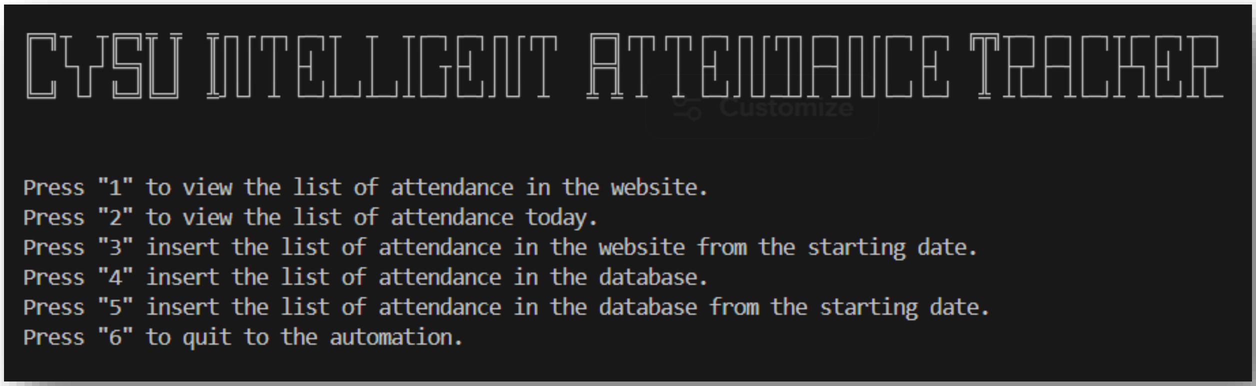
ISAPI CVSU



```
CVSU INTELLIGENT ATTENDANCE TRACKER

Press "1" to view the list of attendance in the website.
Press "2" to view the list of attendance today.
Press "3" insert the list of attendance in the website from the starting date.
Press "4" insert the list of attendance in the database.
Press "5" insert the list of attendance in the database from the starting date.
Press "6" to quit to the automation.
```

This ISAPI CvSU Attendance Automation System is designed to streamline and simplify the process of tracking attendance for various buildings in the Cavite State University. By leveraging the power of Python programming, this automation tool offers efficiency, accuracy, and convenience in storing, recording, and managing attendance records.

Purpose

The primary goal of the ISAPI CvSU Attendance Automation System is to eliminate manual attendance tracking methods, which are often time-consuming and error-prone. This system automates the attendance process, allowing users to efficiently record, monitor, and analyze attendance data with ease.

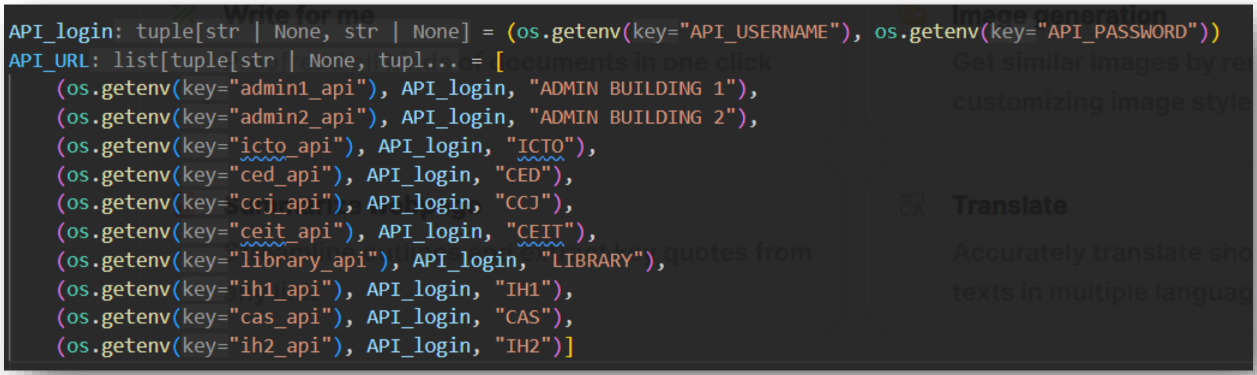
Features

- **Automated Attendance Recording**: The system automates the process of recording attendance, reducing the need for manual data entry.
- **Real-time Updates**: Attendance data is updated in real-time, providing instant access to attendance records.
- **Integration Capabilities**: Seamlessly integrate with other systems or in specific databases such as SQLite for enhanced functionality.

