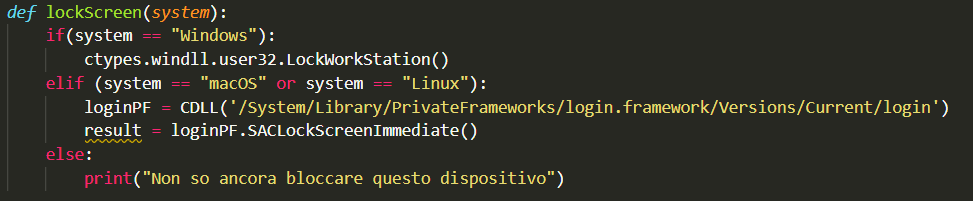
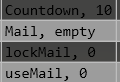
# Lavori svolti

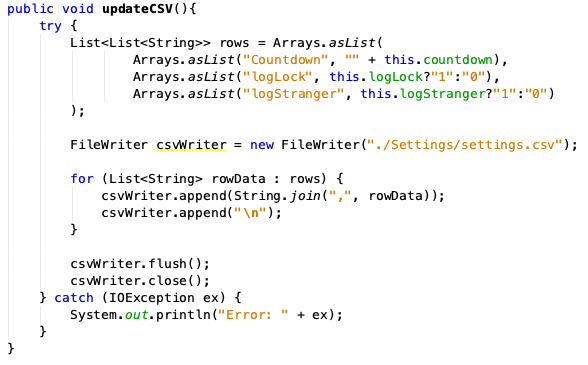
|  |  |  |  |
| --- | --- | --- | --- |
| Orario | lavoro svolto | persone | ID Lavoro |
| 08:20 – 11:35 | Blocco del computer | Matteo | 1 |
| 08:20 – 11:35 | Multipiattaformità e comunicazione Java-Python | Luca, Bruno | 2 |
| 13:15 – 14:30 | Trasportazione delle impostazioni tramite un csv | Matteo | 3 |
| 13:15 – 14:45 | CSV | Luca, Bruno | 4 |
| 14:30 – 15:00 | Funzione per logging degli eventi | Matteo | 5 |
| 15:00 – 16:30 | Diari | Matteo, Luca, Bruno | 6 |

# Lavori

1. In questo lasso di tempo mi sono occupato di trovare un comando che potesse bloccare lo schermo sia di windows, di macOS e anche di linux.  
   Grazie alla funzione che possiamo vedere qui sopra ho gestito il blocco del computer in base al sistema che esegue il programma. La variabile system viene istanziata all’inizio del programma e rappresenta una stringa che può prendere 4 valori, ovvero:
   * **Windows**: rappresenta il sistema operativo di Microsoft.
   * **macOS**: sistema operativo di Apple (eseguito sui mac).
   * **Linux**: sistemi che eseguono Linux.
   * **“”**: la stringa vuota rappresenta un sistema operativo sconosciuto, in questo caso nella funzione *lockScreen()* stampa “non so ancora bloccare questo dispositivo”.
2. Abbiamo continuato e finito i pezzi di codice che ci hanno permesso di eseguire il file Python tramite la GUI di Java, su qualsiasi sistema operativo.

Questo è stato fatto tramite la classe ProcessBuilder, la quale permette di eseguire un qualsiasi comando da linea di comando.

1. Alla fine abbiamo optato per scambiare le informazioni tra il file Java e quello python mediante un semplicissimo file csv, abbiamo preso questa scelta perché in entrambi i linguaggi (al contrario di json) l’implementazione è estremamente facile. La struttura del file csv è quella che possiamo vedere nell’immagine qua vicina. Le informazioni che per ora ci interessano sono il countdown totale che il programma deve aspettare prima di bloccare il computer, la mail alla quale notificare l’utente, il flag se l’utente vuole ricevere la mail quando il computer viene bloccato e il flag se l’utente vuole essere avvisato quando un altro utente sta usando il computer.
2. È stato creato il metodo di scrittura delle impostazioni, in formato CSV.

Questo metodo non si occupa di scrivere un CSV generico, bensì uno dedicato alla struttura.

1. Il file csv è stato modificato nella struttura, più specificatamente la mail è scomparsa, il lockMail è diventato logLock e useMail è diventato logStranger. Questi flag adesso rappresentano quello che verrà scritto nel file di log, che è giornaliero. L’idea è quella di visualizzare poi il file di log (sotto forma tabellare) all’interno dell’interfaccia grafica. I log sono stati strutturati come dei file csv e al loro interno troviamo i campi:
   * **Time**: l’ora completa (ora minuti e secondi) in cui l’azione è stata eseguita
   * **Action**: l’azione eseguita dal programma, per ora solo se ha riconosciuto uno sconosciuto o se ha dovuto bloccare il computer
   * **LastUser**. L’ultimo utente che il programma ha riconosciuto prima di aver eseguito l’azione.



# Problemi riscontrati

2) Non si riusciva ad eseguire lo script python, né con la classe ProcessBuilder, né con la classe Runtime, usata come primo approccio. Tale problema risultava in una fallita esecuzione. Inizialmente si cercava di eseguire un comando cd per entrare nella cartella di esecuzione, ma questo approccio risultava nella mancata accessibilità alla cartella. In seguito ho scoperto la presenza del metodo directory(), il quale sposta l’esecuzione nella directory selezionata.

# Programma di massima per la prossima giornata di lavoro

# Punto rispetto alla pianifica

Estremamente in avanti