Documentazione

**GuestDataLogger v2**

[1 Introduzione 3](#_Toc72399211)

[1.1 Informazioni sul progetto 3](#_Toc72399212)

[1.2 Abstract 3](#_Toc72399213)

[1.3 Scopo 3](#_Toc72399214)

[2 Analisi 4](#_Toc72399215)

[2.1 Analisi del dominio 4](#_Toc72399216)

[2.2 Analisi e specifica dei requisiti 4](#_Toc72399217)

[2.3 Use case 7](#_Toc72399218)

[2.4 Pianificazione 1](#_Toc72399219)

[2.5 Analisi dei mezzi 1](#_Toc72399220)

[2.5.1 Software 1](#_Toc72399221)

[2.5.2 Hardware 1](#_Toc72399222)

[3 Progettazione 2](#_Toc72399223)

[3.1 Design dell’architettura del sistema 2](#_Toc72399224)

[3.2 Design dei dati e database 2](#_Toc72399225)

[3.3 Design delle interfacce 3](#_Toc72399226)

[4 Implementazione 4](#_Toc72399227)

[4.1 Software 4](#_Toc72399228)

[4.2 Site 9](#_Toc72399229)

[5 Test 18](#_Toc72399230)

[5.1 Protocollo di test 18](#_Toc72399231)

[5.2 Risultati test 21](#_Toc72399232)

[5.3 Mancanze/limitazioni conosciute 21](#_Toc72399233)

[6 Consuntivo 22](#_Toc72399234)

[7 Conclusioni 23](#_Toc72399235)

[7.1 Sviluppi futuri 23](#_Toc72399236)

[7.2 Considerazioni personali 23](#_Toc72399237)

[8 Bibliografia 24](#_Toc72399238)

[8.1 Sitografia 24](#_Toc72399239)

[9 Allegati 24](#_Toc72399240)

# Introduzione

## Informazioni sul progetto

Allievi: Mattia Bralla, Stefano Gnesa, Samuel Agustoni, Nikola Nikolić

Docenti: Geo Petrini, Luca Muggiasca

SAMT Informatica, Modulo 306

14.01.2021 – 20.05.2021

## Abstract

*Se stai cercando una soluzione che ti permetta di tenere traccia del numero di visitatori che passano davanti al tuo stand o alla tua attività, GuestDataLogger è l’applicativo giusto per te. Infatti si tratta di un’applicazione portable e dispone di tutto il necessario per configurare un infrastruttura di tracciamento del traffico di persone. Ciò è possibile grazie a una o più webcam che, una volta posizionate e configurate, si occuperanno del conteggio delle persone presenti e di mandare i dati raccolti al nostro database ogni dieci secondi. I dati, una volta ricevuti dal server, saranno immediatamente disponibili sul nostro sito web:* [*http://www.samtinfo.ch/gdl/site/*](http://www.samtinfo.ch/gdl/site/) *dove saranno organizzati su grafici o in forma tabellare, per consentire un’accessibilità più immediata e performante.*

*If ​you are looking for a solution that allows you to keep track of the number of visitors that pass by your stand or business, GuestDataLogger is the right application for you. In fact, it is a portable application and has everything you need to set up a people traffic tracking infrastructure. This is possible thanks to one or more webcams which, once positioned and configured, will count the number of people present and send the data collected to our database every ten seconds. Once the data has been received by the server, it will be immediately available on our website: http://www.samtinfo.ch/gdl/site/ where it will be organised in graphical or tabular form to allow for more immediate and efficient accessibility*

## Scopo

Lo scopo del progetto “GuestDataLogger v2” è, partendo da un progetto già esistente (ScanSpect), quello di creare un’infrastruttura client-server che permetta di tenere traccia del flusso di persone che visitano uno stand di un’ipotetica fiera. I dati dovranno venire raccolti da alcune webcam che vengono posizionate nei vari stand. Queste saranno gestite da un applicativo portable che si occuperà di contare le persone e di mandare i dati raccolti al server. Sul server dovrà esserci un sito web utile alla visualizzazione dei dati sottoforma di grafici e/o tabelle. Attraverso queste componenti l’utente dovrà essere in grado di tenere traccia nel tempo del traffico di visitatori della sua attività.

# Analisi

## Analisi del dominio

Il nostro progetto parte già con del materiale, ovvero un altro progetto che non era stato completato (ScanSpect). Il lavoro in questione, che era stato realizzato nell’anno scolastico 2019/2020 dagli allievi: Alessandro Aloise, Nathan Luè e Andrè da Silva, riconosceva solamente i volti inquadrate da una webcam, perciò abbiamo dovuto implementare il conteggio, la raccolta dei dati e la visualizzazione di essi attraverso la creazione di un sito web. Funzionalità che analizzeremo nel dettaglio nel corso della documentazione.  
A grandi linee, il cuore centrale del prodotto è un applicativo portable, da sviluppare in python, che si occupa di gestire le webcam e di contare le persone. Dispone di un’interfaccia grafica contenente il conteggio in tempo reale e il frame con l’analisi effettuata dalla webcam. Il software, ogni 10 secondi, spedisce i dati raccolti sotto forma di Json al server che si occupa di leggerli e di inserirli all’interno del database. I dati sono poi accessibili dal sito, organizzati in grafici e tabelle.

In poche parole il progetto consiste nel realizzare un’infrastruttura client-server dove il client è l’applicativo di raccolta dati (conteggio persone) e il server è il sito che dobbiamo sviluppare per accedere ad essi, sito che è ospitato sul dominio scolastico.

## Analisi e specifica dei requisiti

Qui di seguito riportiamo in dettaglio la lista di tutti i requisiti presenti nel mandato di questo lavoro.

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Modificare il software originale in modo che possa gestire più webcam contemporaneamente, o in alternativa, consentire l’avvio di più istanze. In entrambi i casi deve essere possibile scegliere quale webcam utilizzare per il rilevamento |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Ogni flusso di dati proveniente da una webcam deve avere un ID |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Realizzazione di un sito web |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Registrazione nel tempo del numero di visitatori |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Il sito deve avere un sistema di amministrazione utenti (gestibile da un utente admin) |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |
| **Sotto requisiti** | |
| **001** | Pagina di login e registrazione |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Il sito deve avere un sistema di autenticazione per utenti normali |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | Ogni utente deve poter gestire più flussi di dati, ad ogni flusso viene associata una webcam. In questo modo un utente che ha più stand in contemporanea, può raccogliere le statistiche di ogni sorgente dati |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-008** | |
| **Nome** | Ogni flusso dati ha un luogo, che può cambiare nel tempo (lo stand si può spostare di fiera in fiera) o avere più stand in luoghi diversi |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-009** | |
| **Nome** | Ogni utente deve poter accedere ai propri dati statistici, e filtrarli per stand/flusso, luogo, periodo (data da/a) |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-010** | |
| **Nome** | Le statistiche devono essere rappresentate principalmente come grafico su timeline, ma anche in forma tabellare (con granularità oraria) |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-011** | |
| **Nome** | L’utente può rendere pubbliche le statistiche, questi sono ricercabili e visibili graficamente in una pagina apposita del sito (analogamente alla visualizzazione privata) ma anche in formato JSON tramite chiamate http get (l’url deve essere accessibile nella visualizzazione) |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

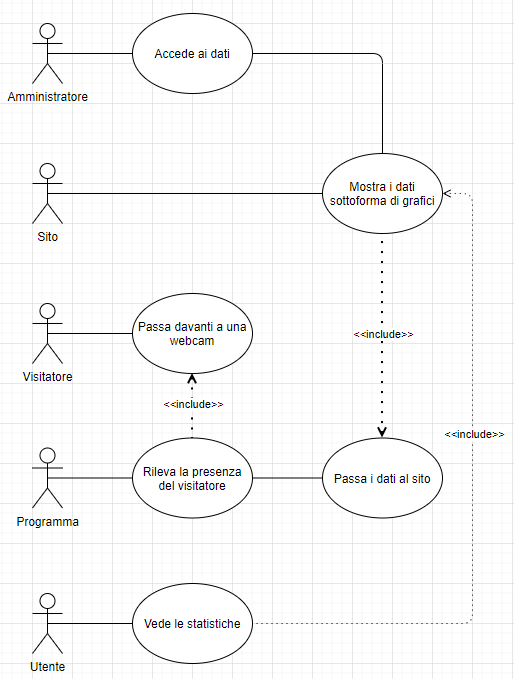
|  |  |
| --- | --- |
| **ID: REQ-012** | |
| **Nome** | L’utente può assegnare una categoria alle attività |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-013** | |
| **Nome** | Deve essere presente un manuale per l’installazione |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-014** | |
| **Nome** | Tutte le dipendenze del software devono essere incluse |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