Benefits

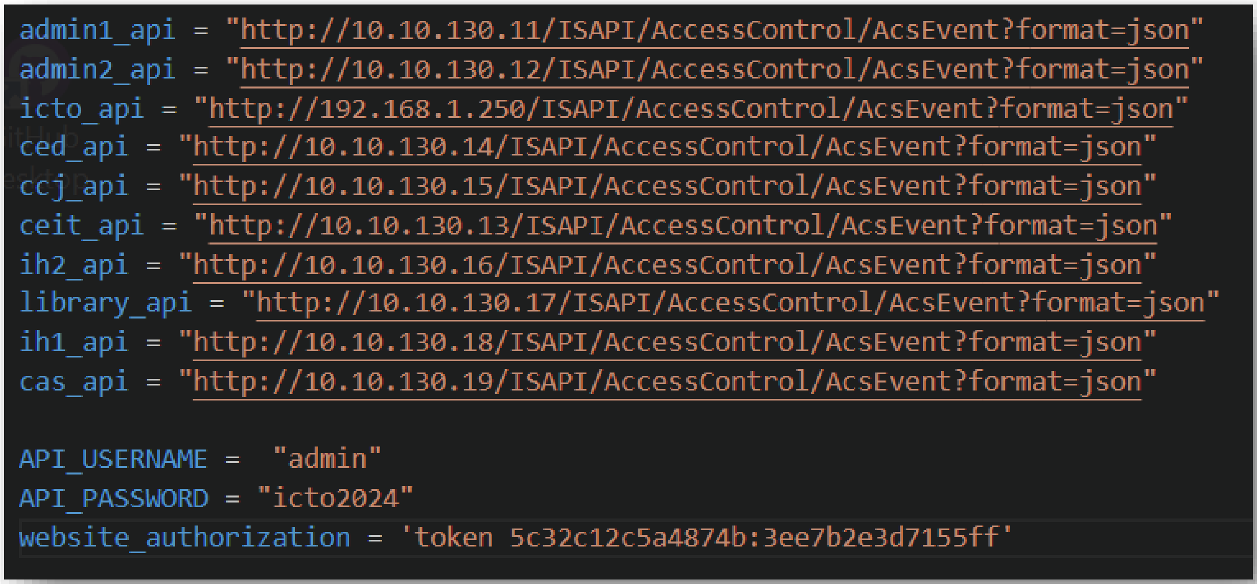
- **Time Savings**: Eliminate manual attendance tracking tasks, saving time and resources.
- **Accuracy**: Reduce the risk of human error in attendance recording and reporting.
- **Efficiency**: Streamline the attendance process for employees in the Cavite State University.

This system is only accessible under the monitoring management of the Information System Manager or Network Administrator. Therefore, the security of the data is limited to the permitted personnel.



```
API_login: tuple[str | None, str | None] = (os.getenv(key="API_USERNAME"), os.getenv(key="API_PASSWORD"))
API_URL: list[tuple[str | None, tupl... = [
    (os.getenv(key="admin1_api"), API_login, "ADMIN BUILDING 1"),
    (os.getenv(key="admin2_api"), API_login, "ADMIN BUILDING 2"),
    (os.getenv(key="icto_api"), API_login, "ICTO"),
    (os.getenv(key="ced_api"), API_login, "CED"),
    (os.getenv(key="ccj_api"), API_login, "CCJ"),
    (os.getenv(key="ceit_api"), API_login, "CEIT"),
    (os.getenv(key="library_api"), API_login, "LIBRARY"),
    (os.getenv(key="ih1_api"), API_login, "IH1"),
    (os.getenv(key="cas_api"), API_login, "CAS"),
    (os.getenv(key="ih2_api"), API_login, "IH2")]
```

The system consists of ISAPI base URL for the POST-retrieval method of the data from the biometric devices. Each URL requires a body, authorization method, and header.



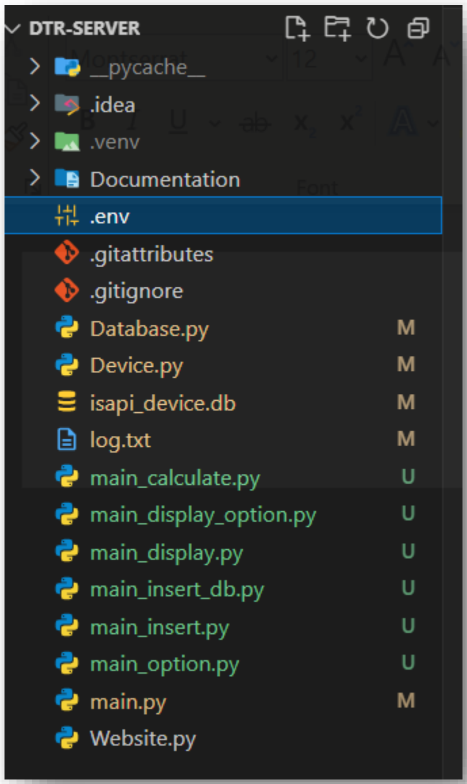
```
admin1_api = "http://10.10.130.11/ISAPI/AccessControl/AcsEvent?format=json"
admin2_api = "http://10.10.130.12/ISAPI/AccessControl/AcsEvent?format=json"
icto_api = "http://192.168.1.250/ISAPI/AccessControl/AcsEvent?format=json"
ced_api = "http://10.10.130.14/ISAPI/AccessControl/AcsEvent?format=json"
ccj_api = "http://10.10.130.15/ISAPI/AccessControl/AcsEvent?format=json"
ceit_api = "http://10.10.130.13/ISAPI/AccessControl/AcsEvent?format=json"
ih2_api = "http://10.10.130.16/ISAPI/AccessControl/AcsEvent?format=json"
library_api = "http://10.10.130.17/ISAPI/AccessControl/AcsEvent?format=json"
ih1_api = "http://10.10.130.18/ISAPI/AccessControl/AcsEvent?format=json"
cas_api = "http://10.10.130.19/ISAPI/AccessControl/AcsEvent?format=json"

API_USERNAME =  "admin"
API_PASSWORD = "icto2024"
website_authorization = 'token 5c32c12c5a4874b:3ee7b2e3d7155ff'
```

