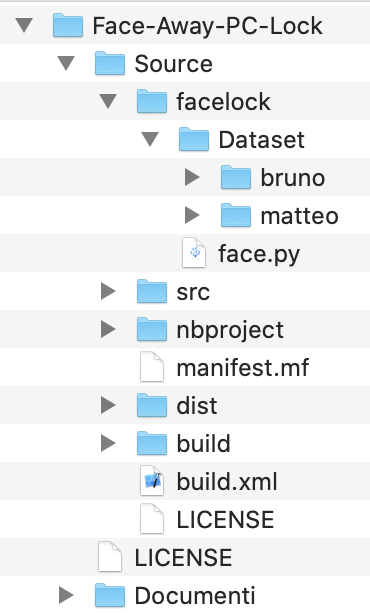
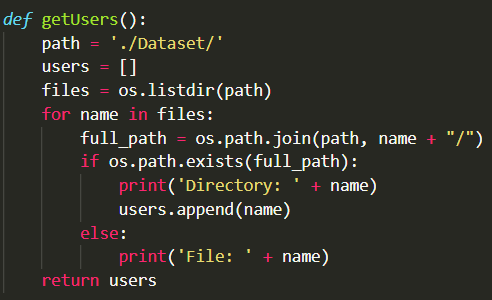
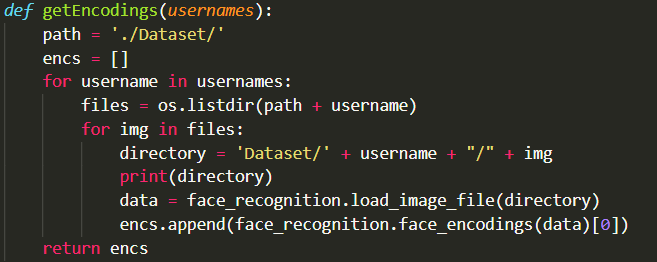
# Lavori svolti

|  |  |  |  |
| --- | --- | --- | --- |
| Orario | lavoro svolto | persone | ID Lavoro |
| 8:20 – 9:00 | Definito standard per lo sviluppo | Matteo, Jonas | 1 |
| 9:00 – 10:30 | Sviluppata funzione per ricevere il numero di utenti | Matteo | 2 |
| 8:20 – 16:30 | Aggiunta di volti | Bruno | 3 |
| 10:30 – 11:35 | Sviluppata funzione per ricevere gli encodings delle immagini | Matteo | 4 |
| 13:15 – 16:30 | Funzione per riconoscere se in un frame ci sono uno o più volti | Jonas | 5 |
| 13:15 – 16:30 | Funzione per ottenere un array di nomi lungo quanto gli encodings | Matteo | 6 |
| 8:20 – 16:30 | Aggiunta utenti tramite interfaccia grafica | Luca | 7 |

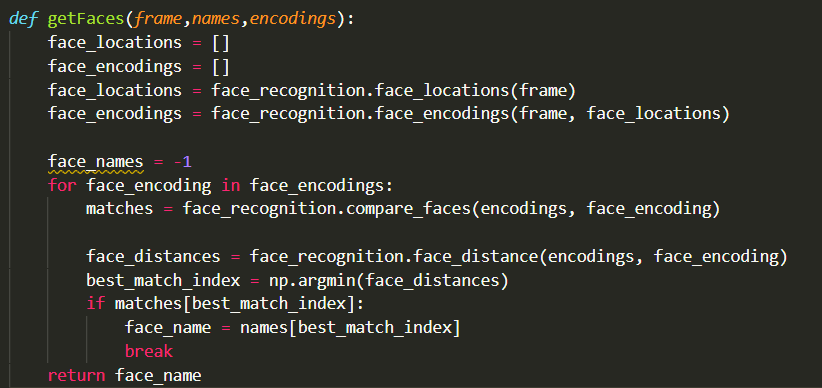
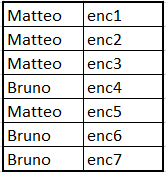
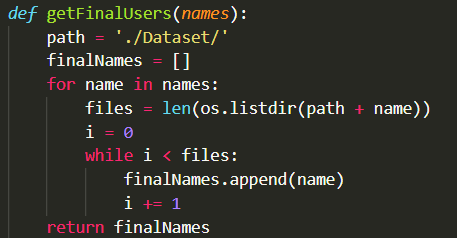
# Lavori