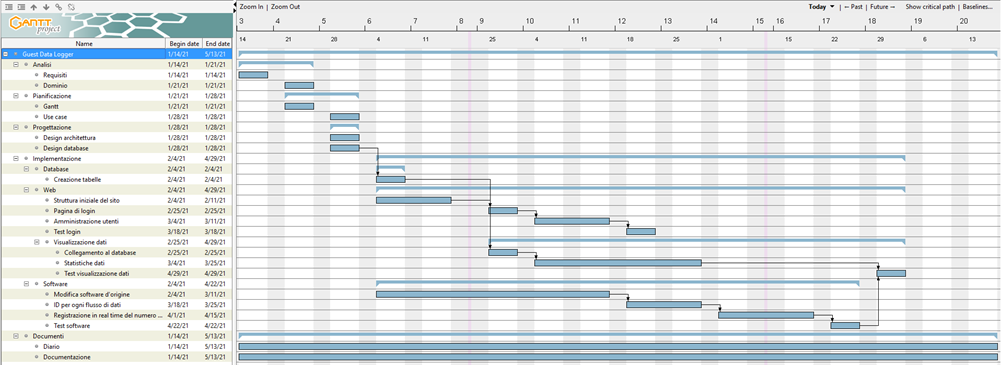
|  |  |
| --- | --- |
| **ID: REQ-015** | |
| **Nome** | Deve esserci un server per l’hosting del servizio |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

## Use case



Use Case

## Pianificazione

Abbiamo deciso di dividere questo progetto in 4 fasi ben definite. La prima di analisi, che comprende la stesura dei requisiti e l’analisi del dominio. La seconda di pianificazione dove abbiamo realizzato il diagramma di Gannt preventivo, riportato qui di seguito, e abbiamo definito i casi d’uso della infrastruttura. In seguito abbiamo inserito una fase di progettazione dove abbiamo pensato al design del database e dell’architettura del software. Infine la fase di implementazione che comprende lo sviluppo di tutto il progetto e i test.

Gantt preventivo

## Analisi dei mezzi

All’inizio del progetto ci siamo accorti che per poter sviluppare il nostro progetto, ci servivano dei mezzi utili a noi. Per la parte del software (rilevatore dei volti) abbiamo installato l’interprete Python, trovato nella pagina home di python.org. La versione da noi scaricata è la 3.8 per Windows. Per poter usare delle librerie per l’implementazione del software abbiamo dovuto installarle consultandoci sul sito di pypi.com. Per avere il sito pubblico con un database disponibile per il mantenimento ci siamo connessi all’hosting infomaniak.com, connettendoci anche al database che ci ospita l’hosting. Infine, per rilevare i volti si ha bisogno di una webcam funzionante.

### Software

Python, linguaggio di scrittura per l’interfaccia grafica dei rilevamenti dei volti.

PHP, linguaggio usato per il lato server del sito.

JavaScript, linguaggio usato per il lato client del sito.

XAMPP, WebServer usato per i test in locale del sito.

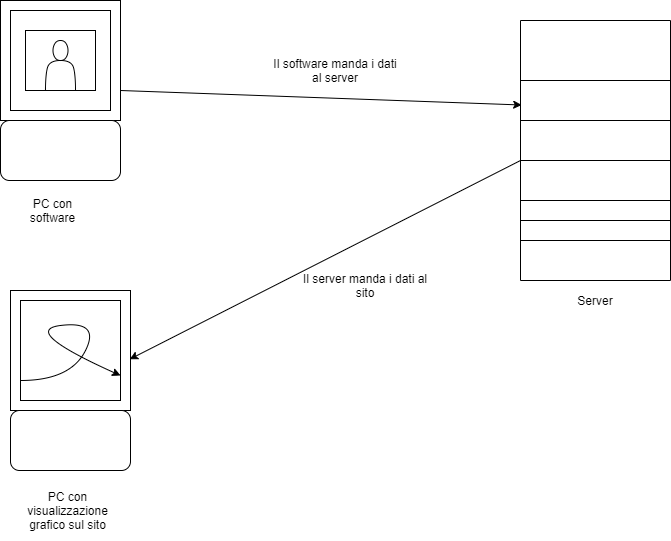
MySQL Workbench, strumento di progettazione del database usato per i test in locale.

### Hardware

L’unica piattaforma su cui è possibile utilizzare il software è Windows. Per poter utilizzare il programma per il rilevamento dei volti, bisogna avere a disposizione almeno una webcam e un computer connesso a internet sul luogo dello stand.

# Progettazione

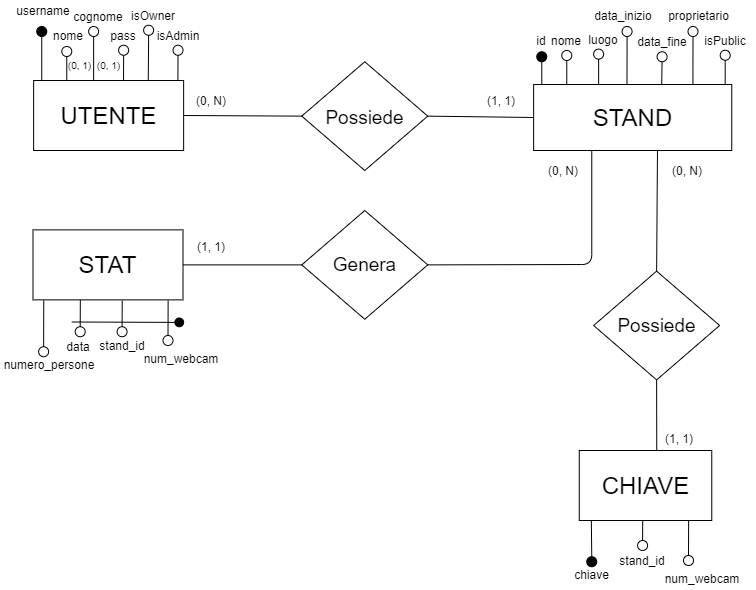
## Design dell’architettura del sistema



Design architettura

## Design dei dati e database

Nello schema seguente possiamo trovare lo schema er relativo al database che serve per la gestione dei dati (per tenere traccia delle persone rilevate dalla webcam).



Design database

## Design delle interfacce

Non abbiamo ideato un design delle interfacce iniziali perché ci siamo basati, sia per il software che per il sito, sulle interfacce del progetto precedente.

# Implementazione

## 4.1 Software

Il nostro progetto parte da uno già esistente. Perciò, prima di partire con la modifica del codice sorgente e con l’aggiunta di varie parti, dovevamo prima analizzarlo per capire il funzionamento di base, e poi capire cosa poteva essere tenuto e cosa modificato.

Il nome del file GuestDataLogger.py ha al suo interno 4 classi essenziali:

* StartWindow, che si occupa di inizializzare il programma;
* SenderData, che si occupa di inviare i dati al sito web;
* Capture, che si occupa della cattura della webcam;
* CameraWindow, che si occupa di mostrare la cattura della webcam con le cornici nei volti.

Prima della creazione delle classi bisogna prendere le configurazioni di base, quindi i siti necessari per il software, il proxy (bisogna settare nel file di configurazione il proxy, in caso necessario) e l’api\_key (che questa inizialmente sarà vuota fino a quando non ne viene inserita una nella finestra del software) ed impostarli in una variabile globale così da poter utilizzare dove è necessario.

Per poter comunicare tra le varie classi istanziamo un oggetto **Queue**, un oggetto utile a mettere informazioni necessarie in una coda senza che questa interrompi il funzionamento del programma. Nel software vengono istanziate due oggetti **Queue** che hanno come scopo di:

* Mettere in coda informazioni rilevanti al sito web (api\_key, numero di persone e data);
* Mettere in coda informazioni rilevanti al software (come la chiusura del programma).

**Classe StartWindow**

In questa classe troviamo tutta l’implementazione per la GUI di avvio, ovvero la finestra per la connessione dall’applicativo al sito.