The image above shows what is the environment variable of each data in the python, this provides additional security of the data inside the program.

Before executing the “main.py” file. Few python package installations are needed:

1. Install the python request.
    a. Steps of installing the python request:
        i. Open the command prompt in your text editor: visual studio code, PyCharm, and others.
        ii. Ensure that the python interpreter is set in the recommended path.
        iii. Type “pip install request” or “pip3 install request” in the command line.
        iv. If error persists, try this instruction: https://pypi.org/project/requests/

2. Install the python dotenv
    a. Steps in installing the python dotenv
        i. Open the command prompt in your text editor: visual studio, PyCharm, and others.
        ii. Ensure that the python interpreter is set in the recommended path
        iii. Type “pip install dotenv” or “pip3 install dotenv” in the command line.
        iv. If error persists, try this instruction: https://pypi.org/project/python-dotenv/

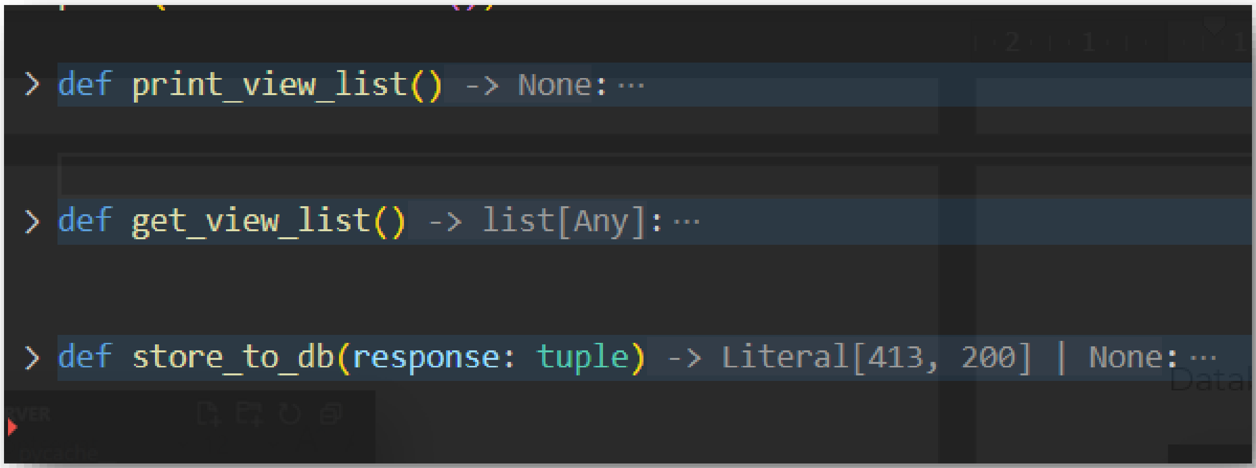


```
DTR-SERVER
> __pycache__
> .idea
> .venv
> Documentation
  .env
  .gitattributes
  .gitignore
  Database.py                M
  Device.py                  M
  isapi_device.db            M
  log.txt                    M
  main_calculate.py          U
  main_display_option.py     U
  main_display.py            U
  main_insert_db.py          U
  main_insert.py             U
  main_option.py             U
  main.py                    M
  Website.py
```

The program is separated into two python files: main.py and main_option.py. The program "main.py" holds the connection of the data automation that holds every iteration; this iteration can be adjusted in the code by modification of the time (seconds). This file holds the process connected to the several files: main_insert.py, main_display.py, and main_calculation.py.

While the "main_option.py" holds the connection of manual trigger data insertion in either local database (SQLite) or the online database (Website). This filed holds the connection from few files: main_insert_db.py, main_display_option.py, main_calculation.py, and main_insert.py.

