Università degli studi di Napoli Parthenope

Green Pass

Documentazione progetto di
Reti dei Calcolatori

A.A. 2022/2023

Candidata:
Ruocco Noemi - mat. 0124002445

Sommario

1 - Descrizione del progetto

Di seguito sono riportate le informazioni necessarie a comprendere il funzionamento dell'intero progetto.

1.1 - Introduzione

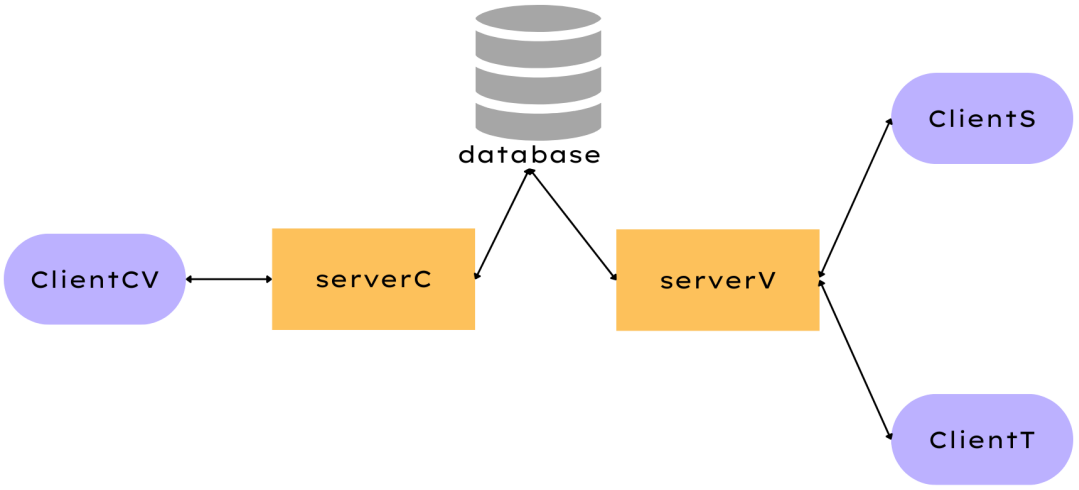
Il progetto *Green Pass* si occupa di gestire, come suggerito dal nome, i green pass e le loro validità.
Studiando il caso reale di come vengono emessi e gestiti i green pass, le specifiche trovate per l'implementazione sono le seguenti:

- Il centro vaccinale, tramite un client, collegandosi al server che si occupa dell'inserimento dei nuovi vaccinati nel database, fornisce informazioni su tessera sanitaria, tempo di validità del green pass e validità di quest'ultimo a seguito della vaccinazione della persona associata a quel codice di tessera sanitaria.

- Un utente, come ad esempio può essere un negoziante, tramite un diverso client si connetterà ad un server di controllo della validità dei green pass per richiedere la verifica dello stato di validità di un green pass inviando il codice della tessera sanitaria al server.

- Un altro utente, presumibilmente un addetto della ASL, tramite un ulteriore client può connettersi allo stesso server di validità citato nel punto precedente per comunicare un contagio, o la guarigione, di una persona, inviando al server il codice della tessera sanitaria e il tipo di modifica da effettuare.

1.2 - Descrizione e schema dell’architettura



Struttura architettura

Come mostra la figura, nell’implementare l’architettura si è cercato di essere il più fedeli possibile alle specifiche trovate dallo studio del problema.

L'architettura del sistema è basata su un modello client-server, in cui ci sono diversi componenti che comunicano tra loro attraverso connessioni socket.
I principali componenti dell'architettura includono:

- Client Centro Vaccinale (clientCV): Questo client rappresenta il centro vaccinale e si occupa dell'invio delle informazioni relative ai nuovi green pass al server delle vaccinazioni (serverC).

- Server Centro Vaccinazioni (serverC): Questo server è essenzialmente il mezzo con cui il centro vaccinale può inserire informazioni sui nuovi green pass emessi, inviandole appunto al serverC tramite il clientCV, e il server poi le salverà all’interno del database.

- Server Controllo Validità Green Pass (serverV): Questo server è il fulcro del sistema e gestisce le richieste di verifica e modifica dello stato dei green pass da parte dei clientS e clientT. Inoltre, comunica con il database per modificare e recuperare le informazioni dei green pass.

- Client per la Verifica della Validità (clientS): Questo client rappresenta gli enti o i soggetti che desiderano verificare la validità di un green pass. Invia richieste di verifica al serverV e riceve lo stato del green pass in risposta.