Il metodo **def \_init\_(self, q)** ha lo scopo di settare vari attributi (necessari per altri metodi della classe) e di richiamare tutti i metodi necessari (metodi che creano ogni pezzo della GUI). I metodi che vengono richiamati sono tutti quelli che servono all’utente per l’inserimento dell’api\_key e per la scelta del numero della webcam.

def returnCameraIndexes(self):

index = 0

freeIndexes = []

while index < 255:

video = cv.VideoCapture(index, cv.CAP\_DSHOW)

if video.isOpened():

freeIndexes.append(index)

index += 1

else:

index += 1

video.release()

if len(freeIndexes) < 1:

return ["None"]

return freeIndexes

Il metodo ***returnCameraIndexes(self)*** ha lo scopo di individuare tutte le webcam collegate al PC, e quali sono quelle disponibili, ritornando un array dei valori. Per poter individuare le webcam, abbiamo chiesto aiuto a Nicholas Pigni, un ragazzo del 4° anno di informatica, che come noi stava svolgendo un progetto con Python. Ci ha fornito le informazioni necessarie per l’integrazione di questo metodo nel software.

Quando viene premuto il bottone “Start”, viene richiamato il metodo ***buttonPressed(self)*** che controlla quale webcam è stato selezionato dall’utente ed effettua un controllo sull’api\_key richiamando il metodo ***testConnection(self, api\_key)*** e verificando che questa esista veramente. Questo metodo effettua anche un controllo della connessione al sito web.

Di seguito troviamo il metodo per l’implementare il controllo sull’api\_key e sulla connessione al sito web. Il metodo ha un parametro in entrata, cioè l’api\_key, dove questa viene messa in un tipo di dato json. Dopo viene fatta una richesta di tipo post, che ha come parametri l’url (presa dalla configurazione), la variabile di tipo json e il proxy, se ne ha una (anche questa viene presa dalla configurazione). Fatta la richiesta, il sito web ritorna uno status code, con il valore di 200, dove confermano la connessione stabilita con il software e che effettivamente l’api\_key esiste. Quindi, il metodo ritorna **True** se va tutto bene, altrimenti **False**.

def testConnection(self, api\_key):

response = None

url = str(config['key\_checker'])

myJson = {

"api\_key":api\_key

}

try:

if 'http' in proxies or 'https' in proxies:

response = requests.post(url, json=myJson, proxies=proxies, timeout=5)

else:

response = requests.post(url, json=myJson, timeout=5)

logging.info(response.status\_code)

print(f'INFO: response status code {response.status\_code}')

except requests.Timeout:

logging.exception("Connection error!")

return "Connection error"

if response and response.status\_code and response.status\_code == 200:

return True

else:

return False

**Classe SenderData**

La seguente classe viene istanziata come **Thread**, una volta che il frame di inizializzazione viene distrutta. Quando viene istanziata, viene eseguita in modo che possa “lavorare” senza ostacolare il funzionamento. Grazie alle **Queue** istanziate in precedenza, le possiamo utilizzare per poter estrapolare le informazioni che dovranno essere passate al sito web. Al suo interno troviamo il metodo ***run(self)*** che ha l’obbiettivo di controllare se sono state messe informazioni per poter mandare i dati. Per far ciò, bisogna fare un continuo controllo del tempo, perché ad ogni 10 secondi vengono passati e poi mandati i dati, ed un controllo nella coda della chiusura del software. Ogni 10 secondi, viene richiamato il metodo ***send\_data(self)*** dove invia i dati al sito web. L’esecuzione del seguente metodo è quello di mettere i dati in un formato json, e mandarli al sito web. Per ricevere i dati in un formato json, viene richiamato il metodo ***data\_to\_JSON(self)*** e ritorna la variabile di tipo json, con le informazioni necessarie al sito. Per l’invio dei dati, viene eseguito nello stesso modo in cui si accerta la connessione per la prima volta, solamente che non ritorna nulla.

def send\_data(self):

json\_data = self.data\_to\_JSON()

if json\_data is not None:

url = str(config['site\_sender'])

req = requests.post(url, json= json\_data, proxies=proxies)

s = req.text

logging.info(f'request text; {s}')

print(f'INFO: request text; {s}')

**Classe Capture**

La classe Capture serve ad analizzare i frame catturati dalla webcam, a contare le facce inquadrate e a inserire i dati raccolti nella coda ogni dieci secondi. È una thread, perciò una volta istanziata e lanciata, viene eseguito il contenuto del metodo run. Tale metodo si occupa di gestire tutte le funzionalità della classe.

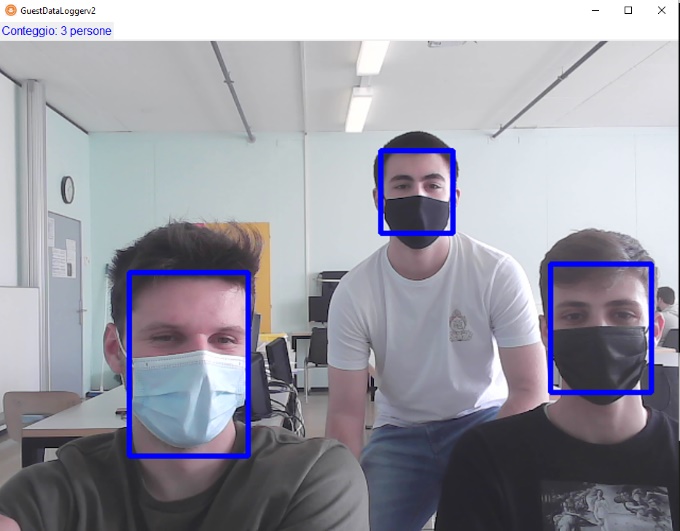
Prima di tutto riconosce le facce contenute nel frame catturato dalla webcam e genera un riquadro di colore blu attorno ad ogni faccia presente nell’inquadratura. Questo grazie al metodo ***getFaceBox()***, metodo che abbiamo riciclato dal progetto ScanSpect che ci è stato consegnato inizialmente.

Immagine rilevamento volto

Per il riconoscimento facciale utilizziamo la libreria openCV che fornisce due file assolutamente necessari per il funzionamento del nostro software. Vanno scaricati e importati:

self.faceProto = "opencv\_face\_detector.pbtxt"

self.faceModel = "opencv\_face\_detector\_uint8.pb"

self.faceNet = cv.dnn.readNet(self.faceModel, self.faceProto)

Questi due file definiscono il modello e il prototipo di una faccia e ci servono per riconoscere le persone all’interno del frame. Senza di essi il funzionamento del software sarebbe compromesso, non funzionerebbe. Una volta importati va creata una rete neurale a cui vanno passati questi due file. La rete neurale va passata come parametro al metodo che si occupa di generare i riquadri attorno alle facce (***getFaceBox()***).

Per implementare il conteggio abbiamo creato un metodo ***detectFaces()*** che si occupa di contare i riquadri presenti nel frame in quell’istante.

def detectFaces(self, cap):

frameFace = None

count = 0

hasFrame, frame = cap.read()

if not hasFrame:

cv.waitKey()

else:

frameFace, bboxes = self.getFaceBox(self.faceNet, frame)

count = len(bboxes)

return {'frame':frameFace, 'count':count }

Legge il frame attuale della camera, per ogni volto che vede genera un riquadro e in base al numero di riquadri presenti aggiorna il contatore count. Infine ritorna il frame attuale con le cornici generate e il contatore. Questo metodo lo richiamiamo nel run per tenere traccia del numero di persone corrente.

**Classe CameraWindow**

La classe in questione ha come scopo di estrarre dalla coda la cattura dello schermo (la cattura della webcam viene presa dalla classe **Capture**). I metodi iniziali come ***\_build\_frame(self)*** e ***\_build\_label(self)*** sono utilizzati per poter creare la GUI dove verrà mostrata la cattura della webcam.

def \_refresh(self, data\_queue, stop\_event):

logging.info('frame refresh started')

print(f'INFO: frame refresh started')

exit\_event = False

while not exit\_event:

data = None

#prende la grandezza della queue

s = data\_queue.qsize()

if s > 0:

#cicla per ogni frame prendendo sempre l'ultimo frame

for i in range(s):

try:

data = data\_queue.get(False) #get non blocking

except Exception as e:

data = None

if data:

try:

self.update\_picture(cv.cvtColor(data['frame'], cv.COLOR\_BGR2RGB))

except Exception as e:

logging.exception('error updating frame')

try:

self.update\_counter(data['count'])

except Exception as e:

logging.exception('error updating count')

exit\_event=stop\_event.is\_set()

logging.debug('frame refresh ended')

Il metodo ***\_refresh(self, data\_queue, stop\_event)*** fa visualizzare la cattura della webcam nella GUI. I parametri in entrata utili a questo metodo sono la coda della cattura, dove viene inserita la cattura stessa della webcam, e la coda dell’uscita. La procedura che viene svolta all’interno è la seguente:

1. Un ciclo per continuare a prendere i dati e a mostrarli, con la condizione dell’uscita (in caso viene cliccato il bottone di uscita, il ciclo si blocca, ed esce dal metodo);
2. Estrae la grandezza della coda della cattura;
3. Prende l’ultima cattura della webcam;
4. Richiama il metodo ***update\_capture(self, frame, resize=True)*** passandogli come parametro la cattura;
5. Richiama il metodo ***update\_counter(self, count)*** passandogli come parametro il numero di persone attuali.