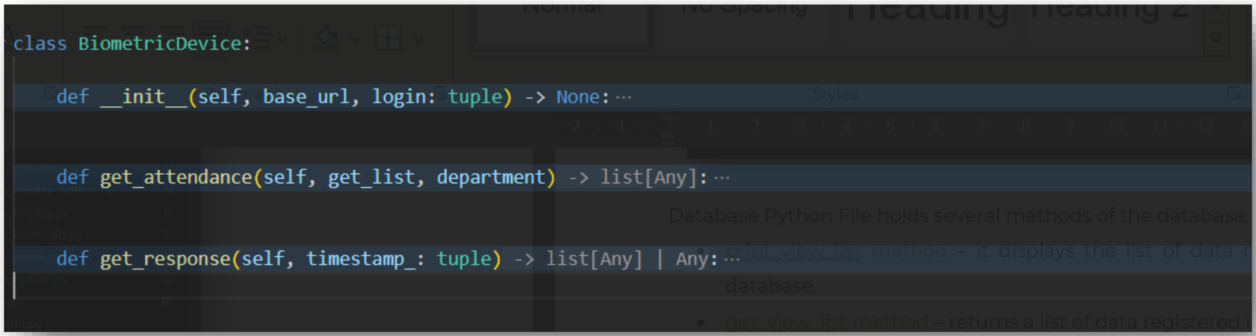
Both of these files are connected to the Database, Device, and Website python files.



```
> def print_view_list() -> None: ···

> def get_view_list() -> list[Any]: ···

> def store_to_db(response: tuple) -> Literal[413, 200] | None: ···
```

Database Python File holds several methods of the database:

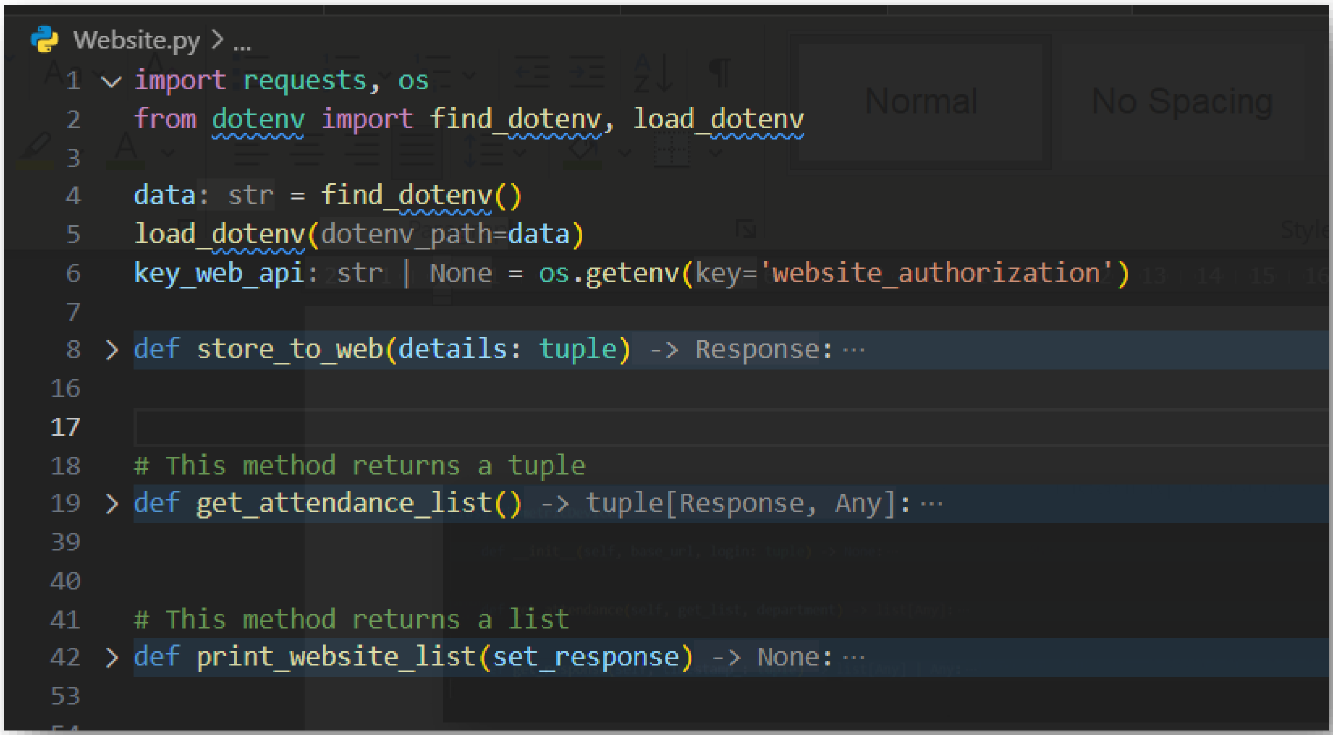
- print_view_list method – it displays the list of data registered in the database.
- get_view_list method – returns a list of data registered in the database.
- store_log_db method – process the insertion of data in the local database.



```
class BiometricDevice:

    def __init__(self, base_url, login: tuple) -> None: ···

    def get_attendance(self, get_list, department) -> list[Any]: ···

    def get_response(self, timestamp_: tuple) -> list[Any] | Any: ···
```