- Client per la Modifica dello Stato (clientT): Questo client rappresenta gli enti autorizzati a modificare lo stato di validità di un green pass. Invia richieste di modifica al serverV per aggiornare lo stato di un green pass nel database.

- Database SQLite: Il database è utilizzato dal serverV per memorizzare le informazioni relative ai green pass. Contiene i dettagli come il codice tessera, il tempo di validità e lo stato di ogni green pass.

Per quanto concerne il flusso di comunicazione tra client e server:

- Il clientCV si collegano al serverC per inviare i dati relativi ai nuovi green pass emessi dal centro vaccinale. Questi dati vengono poi memorizzati nel database dal serverC.

- I clientS possono connettersi al serverV per verificare la validità di un green pass. Il serverV verifica lo stato nel database e invia la risposta al clientS.

- I clientT possono connettersi al serverV per richiedere la modifica dello stato di un green pass. Il serverV apporta le modifiche necessarie nel database e invia la conferma al clientT.

1.3 - Descrizione del protocollo applicazione

Il protocollo applicativo svolge un ruolo cruciale nell'architettura del sistema di gestione e controllo dei green pass. Definisce le regole e i metodi secondo cui i diversi componenti del sistema comunicano tra loro.
In questa implementazione, il protocollo applicativo stabilisce il modo in cui il client centro vaccinale (clientCV), il server di controllo validità (serverV), i client per la verifica (clientS) e i client per la modifica dello stato (clientT) interagiscono per garantire l'emissione, la verifica e la gestione dei green pass.

Attraverso scambi di messaggi, il protocollo permette la trasmissione sicura dei dati tra i componenti, facilitando operazioni come l'invio delle informazioni sui nuovi green pass, la verifica della validità dei green pass e la notifica di cambiamenti di stato.

Di seguito sono mostrati nel dettaglio i passaggi e i messaggi che costituiscono il protocollo applicativo all'interno di questa architettura.

Client centro vaccinale (clientCV)

I. Crea una connessione con il server vaccinale (serverV) utilizzando una socket.
II. Si richiede all'utente se desidera continuare o terminare la connessione.
III. Se la scelta è "continua":
    A. Si richiedono le informazioni relative al green pass da inserire (tessera sanitaria, tempo di validità del green pass, e stato della validità del green pass).
    B. Si verifica il formato del codice inserito per la tessera sanitaria, tramite la funzione verify_code, e solo se è stato inserito correttamente si prosegue.
    C. I dati inseriti vengono inviati al serverV.
    D. Si richiede all'utente se vuole proseguire o terminare la connessione.
IV. Se la scelta è "fine" la connessione con il server viene direttamente chiusa.
V. Viene chiusa la connessione clientCV.

Server centro vaccinale (serverC)

I. Inizializza il server creando una socket.
II. Inizia ad essere in ascolto di eventuali connessioni.

III. Quando un clientCV si connette accetta la connessione e gestisce le sue richiestre tramite la funzione handle_client.
IV. Riceve il messaggio iniziale dal client per decidere come gestire la connessione.
V. Se la richiesta è “continua”:
    A. Riceve i dati del green pass dal client.
    B. Salva i dati nel database tramite la funzione save_to_database.
    C. Richiede al clientCV come proseguire.
VI. Se il messaggio è “fine”, la connessione con il clientCV viene direttamente chiusa.
VII. Il server viene chiuso alla fine di operazioni.

Client verifica dei green pass (clientS)

I. Crea una connessione con il serverV utilizzando una socket.
II. Invia al server un messaggio con il proprio tipo di client così da garantire la giusta gestione.
III. Si richiede all’utente come desidera procedere.
IV. Se si riceve un “continua”:
    A. Si richiede il codice della tessera sanitaria che si vuole verificare.
    B. Si verifica il formato fornito per la tessera sanitaria tramite verify_code.
    C. Il codice viene inviato al server.
    D. Riceve la risposta sulla validità corrispondente da parte del server.
    E. Si richiede all’utente come proseguire.
V. Se la scelta è “fine” la connessione con il server viene terminata.
VI. Viene chiusa la connessione clientS.

Client notifica modifica del green pass (clientT)

I. Si crea una connessione con il serverV tramite una socket.
II. Invia anch’esso un messaggio al server specificando il proprio tipo di client per poter essere gestito correttamente.
III. Si richiede all’utente come desidera procedere.
IV. Se il messaggio è “continua”:
    A. Si richiede il codice della tessera sanitaria e il tipo di modifica da effettuare.
    B. Verifica il formato inserito per la tessera sanitaria con verify_code.

