly, the shutdown function composes an email body indicating the imminent shutdown of the user's PC and then notifying the sender of the impending shutdown via email. The function proceeds to execute a countdown, displaying the remaining seconds until shutdown in the console. Finally, it initiates a system shutdown using the *os.system("shutdown /s /t now")* command.
  + The **list\_app** function retrieves the list of currently running applications on the user's PC using the **list\_running\_application** function. Then the **set\_info** support function is called to set additional information. The running applications list is attached as a text file named *"RunningApplications.txt"* to the email for the sender's reference.
  + The **terminate\_process** function attempts to terminate a specified process using the **kill\_process** function. It configures email information with the subject "Terminate Process" and attaches a message to the email. If an IndexError occurs, indicating a missing argument, it sets the email subject as "Terminate Process Failed" and appends an error message to the email body.
  + The **send\_log** function facilitates the sending of the program's log file via email. It configures the email subject as "Log File," with an attachment named "*mail.log*". The primary purpose is to provide the sender access to the program's log information for review.
  + The **process\_command** method interprets and executes commands received from email messages. It sets up email information and performs corresponding actions based on a predefined dictionary of commands. Error handling is implemented to capture exceptions during command execution, and an error message is appended to the email body in case of failure. If no valid commands are detected, an error message is set, and the **help** method is invoked to provide a list of available commands.
  + The **run** method serves as the main operational loop of the Mail class. It periodically fetches new emails, processes commands if any are detected, and displays relevant information.

Utility functions:

* + The **decode\_mail** function processes an email message, extracting information such as the sender's email address and command list from the email body.
  + The **check\_ext** function verifies the extension of a given file name in picture format. If not satisfied, the PNG extension is assigned to it automatically.
  + The **capture\_SS** function utilizes the *ImageGrab.grab()* from the Python Imaging Library (PIL) to capture the current screen.
  + Similarly, the **capture\_webcam** function, with the support method from the CV2 library, allows the program to grab a webcam image.
  + The **key\_logger** function sets up a logger using the Python logging module, specifying the log format and file handler. Utilizes the pynput library's Listener to start capturing keystrokes. The logger captures each keystroke and logs it to the file.
  + The **note2log** function records any instance of commands received from email and stores it inside a file named “*mail.log”*
  + The **list\_running\_application** function generates a list of currently running applications on the system using PowerShell. It executes a PowerShell script through the subprocess module, captures the output, and writes the list to a file named *“Applications.txt”* in the "Files" directory. The function then returns the file path.
  + The **list\_running\_process** function generates a list of currently running processes on the system using the "tasklist" command. It executes the command through the *os.popen*, processes the output to remove unnecessary information, and writes the list to a file named “*Processes.txt*” in the "Files" directory. The function then returns the file path.
  + The **kill\_process** function terminates a running process on the system based on the provided data. It accepts either a process ID (PID) or a process name, constructs the appropriate termination command using the "*taskkill*" utility, executes the command in cmd using the *os.open* command, and then returns the state whether successful or not

// Huy viet

GUI

*Class App*

**APP\_WIDTH and APP\_HEIGHT**: int to store the width and height of the application window

**Self\_email**: a string to store our server email

**Sender\_email**: a string to store senders’ emails

**Widget\_list**: a list to store all “Widget”

**Widget\_label**: a “Mail\_page” object that stores and represent all "Widget" on the App

The **add\_new\_mail** function will take the sender's email, list of commands, current mail, our server mail, list of paths to attachments, and reply subject, then send them to the **add\_widget** function of **widget\_label**.

*Class Mail*\_page

**\_\_info\_lb**: a zone that helps us know some details about mail or **Widget**, The details are

"**Email**": senders’ email

"**Content**": a list of all commands separated by a comma

"**Date**": the send date of the mail

The **add\_widget** function will take agruments from the app and create a **Widget** with those agruments. then add that widget to the widget\_list of the app and place that widget below **\_\_info\_lb**. The latest widget is always at the top.

*Class* *Widget*

* A widget is a button representing a message. The widget contains senders’ email, server email, list of all commands, send date of the mail, path of all attachments,

**\_\_m\_master**: the App

The **attachment\_event** function is the event trigger when clicking on the attachment. It will create a pop-up window representing the file based on its extension.

The **home\_button\_event** function will take you back to the mail page.

The **add\_widget** creates content for “Email”, “Content" and “Date” and bindings them with **button\_event**.

The **button\_event** function creates a new frame we call the **widget page**.

In the **widget page**, it contains:

* Space for the home button
* **Sender zone**: contain senders’ email, date of the mail, commands zone,
* **Commands zone**: list of all senders’ commands, each command per line
* **Break line**: separate sender zone from reply zone
* **Reply zone**: content server email, reply content, attachments zone
* **Attachment zone**: contain all attachments. Each line contains a maximum of 3 attachments, with each attachment corresponding to "attachment\_event".

*Class* *Popup*

Create a small window representing content depending on the attachment extension.

If the extension type is

* **Picture file** (“png”, “jpg, “tiff”, "bmp"): it will paste that picture into the pop-up
* **Text file** (“txt” or "log,”) it will read the file and paste all the content into the pop-up.