Device Python File is a class object that holds the key information to access the biometric device. In addition, it holds two methods to process data retrieval from the biometric devices.

- Get_attendance method – it returns a list of attendance in the biometric device.
- get_response method – returns a response from the biometric device.



Website.py > ...

```python
import requests, os
from dotenv import find_dotenv, load_dotenv

data: str = find_dotenv()
load_dotenv(dotenv_path=data)
key_web_api: str | None = os.getenv(key='website_authorization')

> def store_to_web(details: tuple) -> Response: ···


# This method returns a tuple
> def get_attendance_list() -> tuple[Response, Any]: ···


# This method returns a list
> def print_website_list(set_response) -> None: ···
```

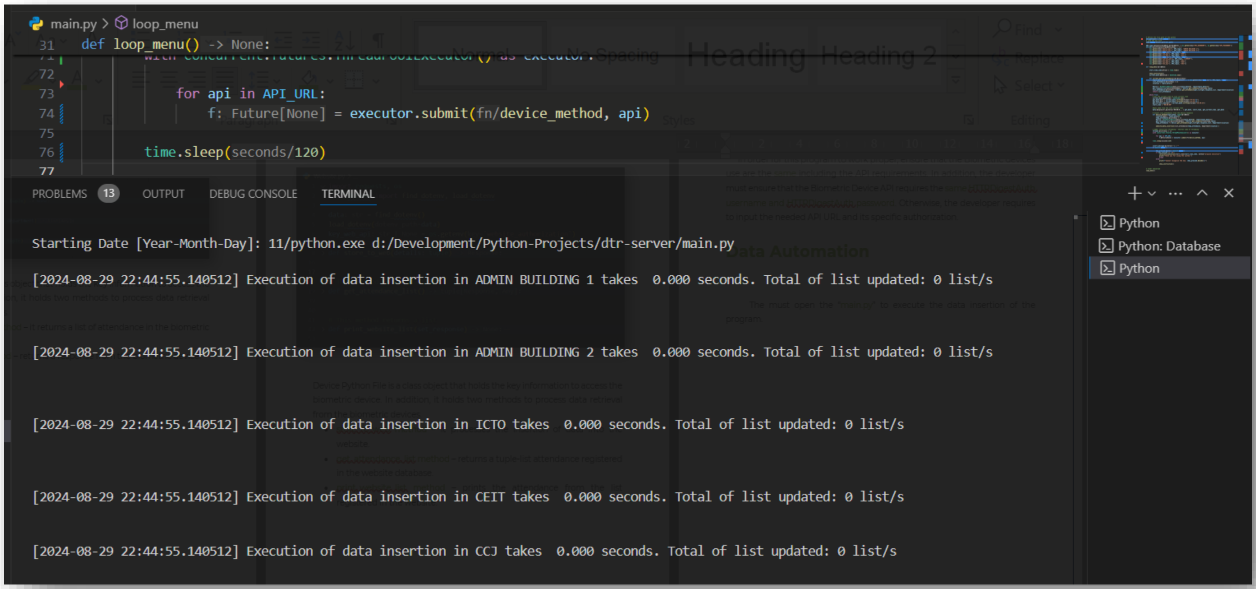
Device Python File is a class object that holds the key information to access the biometric device. In addition, it holds two methods to process data retrieval from the biometric devices.

- store_to_web method – it processes the insertion of data into the website.
- get_attendance_list method – returns a tuple-list attendance registered in the website database.
- print_website_list method – prints the attendance from the list registered in the website.

In order for this program to work proper, ensure that the Biometric devices use are the same including the API requirements. In addition, the developer must ensure that the Biometric Device API requires the same HTTPDigestAuth username and HTTPDigestAuth password. Otherwise, the developer requires to input the needed API URL and its specific authorization.

Data Automation

The must open the “main.py” to execute the data insertion of the program.