C. Se il formato è corretto invia i dati al server.
D. Richiede all'utente la prossima azione.

V. Se il messaggio ricevuto è "fine", la connessione con il server viene chiusa.
VI. Termina la connessione clientT.

Server controllo green pass (serverV)

I. Viene inizializzato il serverV tramite socket.
II. Si mette in ascolto per connessioni in arrivo.
III. Quando un client si connette:
    A. Accetta la connessione.
    B. Verifica di che tipo di client si tratta.
    C. In base al tipo di client utilizza la funzione handle_clientS o la funzione handle_clientT.
    D. Alla fine delle operazioni chiude il server.

2 - Dettagli implementativi dei client

Di seguito sono riportati i dettagli implementativi, sia per quanto riguarda i client che per ciò che concerne i server.

2.1 - Dettagli implementativi clientCV

Il clientCV rappresenta il centro vaccinale, si occupa di inviare informazioni sui nuovi green pass rilasciati al server dei vaccini (serverC) per farli inserire nel database.
Di seguito sono riportati tutti i dettagli con relativo codice.

Per poter eseguire correttamente il codice è importante importare le librerie necessarie.
La libreria socket, per usufruire delle socket per le connessioni, e la libreria re, necessaria per la funzione di controllo del codice di tessera sanitaria inserito.

```
5   import socket
6   import re
```

Le prime azioni che compie la funzione principaledi controllo del clientCV sono quelle necessarie all'inizializzazione del client e alla connessione verso il serverC.

```
8   def clientCV_program():
9       # poiché sia server che client sono sullo stesso pc
10      host = socket.gethostname()
11      # si assegna alla porta lo stesso numero di quello assegnato al serverV
12      port = 5000
13
14      # inizializzazione della socket clientCV
15      clientCV_socket = socket.socket()
16      # connessione al serverV
17      clientCV_socket.connect((host, port))
18      print("      --------------------")
19      print("Connessione al server effettuata.")
20      print("      --------------------")
```

Successivamente si richiede di inserire un input iniziale, necessario a capire se si vuole proseguire con le operazioni o si desidera chiudere la connessione.

```
22      # per poter effettuare operazioni
23      print("Se si desidera terminare la connessione scrivere fine,\naltrimenti scrivere continua")
24      message = input(" -> ")
```

Seguono controlli sul messaggio inserito.
Il primo controllo è per il caso in cui il messaggio inserito sia “continua”. A questo punto vengono inviate al serverC le informazioni necessarie all’inserimento del green pass nel database.

```
    # se il messaggio iniziale è "continua", verrà richiesto l'inserimento
    # dei dati del green pass
    if message.lower() == "continua":
        tessera_sanitaria = input("Inserisci il codice tessera sanitaria: ")
        tempo_val = input("Inserisci il tempo di validità del green pass: ")
        stato = input("Inserisci lo stato: ")
```

Con a seguire i relativi controlli per i dati inseriti, e un controllo finale sul messaggio inserito inizialmente.

```
        # controllo iniziale sul formato della tessera sanitaria (ultime 8 cifre)
        if verify_code(tessera_sanitaria):
            # controllo poiché l'unico stato ammissibile post vaccinazione è "valido"
            if stato.lower() == "valido":
                #inviare i dati inseriti al serverV
                data = f"{tessera_sanitaria},{tempo_val},{stato}"
                mess1 = f"{message}"
                clientCV_socket.send(mess1.encode())
                clientCV_socket.send(data.encode())
            else:
                print("Non è stato inserito uno stato valido. \n")
        else:
            print("Il formato inserito non è corretto. Inserire gli ultimi 8 numeri della tessera. \n")
```

Successivamente si richiede all’utente come vuole proseguire, se la scelta sarà “continua” allora inserirà altre informazioni, se invece la scelta è “fine” allora si proseguirà con il resto del programma.

```
            # per scegliere come proseguire
            print("      --------------------")
            print("Scegliere prossima azione:")
            message2 = input(" -> ")
            mess2 = f"{message2}"
            clientCV_socket.send(mess2.encode())
```