Il metodo ***update\_capture(self, frame, resize=True)*** viene utilizzato per estrarre la cattura, e poi inserirla nella **Label** che verrà mostrato a schermo il risultato finale. Prima di inserire la cattura, viene fatta diventare un’immagine e poi viene scalata, con il metodo ***rescale\_frame(self,frame)*** così che si possa vedere in dimensioni più grandi, a dipendenza della grandezza del frame. Dopodiché l’immagine viene settata nel **Label** che mostrerà la cattura.

Il metodo ***rescale\_frame(self,frame)*** prende le dimensioni dello schermo (larghezza e altezza), le dimensioni del frame (larghezza e altezza) e viene calcolata per individuare il massimo della dimensione della cattura.

def rescale\_frame(self, frame):

self.top.update()

frameWidth = self.top.winfo\_width()

frameHeight = self.top.winfo\_height()

width = int(frame.size[0])

height = int(frame.size[1])

multiplierWidth = frameWidth / width

multiplierHeight = frameHeight / height

#larghezza minore di altezza

if multiplierWidth < multiplierHeight:

dim = (int(width\*multiplierWidth), int(height\*multiplierWidth))

self.label.place()

#altezza minore di larghezza

else:

dim = (int(width\*multiplierHeight), int(height\*multiplierHeight))

self.label.place(relx = 0.5,

rely = 0.5,

anchor = 'center')

return frame.resize(dim)

Il metodo ***update\_counter(self, counter)*** ha lo scopo di configurare la stringa indicante il numero delle persone rilevate. Questa aggiorna la scritta della **Label** che si trova in alto del finestra del software.

def update\_counter(self, count):

if(count > 1 or count == 0):

self.labelInfo.configure(text = "Conteggio: " + str(count) + " persone")

else:

self.labelInfo.configure(text = "Conteggio: " + str(count) + " persona")

In questa classe troviamo anche un metodo chiamato ***run(self)*** che viene utilizzata come se fosse una **Thread**. Questa viene richiamata subito dopo istanziata la classe stessa, e al suo interno vediamo una riga codice molto importante:

self.rth = thread.Thread(target=self.\_refresh, args=[self.frames\_queue, self.stop\_event])

La procedura che svolge è quella di, tramite il “*target*”, settare un metodo a **Thread**, ovvero far si che il metodo lavori come una **Thread** senza che ostacoli l’elaborazione dei metodi della classe.

## 4.2 Site

Il sito è stato implementato utilizzando il framework MVC per PHP, che funziona tramite delle classi definite model, che si occupano della logica dell’applicazione, delle view, che si occupano della presentazione, e dei controller, che si occupa della comunicazione tra model e view.

Nel nostro sito ogni pagina ha un proprio controller che si occupa poi di richiamare i model necessari per l’elaborazione dei dati per la sottopagina richiesta e richiamare tutte le view necessare a mostrare suddetta sottopagina.

La lista completa dei file controller, model e view è la seguente:

* controller
  + administrator.php
  + errors.php
  + graph.php
  + home.php
  + key.php
  + login.php
  + register.php
  + stand.php
  + user.php
* models
  + administratormodel.php
  + databasemodel.php
  + graphmodel.php
  + keymodel.php
  + loginmodel.php
  + scriptmodel.php
  + standmodel.php
  + usermodel.php
* views
  + administrator
    - index.php
    - stand.php
    - stands.php
    - user.php
    - users.php
  + error
    - administratorError.php
    - databaseError.php
    - dateError.php
    - errorHeader.php
    - loginError.php
    - noGraphs.php
    - passwordError.php
    - sessionError.php
    - unexpectedError.php
    - usernameError.php
    - usernameTakenError.php
  + graph
    - graph.php
    - graphForm.php
    - index.php
    - tabledata.php
  + home
    - index.php
  + key
    - index.php
  + login
    - index.php
  + register
    - index.php
  + shared
    - footer.php
    - header.php
    - nav.php
  + stand
    - addStand.php
    - index.php
  + user
    - index.php

**Models**

La funzione principale dei model è quella di connettersi al database, eseguire una query ed eventualmente elaborare i dati ricevuti prima di ritornarli. Per fare ciò, abbiamo creato la classe DatabaseModel per non dover rifare i primi due passaggi in tutti i model.

**DatabaseModel**

La classe DatabaseModel possiede solo due funzioni: static function executeQuery($query) e static function executeSelectQuery($query), che sono molto simili alla base. Entrambi istanziano un oggetto mysqli, utile alla connessione al server MySQL, e poi eseguono la query ricevuta. La differenza sostanziale è il valore ritornato. Se la connessione al server non è riuscita entrambi ritornano la stringa “MYSQL”, però, mentre la funzione executeQuery($query) ritorna TRUE se la query è andata a buon fine e FALSE nel caso contrario, la funzione executeSelectQuery($query) viene utilizzata per le query di tipo select, quindi si occupa prima di creare un array associativo contenente tutti i dati prima di ritornare suddetto array.

Entrambi i metodi sono statici per permettere agli altri model di eseguire delle querry senza dover instanziare la classe DatabaseModel.

**AdministratorModel, KeyModel, ScriptModel, StandModel e UserModel**

Le classi AdministratorModel, KeyModel, ScriptModel, StandModel e UserModel si occupano solo di eseguire le query necessarie tramite la classe DatabaseModel senza elaborare dati, nè prima nè dopo. Ad esempio, la funzione fetchInfo($username) della classe UserModel, utile a ritornare le informazioni di un utente, è così:

function fetchInfo($username){

$query = "SELECT username, nome, cognome FROM utente WHERE username = '$username'";

$info = DatabaseModel::executeSelectQuery($query);

return $info;

}

Tutte le funzioni di queste classi sono simili a questa e differiscono solo nella query.

**GraphModel**

Questa classe ha tutte le funzioni utili alla richiesta dei dati statistici degli stand e all’elaborazione di essi.

Quando il controller della pagina “Grafici” istanzia un oggetto GraphModel richiama anche la funzione function setCamp($filter) che serve ad impostare l’attributo $camp in modo da rispettare il filtro passato.

function setCamp($filter){

$camp = "data";

if($filter != "default"){

if($filter == "YEAR" || $filter == "DATE"){

$camp = "$filter($camp)";

}else{

$camp = "DATE\_FORMAT(data,'%Y-%m";

if($filter == "HOUR" || $filter == "MINUTE"){

$camp = $camp . "-%d %H";

}

if($filter == "MINUTE"){

$camp = $camp . ":%i";

}

$camp = $camp . "')";

}

}

$this->camp = $camp;

}

Questo attributo verrà poi utilizzato in tutte le query per modificare la granularità oraria dei dati.

**LoginModel**

La classe LoginModel serve a gestire il login e la registrazione di un utente. Oltre ad eseguire le query con la classe DatabaseModel, questa classe codifica e decodifica in hash la password ricevuta e poi setta delle variabili di sessione contenenti le info dell’utente così da non dover richiederle dal database ogni volta che sono necessarie.

**Stand**

La pagina degli stand consiste in una tabella contenente tutti gli stand dell’utente con: nome, luogo, data di inizio, data di fine e dei pulsanti con diverse funzioni.

Ogni volta che viene caricata la pagina, viene effettuata una query al database sulla tabella **stand**, che permette di ricavare i dati che vengono mostrati, usando questa funzione:

function fetchStands($proprietario){

$query = "SELECT id, nome, luogo, data\_inizio, data\_fine, isPublic

              FROM stand

              WHERE proprietario = '$proprietario'";

    $stands = DatabaseModel::executeSelectQuery($query);

    return $stands;

}

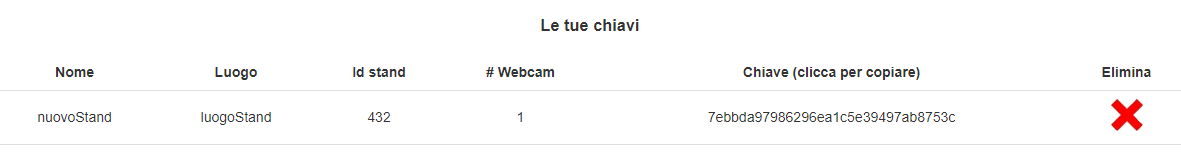
Per la modifica dei dati in tempo reale è stata usata una libreria di jQuery: Tabledit. Essa consiste nel cliccare sul valore da modificare, inserire il nuovo valore e premere “Enter” per rendere effettiva la modifica.