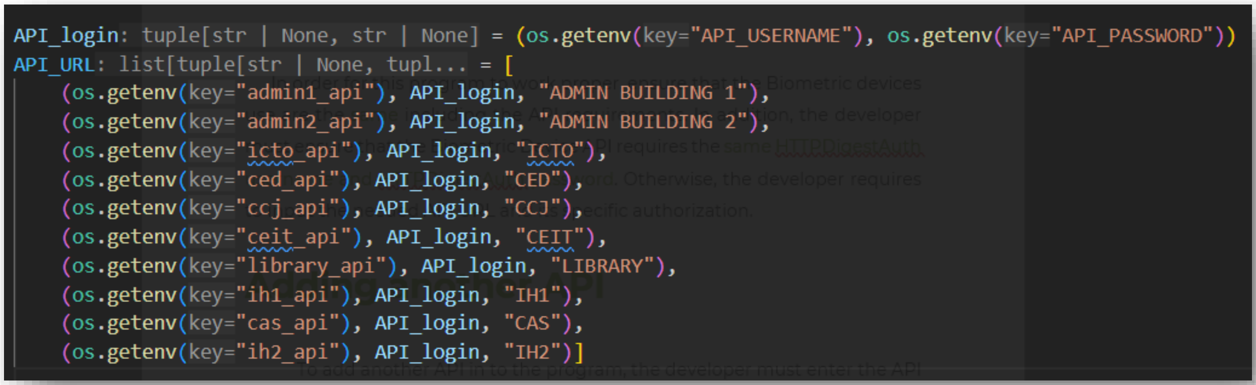


```
def loop_menu() -> None:

        for api in API_URL:
            f: Future[None] = executor.submit(fn/device_method, api)

    time.sleep(seconds/120)
```

```
Starting Date [Year-Month-Day]: 11/python.exe d:/Development/Python-Projects/dtr-server/main.py
[2024-08-29 22:44:55.140512] Execution of data insertion in ADMIN BUILDING 1 takes  0.000 seconds. Total of list updated: 0 list/s

[2024-08-29 22:44:55.140512] Execution of data insertion in ADMIN BUILDING 2 takes  0.000 seconds. Total of list updated: 0 list/s

[2024-08-29 22:44:55.140512] Execution of data insertion in ICTO takes  0.000 seconds. Total of list updated: 0 list/s

[2024-08-29 22:44:55.140512] Execution of data insertion in CEIT takes  0.000 seconds. Total of list updated: 0 list/s

[2024-08-29 22:44:55.140512] Execution of data insertion in CCJ takes  0.000 seconds. Total of list updated: 0 list/s
```

Adding another API

To add another API in to the program, the developer must enter the API URL in the environment variable of python file. Add specific username and password in the ".env" if necessary to avoid conflict during the execution.

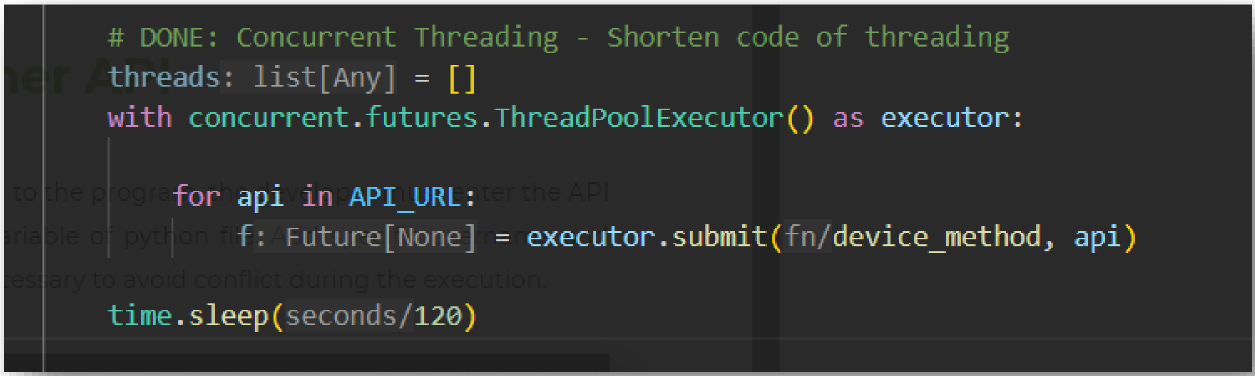


```python
API_login: tuple[str | None, str | None] = (os.getenv(key="API_USERNAME"), os.getenv(key="API_PASSWORD"))
API_URL: list[tuple[str | None, tupl... = [
    (os.getenv(key="admin1_api"), API_login, "ADMIN BUILDING 1"),
    (os.getenv(key="admin2_api"), API_login, "ADMIN BUILDING 2"),
    (os.getenv(key="icto_api"), API_login, "ICTO"),
    (os.getenv(key="ced_api"), API_login, "CED"),
    (os.getenv(key="ccj_api"), API_login, "CCJ"),
    (os.getenv(key="ceit_api"), API_login, "CEIT"),
    (os.getenv(key="library_api"), API_login, "LIBRARY"),
    (os.getenv(key="ih1_api"), API_login, "IH1"),
    (os.getenv(key="cas_api"), API_login, "CAS"),
    (os.getenv(key="ih2_api"), API_login, "IH2")]
```

The developer needs to provide a separated API_login if necessary. As well as an additional API data in the API_URL variable list in a tuple format (os.getenv({api_url})), {API_Login/(or separated API login credentials)}, department/location)

Schedule insertion

To adjust the automated insertion of data in the program. Change the developer must time.sleep seconds desired that can be located right below the future threading in “main.py”.