Ci sono inoltre altri due controlli sul messaggio iniziale.
Se il messaggio iniziale è “fine”, si richiede la chiusura della connessione e si va avanti verso la fine del programma.

```
    # se il messaggio iniziale è "fine", la connessione viene subito chiusa
    if message.lower() == "fine":
        #chiudere la connessione
        clientCV_socket.send("fine".encode())
        print("      --------------------")
        print("Chiusura della connessione avviata.")
```

Se invece il messaggio iniziale non è “continua”, ma nemmeno “fine”, viene comunicato l’errore nell’inserimento del messaggio.

```
62      # ulteriore controllo sul messaggio iniziale
63      if message.lower() != "continua" and message.lower() != "fine":
64          print("Non è stato inserito un messaggio ammissibile per compiere operazioni.\n")
```

Infine, il programma termina con la chiusura della connessione.

```
67      # per chiudere la connessione col serverV alla fine delle operazioni
68      print("      --------------------")
69      print("La connessione verrà chiusa.")
70      clientCV_socket.close()
```

Il file contiene inoltre il codice della funzione di verifica per la formattazione del codice della tessera sanitaria, e la specifica necessaria sul metodo di esecuzione.

```
73  # funzione per verificare il corretto formato della tessera sanitaria
74  def verify_code(tessera_sanitaria):
75      # definizione del pattern regex per una tessera sanitaria
76      # (contando le ultime 8 cifre)
77      pattern = r'^\d{8}$'
78
79      # si effettua la verifica usando il pattern regex
80      if re.match(pattern, tessera_sanitaria):
81          return True
82      else:
83          return False
84
85
86  # per specificare il metodo di esecuzione
87  if __name__ == '__main__':
88      clientCV_program()
```

2.2 - Dettagli implementativi clientS

Il clientS rappresenta il tramite con cui un qualsiasi utente, ad esempio un negoziante, può richiedere al serverV la verifica di validità di un green pass.

Anche in questo caso le librerie da includere sono la socket e la re.

Il codice di questo client risulta molto simile al codice del clientCV, ma in questo caso il clientS invia solo il codice della tessera sanitaria per cui verificare il green pass associato, e aspetterà la risposta da parte del serverV.

Il codice risulta quindi simile a quello del clientCV fino alla parte di codice da eseguire successivamente all'invio del messaggio "continua".
Con l'aggiunta di una variabile client (in questo caso posta come clients), da inviare al server per permettergli di identificare la tipologia di client con cui ha a che fare e agire di conseguenza.

```python
def clientS_program():
    #poiché sia server che client sono sullo stesso pc
    host = socket.gethostname()
    #si assegna alla porta lo stesso numero di quello assegnato al serverC
    port = 5050

    #inizializzazione
    clientS_socket = socket.socket()
    #connessione al serverC
    clientS_socket.connect((host, port))
    #per poter specificare al server il tipo di client che si sta connettendo
    client = "clients"
    clientS_socket.send(client.encode())
    print("      --------------------")
    print("Connessione al server effettuata.")
    print("      --------------------")

    #per poter effettuare operazioni
    print("Per terminare la connessione scrivere fine, \naltrimenti scrivere continua.")
    message = input(" -> ")
```

Il codice interno dell'if nel caso in cui il messaggio sia "continua" varia leggermente, ma la parte finale sulla prossima scelta e il controllo è uguale al clientCV.

```python
    #se il primo messaggio inserito è "continua" si prosegue con la richiesta di validità
    if message.lower() == "continua":
        tessera_sanitaria = input("Inserisci il codice tessera sanitaria: ")
        # controllo sul corretto inserimento del codice della tessera sanitaria (ultime 8 cifre)
        if verify_code(tessera_sanitaria):
            # per poter inviare i dati al serverC
            data = f"{tessera_sanitaria}"
            mess1 = f"{message}"

            clientS_socket.send(mess1.encode())
            clientS_socket.send(data.encode())

            # per ottenere risposta dal serverC
            print("      --------------------")
            server_response = clientS_socket.recv(1024).decode()
            print("Risposta a seguito del controllo:", server_response)

            # per scegliere come proseguire
            print("      --------------------")
            print("Scegliere prossima azione:")
            message2 = input(" -> ")
            mess2 = f"{message2}"
            clientS_socket.send(mess2.encode())
        else:
            print("Il formato inserito per la tessera sanitaria non è corretto. Inserire le utlime 8 cifre del codice.
```