**Aggiungere stand**

Per l’aggiunta di uno stand è stato usato un form che contiene gli input necessari. Cliccando su “Aggiungi”, il

form punterà alla funzione **addStand** del controller, passandogli i dati col metodo **POST**. A questo punto il controller istanzia un nuovo **StandModel**, e, usando la funzione addStand di quest’ ultimo, aggiunge lo stand al database. La pagina viene poi ricaricata dal controller per rendere visibile la modifica.

**Chiavi**



Come per gli stand, anche per la pagina delle chiavi viene usata una funzione per fare una query al database, questa volta sulla tabella **chiave**.

**Aggiungere una chiave**

Quando da uno stand si clicca sul pulsante “Ricevi chiave”, viene chiamata la funzione **getKey** del file api.php, che produce una nuova chiave univoca.

function getKey(){

    $output = null;

    for($loop = 0; $loop <= 31; $loop++) {

        for($isRandomInRange = 0; $isRandomInRange === 0;){

            $isRandomInRange = isRandomInRange(findRandom());

        }

        $output .= html\_entity\_decode('&#' . $isRandomInRange . ';');

    }

    $output = $output . round(microtime(true) \* 1000);

    return md5($output);

}

La funzione findRandom ritorna un numero tra 48 e 122, mentre isRandomInRange ritorna il numero passato se è tra 48 e 57 o se è tra 65 e 90. Questa operazione viene fatta 32 volte e ad ogni ciclo il numero viene decodificato in carattere ASCII e aggiunto ad una stringa.  
Alla fine si avra la stringa completa lunga 32 caratteri, a cui viene aggiunto il tempo Unix in millisecondi, per evitare eventuali improbabili doppioni. Alla fine la stringa viene crittografata in md5 e ritornata.  
Per aumentare la flessibilità, le chiavi funzionano anche prima e dopo le date definite durante la creazione dello stand.

**Tabledit**

Come accennato in precedenza, la modifica dei dati in tempo reale nella pagina degli stand viene fatta usando Tabledit.

Passando l’id “#t1” della tabella si invoca Tabledit, come parametri modificabili gli si mettono quelli desiderati, in questo caso “nome”, “luogo”, “data\_inizio”, “data\_fine”. Si passa anche un identificatore, in questo caso ho usato una colonna nascosta che ho chiamato “id”.

Con “url” si specifica lo script a cui devono essere inviati i dati da modificare.

Con “onDraw” si specifica l’azione da eseguire quando la tabella viene cliccata, in questo caso controllo se è stata cliccata una colonna con la data, se questo dovesse essere il caso, viene mostrato un datepicker.

Con “onAjax” si specifica l’azione da eseguire quando vengono inviati i dati, in questo caso viene fatto un controllo sulle date, per assicurarsi che la data di inizio non sia oltre a quella di fine.

$(document).ready(function(){

    $('#t1').Tabledit({

            deleteButton: false,

            editButton: false,

            columns: {

                    identifier: [0, 'id'],

                    editable: [[1, 'nome'], [2, 'luogo'], [3, 'data\_inizio'], [4, 'data\_fine']]

            },

            hideIdentifier: true,

            url: "application/scripts/stand\_edit.php",

            onDraw: function() {

                $('table tr td:nth-child(4) input').each(function() {

                    $(this).datepicker({

                        format: 'yyyy-mm-dd',

                        todayHighlight: true

                    });

                });

                $('table tr td:nth-child(5) input').each(function() {

                    $(this).datepicker({

                        format: 'yyyy-mm-dd',

                        todayHighlight: true

                    });

                });

            },

            onAjax: function(action, serialize) {

                if (action === 'edit') {

                    var start = new Array();

                    var end = new Array();

                    var validate = true;

                    var i = 0;

                    $("table tr td :input").each(function(){

                        var today = new Date();

                        var dd = String(today.getDate()).padStart(2, '0');

                        var mm = String(today.getMonth() + 1).padStart(2, '0');

                        var yyyy = today.getFullYear();

                        today = yyyy + '-' + mm + '-' + dd;

                        start[i] = $("table tr:nth-child(" + (i+1) + ") td:nth-child(4) :input").val()

                        end[i] = $("table tr:nth-child(" + (i+1) + ") td:nth-child(5) :input").val()

                        if(start[i] > end[i]){

                            alert("La data di inizio non può essere dopo quella di fine.");

                            validate = false;

                            return false;

                        }

                        i++;

                    });

                    if(!validate){

                        return false;

                    }

                    $('.datepicker').remove();

                }

            }

    });

});

**Eliminazione**

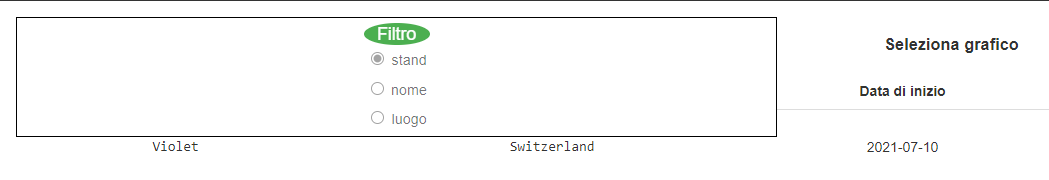
Quando viene premuta la croce rossa accanto a uno stand o una chiave, questi vengono eliminati usando una query al database fatta nei loro rispettivi model. La pagina viene poi ricaricata dal controller per rendere visibile la modifica.

**Grafici**

La pagina dei grafici consiste in una tabella contenente le informazioni di tutti gli stand dell’utente e degli stand pubblici.



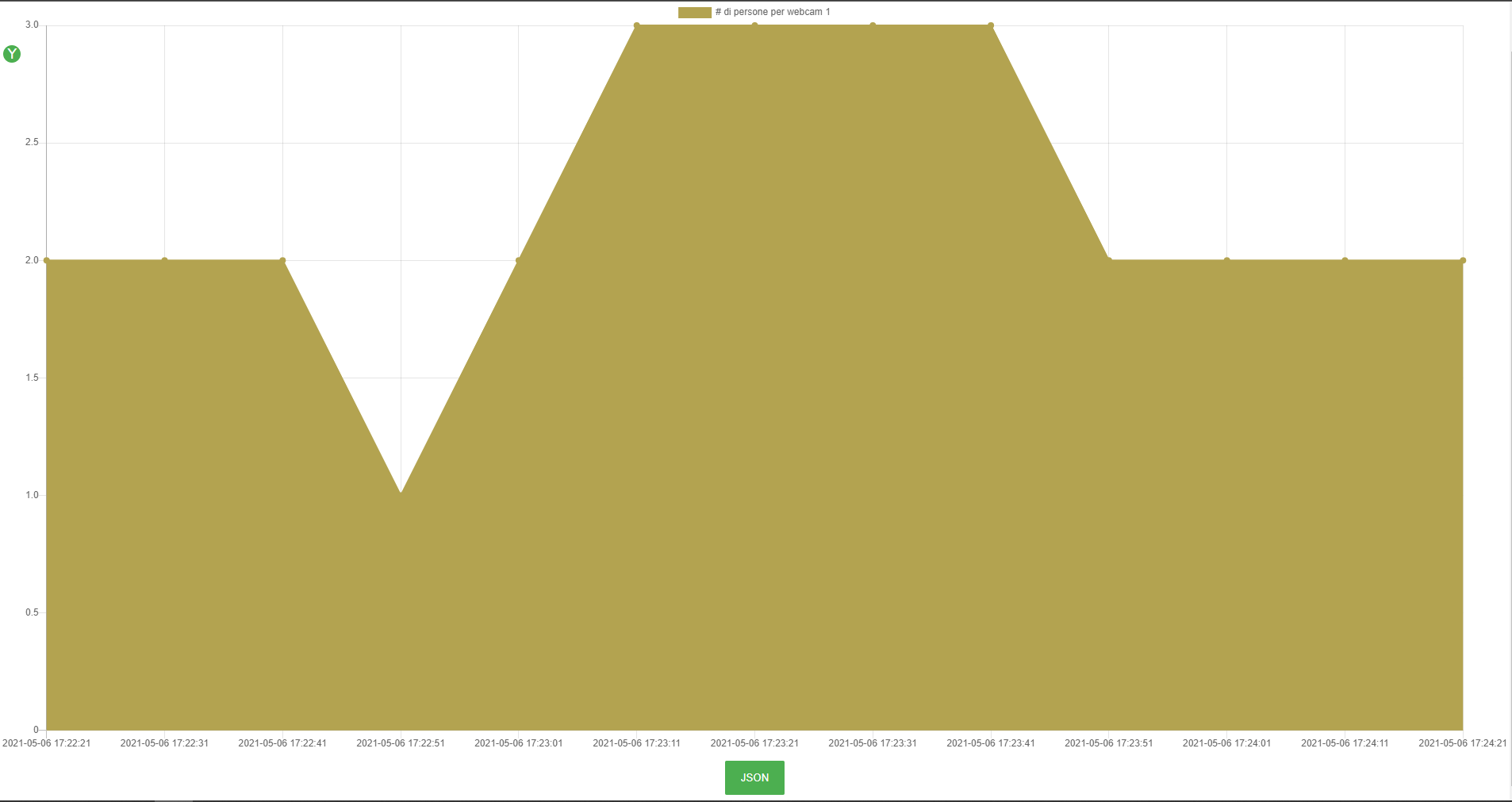
Andando in hover sul simbolo “Y” si aprirà un piccolo menu per il filtraggio degli stand.