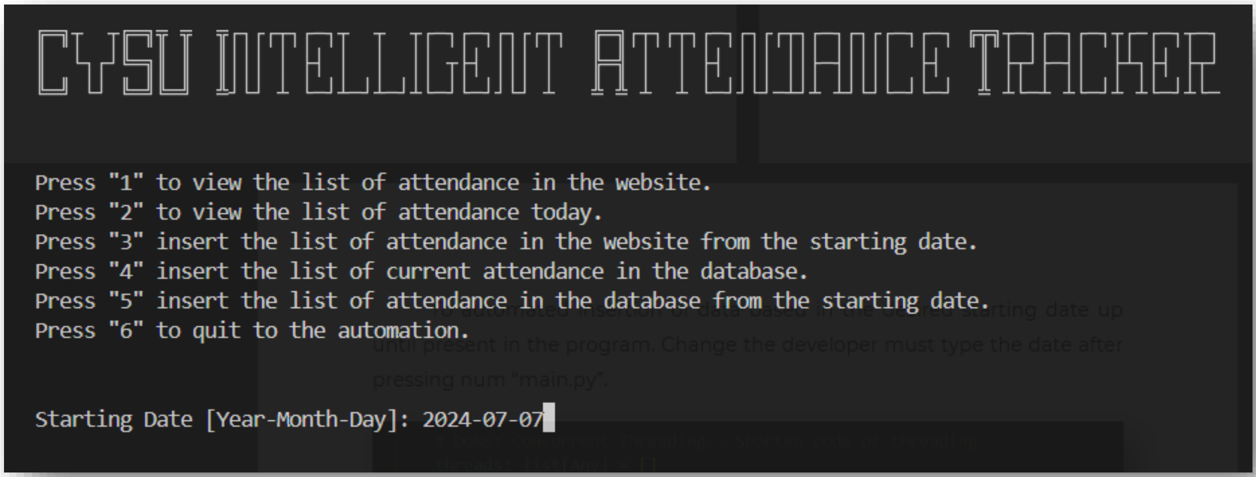


```
# DONE: Concurrent Threading - Shorten code of threading
threads: list[Any] = []
with concurrent.futures.ThreadPoolExecutor() as executor:

    for api in API_URL:
        f: Future[None] = executor.submit(fn/device_method, api)

time.sleep(seconds/120)
```

If the time.sleep() method is in the comment form; uncomment it in the program to apply the effect.

Date-to-Present insertion

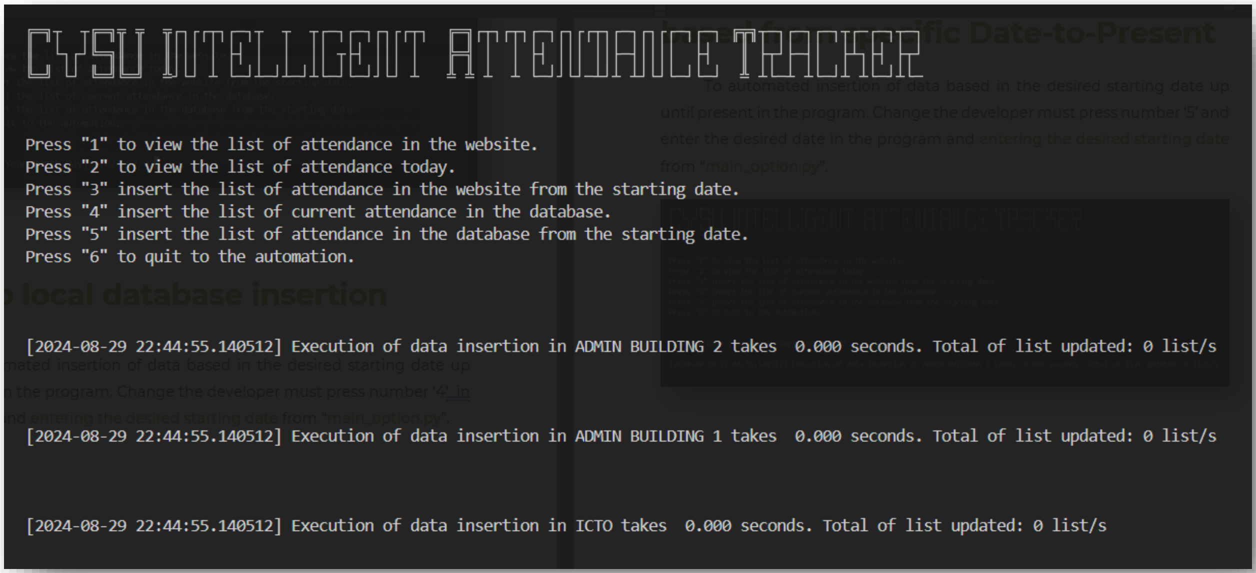
To automated insertion of data based in the desired starting date up until present in the program. Change the developer must type the date after pressing number ‘3’ in the program and entering the desired starting date from “main_option.py”.



```
CVSU INTELLIGENT ATTENDANCE TRACKER

Press "1" to view the list of attendance in the website.
Press "2" to view the list of attendance today.
Press "3" insert the list of attendance in the website from the starting date.
Press "4" insert the list of current attendance in the database.
Press "5" insert the list of attendance in the database from the starting date.
Press "6" to quit to the automation.


Starting Date [Year-Month-Day]: 2024-07-07
```