Il codice continua con la chiusura della connessione nel caso in cui il messaggio sia “fine” e un ulteriore nel caso il messaggio non sia nessuno dei due consentiti.

```
    # se il messaggio iniziale è "fine" la connessione al serverC verrà chiusa
    if message.lower() == "fine":
        #chiudere la connessione
        clientS_socket.send("fine".encode())
        print("      --------------------")
        print("Chiusura della connessione avviata.")
        clientS_socket.close()

    # ulteriore controllo sul messaggio iniziale
    if message.lower() != "continua" and message.lower() != "fine":
        print("Non è stato inserito un messaggio ammissibile per compiere operazioni.\n")
```

Infine c’è la chiusura del client nel caso in cui non sia entrato in uno dei due if.

```
    # per chiudere la connessione col serverC alla fine delle operazioni
    print("      --------------------")
    print("Chiusura della connessione avviata.")
    clientS_socket.close()
```

Anche qui è presente la funzione di controllo sulla forma di codice tessera sanitaria inserito e sulla modalità di esecuzione.

```
# funzione per verificare il corretto formato della tessera sanitaria
def verify_code(tessera_sanitaria):
    # definizione del pattern regex per una tessera sanitaria
    # (contando le ultime 8 cifre)
    pattern = r'^\d{8}$'

    # si effettua la verifica usando il pattern regex
    if re.match(pattern, tessera_sanitaria):
        return True
    else:
        return False


# per specificare il metodo di esecuzione
if __name__ == '__main__':
    clientS_program()
```

2.3 - Dettagli implementativi clientT

Il clientT rappresenta qualsiasi ente che ha il permesso di modificare la validità di un green pass a seguito di un contagio o di una guarigione da Covid-19.

Anche questo necessità di libreria socket e della re per poter essere eseguito correttamente, e l'inizializzazione è implementata sempre allo stesso modo, fino alla richiesta di inserimento del primo messaggio per scegliere come proseguire.
In questo caso il client invierà un informazione (client = "clientt") per permettere al server di identificare il tipo di client che ha instaurato la connessione.

```python
import socket
import re

def clientT_program():
    #poiché sia server che client sono sullo stesso pc
    host = socket.gethostname()
    #si assegna alla porta lo stesso numero di quello assegnato al server
    port = 5050

    #inizializzazione
    clientT_socket = socket.socket()
    #connessione al server
    clientT_socket.connect((host, port))
    # per poter specificare al serverc il tipo di client che si sta connettendo
    client = "clientt"
    clientT_socket.send(client.encode())
    print("      --------------------")
    print("Connessione al server effettuata.")
    print("      --------------------")

    # per poter effettuare operazioni
    print("Per terminare la connessione scrivere fine, \naltrimenti scrivere continua.")
    message = input(" -> ")
```

In questo caso, se si sceglie di proseguire, oltre alla tessera sanitaria viene richiesto anche il tipo di modifica da dover effettuare, ovvero se si tratta di un contagio o di una guarigione, per tanto ci sono tutti i controlli necessari per l'inserimento.

```python
    if message.lower() == "continua":
        tessera_sanitaria = input("Inserisci il codice tessera sanitaria: ")
        modifica = input("Inserisci 'contagio' oppure 'guarigione' per le modifiche allo stato: ")

        # primo controllo sul codice della tessera sanitaria (ultime 8 cifre)
        if verify_code(tessera_sanitaria):
            # controllo sull'inserimento della modifica
            if modifica.lower() == "contagio" or modifica.lower() == "guarigione":
                # invia i dati al server
                data = f"{tessera_sanitaria},{modifica}"
                mess1 = f"{message}"

                clientT_socket.send(mess1.encode())
                clientT_socket.send(data.encode())
            else:
                print("Non è stato inserito un valore corretto per la modifica del green pass.\n")
```

Successivamente il codice risulta simile a quello dei client precedenti, con la richiesta di come proseguire con le operazioni, i controlli sul messaggio iniziale inserito e la chiusura della connessione una volta terminate le operazioni.

```
    # se il messaggio iniziale è "fine" la connessione al serverC verrà chiusa
    if message.lower() == "fine":
        #chiudere la connessione
        clientT_socket.send("fine".encode())
        print("      --------------------")
        print("Chiusura della connessione avviata.")
        clientT_socket.close()

    #ulteriore controllo sul messaggio iniziale
    if message.lower() != "continua" and message.lower() != "fine":
        print("Non è stato inserito un messaggio ammissibile per compiere operazioni.\n")


    # per chiudere la connessione alla fine delle operazioni
    print("      --------------------")
    print("Chiusura della connessione avviata.")
    clientT_socket.close()
```