Ogni volta che la pagina si ricarica, viene effettuata una semplice query al server per ricevere tutti gli stand con almeno un dato statistico che sono pubblici o che appartengono all’utente se è loggato in base al filtro.

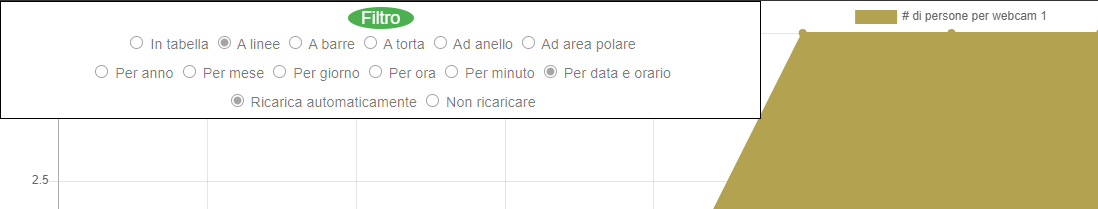
Il filtro funziona solo per la visualizzazione degli stand e non ha effetto sui grafici perché la funzione non è stata implementata per mancanza di tempo. Caricando un grafico con il filtro nome o luogo attivo causerà l’applicazione di caricare il grafico del primo stand in ordine per id del raggruppamento.

Premendo il bottone “Mostra grafici” viene caricata una pagina con i grafici dello stand, realizzata con l’ausilio della libreria Chart.js.



La pagina ha un grafico per ogni webcam dello stand. La ragione per cui avviene questa separazione è perché implementare un grafico unico con tutte le webcam avrebbe richiesto un controllo aggiuntivo per correggere gli sbalzi di tempo dal momento che le webcam potrebbero essere attive in momenti diversi o comunque i dati vengono inviati con orari diversi che causano incongruità nella linea temporale, e questo avrebbe richiesto troppo tempo.

Andando in hover sul simbolo “Y” si aprirà anche qui un piccolo menu con diverse opzioni. La prima opzione è il tipo di grafico, la seconda è la granularità oraria e la terza è se ricaricare automaticamente o meno.



Per default, una pagina appena aperta avrà le opzioni “A linee”, “Per data e orario” e “Ricarica automaticamente” attive.

Per prendere i dati dal server vengono effettuate tre query.

La prima è necessaria per raccogliere il numero delle webcam dello stand:

"SELECT DISTINCT(num\_webcam) FROM stat WHERE stand\_id = $stand\_id ORDER BY num\_webcam;"

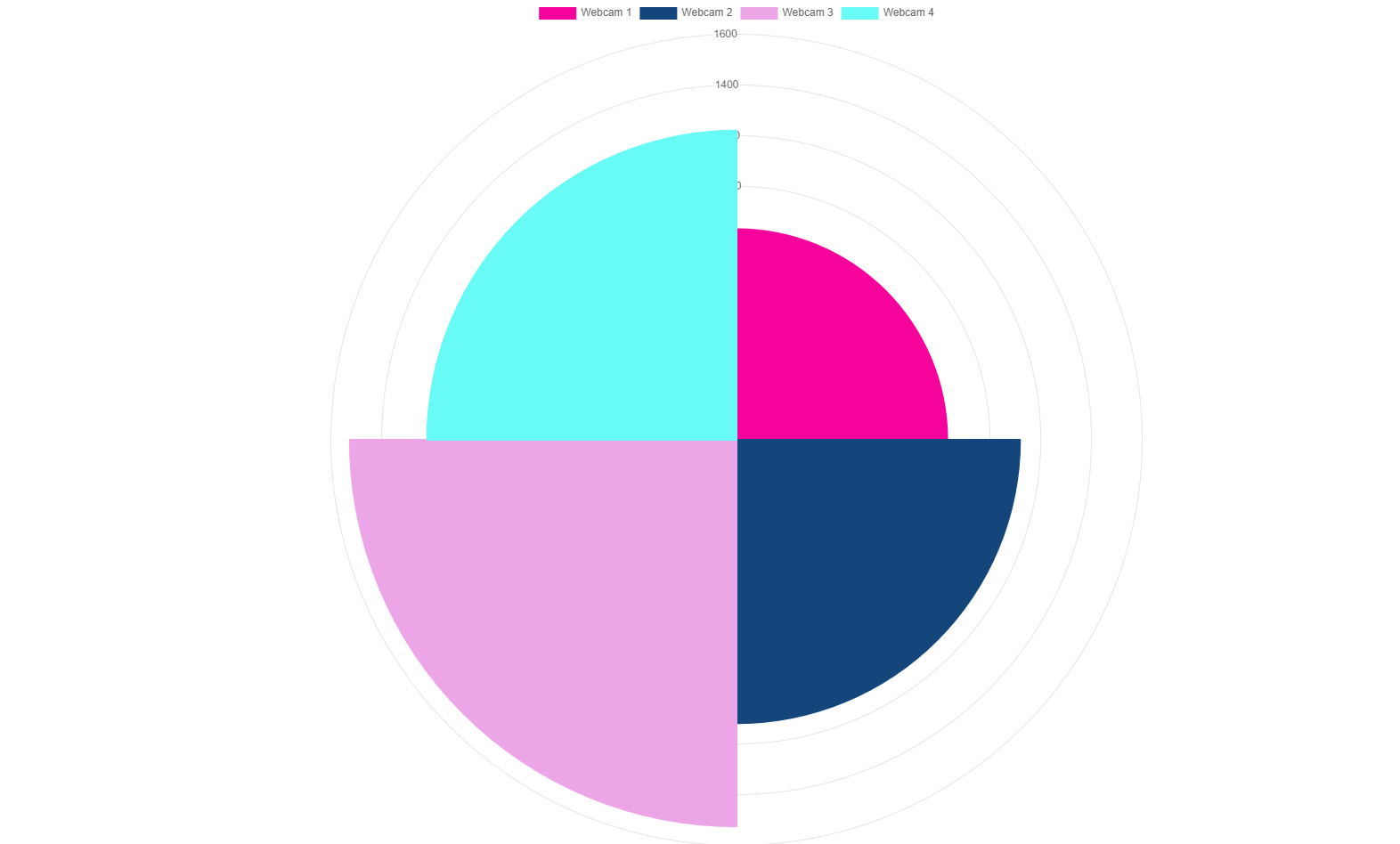
la seconda è necessaria per raccogliere le date/orari in base alla granularità:

"SELECT $camp FROM stat WHERE stand\_id = $stand\_id AND num\_webcam = $num\_webcam GROUP BY $camp ORDER BY $camp;"

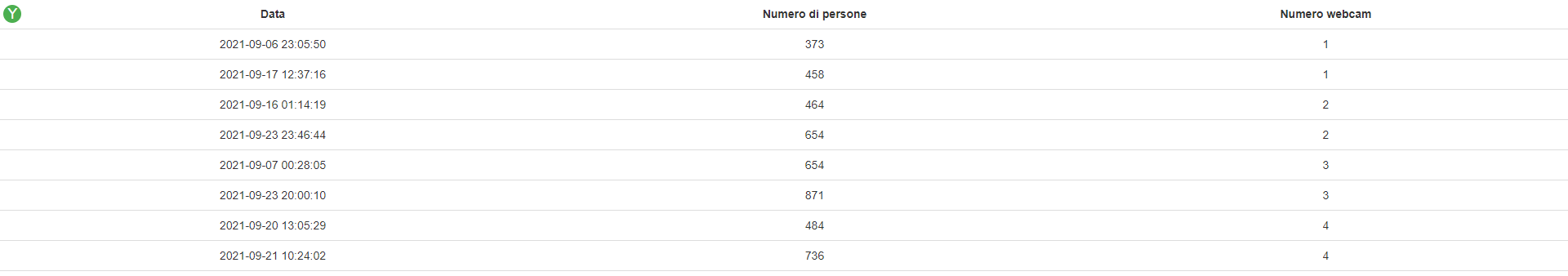
e la terza serve a raccogliere i dati in sé:

"SELECT SUM(numero\_persone) AS numero\_persone FROM stat WHERE stand\_id = $stand\_id AND num\_webcam = $num\_webcam GROUP BY $camp ORDER BY $camp;"

I grafici a torta, ad anello e ad area polare funzionano in maniera leggermente diversa.

