**Figuaroz Handover**

**Installation**

1. Make sure you have python 3.10.X or above installed
2. Create a local environment for python using "python -m venv venv"
3. activate the environment
   1. this would be automatically using IDEs like PyCharm

or manually using

* 1. source venv/bin/activate

1. Install all the needed packages "pip install -r requirements.txt"
2. Run the project from the “main.py” script in the main directory

**Notes**:

1. To activate the env manually , python must be in your system environment variables.
2. On MacOS and Linux, the library “Pillow” must be in its last version. To do so run the command “pip install -U Pillow”
3. There is an os-independent libraries, I recommend you take a look at the **requairments.txt** file

**Structure**

The project subdirectories are:

1. **scripts**: This is the main directory, it contains the views, controllers which is taking over the entire application
   * **controllers.py**: it contains the
     1. **MainController**: it holds sessions, connections, and variables all the way across the application (this is cross threading but not cross process)
     2. **Timer Observers…**: whenever a task/project activated or stopped it will be triggered and log records to the database and monitor it
     3. **InputsObserver**: this keeps an eye on the system inputs (keyboard, mouse clicks an moves)
   * **custom\_views.py**: this contains
     1. **CustomTimer**: this is the observed timer objects and where the trigger rules for timers are implemented… This is the actual thing for timers
     2. **TimeTracker**: this is just tracker for the custom timers, it doesn’t change anything and doesn’t trigger anything… it is only used for main view labels to track changes on the current timer or aggregations in time
   * **trackers.py**: this contains
     1. **Periodic**: this is the most important class; it schedules the tasks and initiate a non-blocking thread every given interval to execute a task once and stop until another interval passed
     2. **ScreenshotsCapture**: this is where the screenshots being taken, saved, and insert to database… it is activated through periodic
     3. **InternetStateTracker**: it tracks the internet speed and connectivity. including geo\_location and ip\_address… it is activated through periodic
   * **test.py**: this is where the stands alone testing goes…
   * **views.py**: this contains
     1. **MainApp**: the holder of the application… just keep an eye on the constructor it explains everything
     2. **LoginForm**: the remember me take place here… **it is worthy to say that the forms move between each other through destroy and create** on each jump
     3. **ForgetPasswordForm**: this is only sent request to the server and doesn’t retrieve anything
     4. **FloatingDialogBar**: this will not destroy anything, and it is basically a frameless window. All what it contains are shadowing items. Hence, I am creating it on minimize and destroying it on maximize
2. **run**: this is the output directory where all screenshots, logs and other are saved
3. **env**: it holds the database, and any config file should be placed here
4. **database**: it is where all our models are taking place and bring called, created, (and mostly inserted)
   * **models.py**: is the main SQLAlchemy db models. It is responsible about creating the main db session and create the database if not exists on new installations… note that each model represents a table in the database.
   * **custom\_models.py**: basically this is just shadow for the main project\_time\_line model. Instead of using the original we are using this to cover for the time trackers the MainApp view.
5. **assets:** it holds all the images and assets used in the project
6. **apis:** it holds all scripts treating the APIs in a moving forward behavior
   * **sync.py**: this contains
     1. **DataWrapper**: this class holds the exception handling process of the loader/requester.
     2. **DataRequester**: This holds the URLs in get and post form. All the get mothods take token and return data. The post methods take data and send it to the URL (Note: All the methods assumes that the data is prepared before received)
     3. **DataLoader**: It loads the unsynced data from the database, format it in the right format and return as list of dicts
     4. **LoadUploadController**: Mapper between the requester and the loader, it calls the loading to prepare the data and send it to the requester to upload it.
7. **active\_window\_watcher**: This is the directory responsible for monitoring window changes and new applications, and URL changes…
   * It must run as a total separate process (This is a pythonic behavior where the thread can see same level and sub level processes. Hence, running this on top level process – on its own – to be able to catch all other processes
   * The main script in this directory is “main\_window\_watcher.py” followed by “lib.py” and then the scripts differ based on the OS used
   * Based on the strategy used in the “config.py” at line 9 -- python might use choices=["jxa", "applescript", "swift"] to monitor the window and application changes in the application
8. **idle\_watcher**: This holds all the scripts related to monitor the idle of the user
   * **afk.py** is the main script here and it is used in a direct behavior within a periodic
9. **common**: it contains the common things between the system scripts.
   * **Utils**.py holds the functions that are supplementary
   * consts.py holds all constants of the system, including paths and css, and config variables
   * logs, dirs: the logging behavior through the system.

**TODO:**

1. Installer: create the setup script, one script for each platform containing the set of libraries to be used in. Also, using the MacOS script to ask for permission.
2. Uploading a screenshot: the behavior of this is mysterious… it is sending a get request to receive a value to do a screenshot upload… and this go for each screenshot. STILL NOT WORKING
3. UI modifications: this is very simple; it is a move forward… look at the consts.py to have more understanding
4. Passing a value of the active projecttimeline id to register it in the UserLog table record at creation of a new window (this works fine with the default). Th issue here is the communication between the two processes. The MainController will not be able to provide its modified values here.
5. The speedtest behavior is not synchronize casing the insert to be done before the results retrieved and casing the entire session to fail unless rolled back (This can be solved through rolling back if the result is not yet retrieved and get it on the next insertion)
6. The path of the current active window on Linux and MacOS:
   1. This is very simple, currently we are retrieving the pid. Using this pid with psutil will get all the information’s needed.
7. The URL active on browser.
   1. Suggested solution is. When the current active application is a browser. Get all information’s of the tabs on that browser, then compare the titles and use this to get the URL and other info about it. (Even if it is duplicate, it will have the same info and hence the same result)