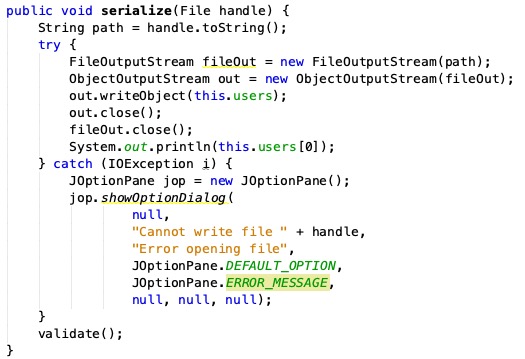
1. Abbiamo deciso di sviluppare le cartelle del programma nella seguente maniera:  
   L’utilizzo delle seguenti directory è:
   * **Face-Away-PC-Lock:** Directory principale del progetto, che contiene tutto.
   * **Documenti:** Contiene i documenti del progetto, tra cui i requisiti, i diari e la documentazione.
   * **Source:** Programma effettivo.
   * **Facelock:** Contiene tutti gli algoritmi in python per riconoscere i volti.
   * **Dataset:** set di immagini da paragonare a quello che vede la webcam, dentro a questa directory ci saranno delle cartelle con il nome del profilo da controllare.
   * **Resto:** Il resto delle cartelle (in Source) non sono altro che cartelle del progetto NetBeans per l’interfaccia grafica.
2. È stata sviluppata la funzione *getUser*che si occupa di ritornare il numero di utenti basandosi sul numero di cartelle all’interno della cartella Dataset. Per essere sicuro che il numero di utenti sia corretto non si basa semplicemente sul numero di file ma sul numero di directory, di cui verranno poi presi i nomi e ritornati sotto forma di lista. Il procedimento esatto che esegue è il medesimo:
   * Prende la lista di tutto quello che è contenuto all’interno della cartella Dataset (che conterrà le immagini degli utenti).
   * Grazie ad un join, metodo che consente di aggiungere elementi ad una path, ottiene la path completa (ovvero quella assoluta) del elemento corrente.
   * Stampa a terminale se si tratta di una directory o di un file (solo debug).
   * Se si tratta di una directory aggiunge il nome della directory, che corrisponde a quello dell’utente, alla lista di utenti che l’algoritmo effettivo dovrà poi andare a controllare.

Gestire le path in python ci ha richiesto di importare *os*, un modulo che si occupa di gestire in maniera molto comoda i file con le relative path, dando a disposizione metodi molto utili.

1. Sono stati creati 2 file diversi (1 Java e 1 Python) che sono stati messi in relazione. Prima è stato creato un file Python che si occupa di accendere la camera e, secondo l’input dell’utente premendo il tasto “S”, scatta una foto che in seguito guarda in una cartella creata apposta per lui. Se la cartella non esiste, viene creata. Il file può ricevere come parametro il nome dell’utente attuale (proveniente dal codice di java). Il file Java invece apre un file in una path scelta (nel nostro caso viene adattato a Python) tramite cmd.
2. **Sfruttando all’incirca il metodo utilizzato per *getUsers()* è stata sviluppata un altro metodo che permette di ricevere gli encodings di tutte le immagini all’interno delle directory in *Dataset.*

La funzione si occupa di entrare nelle cartelle chiamate come i nomi trovati in precedenza (da *getUser*) e, per ogni immagine trovata al suo interno, fa l’encoding e la aggiunge all’array con tutte esse.