In questi, infatti, una linea temporale non ha senso, e perciò mostrano solamente il totale dei visitatori catturati da ogni webcam, quindi l’unica query effettuata serve per raccogliere i dati:

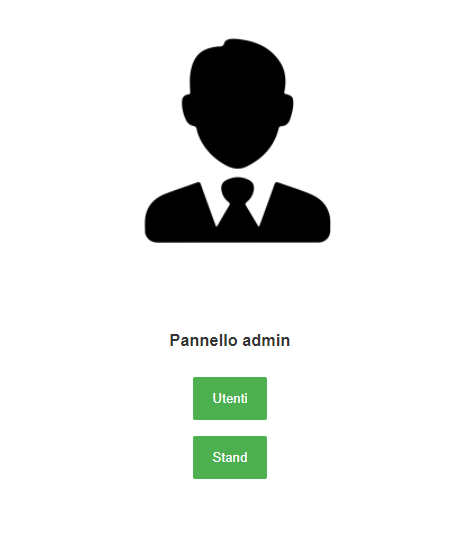
"SELECT SUM(numero\_persone) FROM stat WHERE stand\_id = $stand\_id GROUP BY num\_webcam ORDER BY num\_webcam;"



Anche la raccolta dei dati per la versione tabellare avviene in maniera diversa. Dal momento che non è necessario separare i dati delle date/orari da quelli del numero di visitatori, viene effettuata un’unica query per raccogliere tutti i dati in una volta:

"SELECT $camp AS 'data', SUM(numero\_persone) AS numero\_persone, num\_webcam FROM stat WHERE stand\_id = $stand\_id GROUP BY $camp, num\_webcam ORDER BY num\_webcam, $camp;"

Una cosa importante da notare è che, sia in questa pagina che in quella della lista dei grafici, quando si preme una opzione nei menu visualizzabili andando in hover sul simbolo “Y” le pagine vengono aggiornate subito per riflettere le nuove opzioni.

**Pannello Admin**

Il pannello admin serve per la gestione degli utenti, e si possono modificare lo username, il nome, il cognome e i diritti da admin oltre che alla possibilità di eliminare l’utente stesso. Inoltre, vi è la possibilità di modificare il nome e il luogo degli stand. Il ragionamento alla base della scelta di lasciare la libertà all’amministratore di modificare ogni dato che è un testo è che l’amministratore dovrebbe avere la possibilità di poter cambiare qualsiasi cosa che un utente potrebbe mettere di offensivo, sensibile, etc.

Tutto il procedimento avviene tramite semplici query al server.

# Test

## Protocollo di test

Nelle seguenti tabelle troviamo tutti i test effettuati da noi per verificare il funzionamento dei requisiti richiesti.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-005 | **Nome:** | Il sito deve avere un sistema di amministrazione utenti (gestibile da un utente admin) |
| **Descrizione:** | Una volta fatta la registrazione l’utente riceve un ruolo che, accedendo al sito (login), avrà dei permessi ben precisi per il ruolo. | | |
| **Prerequisiti:** | Pagina di login e registrazione. | | |
| Procedura: | 1. Accedere alla pagina iniziale. 2. Cliccare il bottone “Register”. 3. Registrarti. 4. Cliccare sul bottone “Login”. | | |
| Risultati attesi: | L’utente può svolgere qualsiasi attività consentita dal ruolo. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-001 | **Nome:** | Modificare il software originale in modo che possa gestire più webcam contemporaneamente, o in alternativa, consentire l’avvio di più istanze. In entrambi i casi deve essere possibile scegliere quale webcam utilizzare per il rilevamento |
| **Descrizione:** | Istanziare più volte lo stesso software per avviarle con diverse webcam. | | |
| **Prerequisiti:** |  | | |
| Procedura: | 1. Cliccare sull’eseguibile del software 2. Selezionare una webcam 3. Cliccare nuovamente sull’eseguibile del software 4. Selezionare un’altra webcam disponibile | | |
| Risultati attesi: | L’utente può svolgere qualsiasi attività consentita dal ruolo. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-004 | **Nome:** | Registrazione nel tempo del numero di visitatori |
| **Descrizione:** | Una volta istanziato il software viene mostrato in tempo reale il numero dei visitatori | | |
| **Prerequisiti:** |  | | |
| Procedura: | 1. Cliccare sull’eseguibile del software 2. Selezionare una webcam 3. Controllare il numero delle persone in alto | | |
| Risultati attesi: | Il numero delle persone in totali rilevate | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005  REQ-010 | **Nome:** | Le statistiche devono essere rappresentate principalmente come grafico su timeline, ma anche in forma tabellare (con granularità oraria) |
| **Descrizione:** | Nella visualizzazione dei grafici viene anche mostrato il JSON del grafico | | |
| **Prerequisiti:** |  | | |
| Procedura: | 1. Andare sul sito 2. Cliccare su “Stand” 3. Cliccare su un grafico di uno stand 4. Cliccare su “JSON” | | |
| Risultati attesi: | Visualizzazione del JSON del grafico | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006  REQ-010 | **Nome:** | Le statistiche devono essere rappresentate principalmente come grafico su timeline, ma anche in forma tabellare (con granularità oraria) |
| **Descrizione:** | Nella visualizzazione dei grafici viene anche mostrato il JSON del grafico | | |
| **Prerequisiti:** |  | | |
| Procedura: | 1. Andare sul sito 2. Cliccare su “Stand” 3. Cliccare su un grafico di uno stand 4. Cliccare su “JSON” | | |
| Risultati attesi: | Visualizzazione del JSON del grafico | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-007  REQ-002 | **Nome:** | Autenticazione utenti normali |
| **Descrizione:** | Il sito deve avere un sistema di autenticazione per utenti normali. | | |
| **Prerequisiti:** | Avere un account registrato al sito. | | |
| **Procedura:** | 1. Cliccare in alto a destra su “Accesso” 2. Inserire il nome utente e la password, poi cliccare “Login” | | |
| **Risultati attesi:** | Se I dati inseriti sono corretti, viene eseguito l’accesso e si viene indirizzati alla home page, altrimenti compare un messaggio di errore. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-008  REQ-003 | **Nome:** | Gestione flussi di dati |
| **Descrizione:** | Il sito deve avere un sistema di autenticazione per utenti normali. | | |
| **Prerequisiti:** | Ogni utente deve poter gestire più flussi di dati, ad ogni flusso viene associata una webcam. In questo modo un utente che ha più stand in contemporanea, può raccogliere le statistiche di ogni sorgente dati. | | |
| **Procedura:** | 1. Cliccare su “Aggiungi stand” 2. Inserire i dati necessari e cliccare “Aggiungi” 3. Cliccare su “I miei stand” 4. Cliccare su “Ricevi chiave” 5. Tornare al punto 3 se si vuole aggiungere un nuovo flusso di dati | | |
| **Risultati attesi:** | Si vedono gli stand con le loro chiavi associate e i numeri delle webcam | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-009  REQ-004 | **Nome:** | Gestione flussi di dati |
| **Descrizione:** | I luoghi possono cambiare e possono esserci più stand in luoghi diversi | | |
| **Prerequisiti:** | Ogni flusso dati ha un luogo, che può cambiare nel tempo (lo stand si può spostare di fiera in fiera) o avere più stand in luoghi diversi. Tutto questo deve essere gestito. | | |
| **Procedura:** | 1. Cliccare su “I miei stand” 2. Cliccare sullo stand che si vuole modificare, all’altezza della colonna “Luogo” 3. Inserire il nuovo luogo e premere “Enter” | | |
| **Risultati attesi:** | Il luogo dello stand è cambiato | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-010  REQ-009 | **Nome:** | Accesso ai propri dati statistici con filtro per stand, luogo e periodo. |
| **Descrizione:** | Ogni utente deve poter accedere ai propri dati statistici, e filtrarli per stand/flusso, luogo, periodo (data da/a) | | |
| **Prerequisiti:** | Accesso con utente che possiede almeno uno stand e dei dati | | |
| **Procedura:** | 1. Andare nella pagina “Grafici” 2. Selezionare un filtro 3. Cliccare “Mostra grafici” | | |
| **Risultati attesi:** | L’utente ha accesso ai propri dati filtrati | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-011  REQ-010 | **Nome:** | Statistiche rappresentate come grafico timeline |
| **Descrizione:** | Le statistiche devono essere rappresentate principalmente come grafico su timeline, ma anche in forma tabellare (con granularità oraria) | | |
| **Prerequisiti:** | Accesso con utente che possiede almeno uno stand e dei dati | | |
| **Procedura:** | 1. Andare nella pagina “Grafici” 2. Cliccare “Mostra grafici” 3. Selezione il tipo di grafico e la granularità oraria | | |
| **Risultati attesi:** | I dati sono visibili in diverse forme e con granularità cambiabile | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-012  REQ-011 | **Nome:** | Si possono pubblicare i propri dati |
| **Descrizione:** | L’utente può rendere pubbliche le statistiche, questi sono ricercabili e visibili graficamente in una pagina apposita del sito (analogamente alla visualizzazione privata) ma anche in formato JSON tramite chiamate http get (l’url deve essere accessibile nella visualizzazione) | | |
| **Prerequisiti:** | Utente con almeno uno stand e dei dati | | |
| **Procedura:** | 1. Andare nella pagina “I miei stand” 2. Cliccare “Rendi pubblico” 3. Fare logout 4. Andare nella pagina “Grafici” 5. Cliccare “Mostra grafici” | | |
| **Risultati attesi:** | Lo stand è visibile pubblicamente anche ad un utente non registrato | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-013  REQ-011 | **Nome:** | I dati sono disponibili in formato JSON |
| **Descrizione:** | L’utente può rendere pubbliche le statistiche, questi sono ricercabili e visibili graficamente in una pagina apposita del sito (analogamente alla visualizzazione privata) ma anche in formato JSON tramite chiamate http get (l’url deve essere accessibile nella visualizzazione) | | |
| **Prerequisiti:** | Accesso con utente che possiede almeno uno stand e dei dati | | |
| **Procedura:** | 1. Andare nella pagina “Grafici” 2. Cliccare “Mostra grafici” 3. Cliccare “JSON” | | |
| **Risultati attesi:** | I dati vengono mostrati in formato JSON ed è presente un link per i dati attraverso una richiesta GET | | |