Save to local database insertion

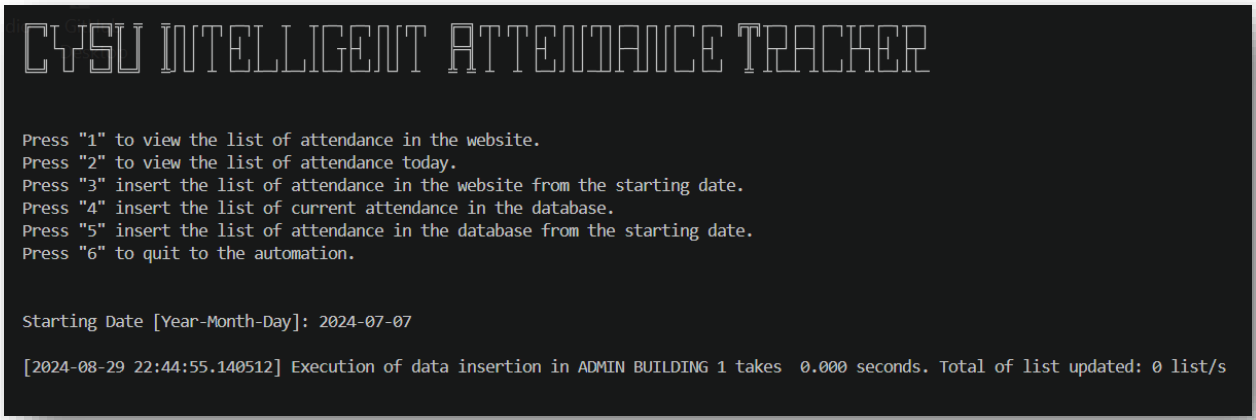
To automated insertion of data based in the desired starting date up until present in the program. Change the developer must press number ‘4’ in the program from “main_option.py”.



```
CVSU INTELLIGENT ATTENDANCE TRACKER

Press "1" to view the list of attendance in the website.
Press "2" to view the list of attendance today.
Press "3" insert the list of attendance in the website from the starting date.
Press "4" insert the list of current attendance in the database.
Press "5" insert the list of attendance in the database from the starting date.
Press "6" to quit to the automation.


[2024-08-29 22:44:55.140512] Execution of data insertion in ADMIN BUILDING 2 takes  0.000 seconds. Total of list updated: 0 list/s


[2024-08-29 22:44:55.140512] Execution of data insertion in ADMIN BUILDING 1 takes  0.000 seconds. Total of list updated: 0 list/s


[2024-08-29 22:44:55.140512] Execution of data insertion in ICTO takes  0.000 seconds. Total of list updated: 0 list/s
```

Save to local database insertion based from specific Date-to-Present

To automated insertion of data based in the desired starting date up until present in the program. Change the developer must press number ‘5’ and enter the desired date in the program and entering the desired starting date from “main_option.py”.



```
CVSU INTELLIGENT ATTENDANCE TRACKER


Press "1" to view the list of attendance in the website.
Press "2" to view the list of attendance today.
Press "3" insert the list of attendance in the website from the starting date.
Press "4" insert the list of current attendance in the database.
Press "5" insert the list of attendance in the database from the starting date.
Press "6" to quit to the automation.


Starting Date [Year-Month-Day]: 2024-07-07

[2024-08-29 22:44:55.140512] Execution of data insertion in ADMIN BUILDING 1 takes  0.000 seconds. Total of list updated: 0 list/s
```