1. La funzione prende un frame catturato dalla webcam come parametro, un array di encodings e un array di nomi. Il funzionamento è il seguente:
   1. Prende l’immagine passata come argomento (frame) e cerca grazie al metodo *face\_locations* di *face\_recognition* le facce che sono contenute.
   2. Dall’immagine precedente prende i volti e ne fa l’encoding.
   3. Compara le immagini prese dal frame con gli encoding passati come argomento (che non sono altro che quelli generati dal metodo *getEncodings* in base alle immagini delle persone)
   4. Se trova un volto scrive il nome del relativo utente ed esce dal for (ritornando il valore) altrimenti se non lo trova ripete il ciclo finché non ne trova uno o non finiscono gli encodings
2. La funzione seguente serve per avere un array di lunghezza uguale a quella degli encodings, in questa maniera si potrà avere un caso simile a quello riportato qui a sinistra, dove una persona può avere più encodings e quindi più immagine che lo identificano. La procedura utilizzata è molto simile a quella della funzione *getEncodings.*
3. È stata aggiunta l’opzione di aggiunta, tramite l’interfaccia grafica. Questa funzione non provvede ancora alla comunicazione tra il software Python e quello dell’interfaccia grafica (Java), ma crea la base per tale collegamento. In primis, al click su un’area utente viene creato un nuovo utente, provvisto solo di nome al momento. Questo utente viene aggiunto ad un array di stringhe.

Per memorizzare gli utenti, con il solo scopo dell’output da GUI, infatti il software di riconoscimento in Python si basa sulle cartelle, viene serializzata l’array di stringhe. La serializzazione viene effettuata tramite il metodo serialize(), il quale si occupa della serializzazione dell’array di stringhe, in modo da renderlo accessibile in modo globale all’esecuzione del programma. Per leggere i dati in modo globale viene utilizzato il metodo deSerialize(), che si occupa di leggere il contenuto del file generato dalla serializzazione. La prima funzione viene richiamata ogni qual volta viene eseguita una modifica all’array di utenti, quindi quando un utente viene aggiunto o eliminato. La seconda viene richiamata all’avvio dell’applicativo e ogni volta che viene richiamata la funzione di serializzazione. Nei due metodi viene utilizzata la funzione validate() per aggiornare i valori dei label.

# Problemi riscontrati

2.   
  
  
  
  
Abbiamo riscontrato un problema con il modulo os, più specificatamente all’inizio utilizzavamo il metodo *isdir().* Quando provavamo ad utilizzarlo funzionava unicamente se il file contenente l’algoritmo era situato all’interno della directory da controllare, per ovviare ho cercato una soluzione alternativa e sono riuscito a trovare *exists()*. Lo scopo di questo metodo è dire se una directory (che gli passiamo tramite parametro) esiste o meno, quindi prendendo il percorso assoluto dell’elemento da verificare e passandoglielo possiamo sapere se si tratta di una directory o di un file.

5. Inizialmente volevamo fare in modo che un utente aveva più encodings (in questa maniera il programma risultava più pulito e leggero) ma purtroppo farlo era troppo complicato e abbiamo preferito avere dei dati ridondanti. Per vedere esattamente come abbiamo fatto guarda lavoro numero 6.

7. L’aggiornamento degli spazi utente apparentemente non era funzionante. Questo era causato dalla dimenticanza della lettura del file ad ogni scrittura. All’inizio era previsto il salvataggio del file in JSON. Per renderci la vita più semplice è stato poi modificato, usando la serializzazione, della quale si era già a conoscenza dei metodi.

# Programma di massima per la prossima giornata di lavoro

Riuscire ad aggiungere utenti dall’interfaccia grafica con le foto e i percorsi giusti, riconoscere utente con più immagini

# Punto rispetto alla pianifica

Come settimana scorsa ci troviamo molto in avanti rispetto al piano di lavoro.