## Risultati test

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Stato** | **Note** |
| TC-001 | OK |  |
| TC-002 | OK |  |
| TC-003 | OK |  |
| TC-004 | OK |  |
| TC-005 | OK |  |
| TC-006 | OK |  |
| TC-007 | OK |  |
| TC-008 | OK |  |
| TC-009 | OK |  |
| TC-010 | Parziale | L’utente ha accesso ai propri dati, ma questi possono essere filtrati solo per stand. È presente la possibilità di filtrare gli stand anche per nome e luogo, ma questo filtro non si applica ai dati |
| TC-011 | OK |  |
| TC-012 | OK |  |
| TC-013 | OK |  |

## Mancanze/limitazioni conosciute

Sito:

L’opzione per il ricaricamento automatico ricarica la pagina dei grafici ogni 15 secondi piuttosto che ricaricare i grafici stessi per mancanza di tempo, in quanto implementare una funzione per aggiornare lo script JavaScript utilizzato dalla libreria Chart.js per visualizzare i grafici e aggiornare i canvas per mostrare i nuovi cambiamenti avrebbe richiesto troppo tempo.

I dati degli stand sono rappresentati tramite grafici singoli per tutte le webcam perché implementare un grafico unico avrebbe richiesto un controllo aggiuntivo per correggere gli sbalzi di tempo causati dal momento che le webcam potrebbero essere attive in momenti diversi o che i dati vengono inviati con orari diversi, che causano incongruità nella linea temporale, e questo avrebbe richiesto troppo tempo.

# Consuntivo

# Conclusioni

Grazie a questo progetto, e al suo sviluppo, ci siamo trovati in situazioni che potrebbero riscontrarsi nel futuro, nel mondo del lavoro. Lavorando in team siamo riusciti ad unire le due parti (software e web) anche se non è stato sempre semplice. La comunicazione era alla base dell’unione del progetto, perché ad ogni cambiamento del software o del sito, poteva essere un problema per l’altra parte, riscontrando qualche problema. Una volta che siamo riusciti a collegare le parti, non dovevamo fare altro che sistemare quei problemi senza dover preoccuparci sempre della comunicazione. Abbiam potuto apprendere fino alla fine che lavorare in team può essere molto difficoltoso perché ogni persona del team ha un suo modo di implementare diverso da altri, però può essere anche molto comodo perché potevamo aiutarci tra di noi.

## Sviluppi futuri

Sofware:

Per avere un software molto più vantaggioso per l’utente che lo utilizza si potrebbe implementare la tastiera come input di ascolto, senza dover ogni volta cliccare sul mouse il campo di testo per poter scrivere.  
Possibilità di utilizzare più webcam contemporaneamente avviando una sola volta l’applicativo.

Sito:

Il sito si può sicuramente sviluppare più a fondo in futuro. Oltre alle varie migliorie e nuove funzionalità che si potrebbero aggiungere ci sono alcune parti che si potrebbero reworkare per renderle più efficienti ed efficaci. Nella parte dei grafici, in particolare, si potrebbero ricaricare i dati senza dover ricaricare la pagina, vedere i grafici filtrati per luogo e nome, e aggiungere la possibilità di vedere i dati di tutte le webcam per uno stand nello stesso grafico invece di grafici separati.  
Nella pagina degli stand, quando si hanno troppi dati e si cerca di utilizzare la modifica in tempo reale, essa diventa lenta per via di un uso di cicli nella parte di Tabledit per l’identificazione del dato da modificate, che però rallentano il processo. Questo problema è risolvibile assegnando un ID ad ogni riga.

## Considerazioni personali

Nikola:  
Dopo questo progetto credo che io sia pronto ad affrontare altri progetti di maggiore difficoltà. Ho potuto apprendere al meglio, osservando lo svolgimento dei vari compiti delle altre persone del team e capendo come affrontare al meglio l’attività. Inizialmente ho avuto molta difficoltà sia a livello pratico che teorico, perché non sono riuscito a capire come si doveva svolgere il tutto, poi andando avanti ho appreso molto la conoscenza.

Bralla:

Con questo progetto ho imparato a lavorare in un team e a dividere i compiti. Ho imparato anche a combinare PHP e JavaScript per implementarli in un sito che lavora su un database con i dati in continuo aggiornamento e penso di esserci riuscito piuttosto bene nel complesso.

Stefano:  
Questo progetto è stato piuttosto interessante in quanto è il mio primo lungo progetto in team ed è stato una sfida piacevole perché è la prima volta che mi sono occupato per così tanto tempo di una applicazione web. È stato anche interessante vedere le problematiche che possono succedere quando si lavora in un gruppo diviso in più parti che si occupano di differenti cose e quanto la comunicazione sia importante.

Samuel:

Questo lavoro, dal mio punto di vista, è stato istruttivo e interessante. Era la prima volta che utilizzavo il linguaggio Python e l’ho trovato molto comodo per certi versi. Lavorare in gruppo non è scontato, bisogna avere una buona organizzazione e saper suddividersi i compiti in modo equo. Nonostante ciò non ci sono stati grossi problemi, se non qualche discussione necessaria ad arrivare alle soluzioni dei problemi. Non amo particolarmente lavorare in team poiché ho i miei tempi e le mie abitudini organizzative, ma è stata comunque un’esperienza abbastanza piacevole. Anche grazie al carattere e alla motivazione dei miei compagni.

# Bibliografia

## Sitografia

* https://www.python.org/downloads/windows, *Python Releases for Windows*, 21-01-2021.
* <https://pypi.org/project/Pillow/2.2.1/>, Pillow PyPI, 11-04-2021

# Allegati

Elenco degli allegati:

* Diari di lavoro
* Codici sorgente
* Manuale per utenti di utilizzo
* Prodotto