Alla fine del file si trovano la funzione per la verifica del codice inserito di tessera sanitaria e per il metodo di esecuzione.

```
# funzione per verificare il corretto formato della tessera sanitaria
def verify_code(tessera_sanitaria):
    # definizione del pattern regex per una tessera sanitaria
    # (contando le ultime 8 cifre)
    pattern = r'^\d{8}$'

    # si effettua la verifica usando il pattern regex
    if re.match(pattern, tessera_sanitaria):
        return True
    else:
        return False


# per specificare il metodo di esecuzione
if __name__ == '__main__':
    clientT_program()
```

3 - Dettagli implementativi dei server

Di seguito sono mostrate nel dettaglio le implementazioni del server centro vaccinale (serverC) e del server di controllo validità dei green pass (serverV).

3.1 - Dettagli implementativi serverC

Il serverC rappresenta il server centro vaccinazioni, a cui vengono fornite tutte le informazioni sulle vaccinazioni effettuate e quindi sui green pass emessi, che dovrà poi andare a salvare all'interno del database dei green pass.

In questo caso, le librerie da includere sono la socket, che è comune a tutti i client e i server, e la libreria sqlite3, necessaria per utilizzare il database.

```
4  import socket
5  import sqlite3
```

Successivamente si trova il programma principale per il funzionamento del server.
Come per i client, le prime operazioni servono per inizializzare il server e per configurare l'ascolto.

```
8  def serverC_program():
9      # per ottenere il nome dell'host
10     host = socket.gethostname()
11     # inizializzare la porta ad un numero superiore a 1024 poiché le precedenti sono riservate ad altro
12     port = 5000
13
14     # per ottenere un istanza
15     serverC_socket = socket.socket()
16     # per legare l'indirizzo dell'host alla porta
17     serverC_socket.bind((host, port))
18
19     # configurare il numero di client che il server può ascoltare simultaneamente
20     serverC_socket.listen(2)
21     print("      --------------------")
22     print("Server centro vaccinazioni in ascolto.")
```

A seguire è stata utilizzata una variabile per verificare lo stato del server, fin quando la variabile running resta True, il server è attivo.

All'interno del while c'è il richiamo alla funzione necessaria per gestire la richiesta del client.

```
24      # variabile per il controllo dello stato del server
25      running = True
26      while running:
27          # per accettare una nuova connessione
28          clientCV_socket, address = serverC_socket.accept()
29          print("Connessione da parte di: " + str(address))
30          print("      --------------------")
31          # per gestire la connessione da parte dei client
32          running = handle_client(clientCV_socket)
```

Alla fine del programma principale si trova il codice necessario alla chiusura delle connessioni e alla chiusura del server.

```
34      #per chiudere le connessioni
35      print("Il programma client è stato chiuso.\n")
36      clientCV_socket.close()
37
38      # per chiudere il server alla fine delle operazioni
39      print("Il server verrà chiuso.")
40      print("      --------------------")
41      serverC_socket.close()
```

Nota: La scelta di chiudere il server, e in particolare di far collegare la chiusura della connessione richiesta dal client alla chiusura anche del server è stata eseguita per una semplice questione di comodità durante la verifica del funzionamento. Il codice rispetta il funzionamento per cui è stato implementato anche in caso di non chiusura del server.

La seconda funzione presente nel file è quella di gestione del client, in questo caso solo il clientCV comunica con il serverC per cui è l'unica funzione di gestione del client presente.
Si richiede inizialmente il messaggio iniziale inserito dal client, nel caso in cui sia "continua", le informazioni vengono prese e salvate all'interno del database tramite l'apposita funzione (non sono necessari controlli poiché sono già presenti nel codice del client).
Successivamente si riceve il secondo messaggio inserito dal client per verificare la prossima azione da eseguire.

Se invece il messaggio non è valido si prosegue direttamente con la chiusura del server, ponendo running come False.

```
45  def handle_client(clientCV_socket):
46      # ricevi le informazioni (pacchetti più grandi di 1024 bytes non verranno accettati)
47      message = clientCV_socket.recv(1024).decode()
48      mess1 = message
49
50      # se il client ha inviato come messaggio iniziale "continua",
51      # allora si posso salvare nel database i dati inviati
52      if mess1.lower() == "continua":
53          # recuperare i dati dal client
54          data = clientCV_socket.recv(1024).decode()
55          tessera_sanitaria, tempo_val, stato = data.split(',')
56          # salvare i dati nel database
57          print("Dati ricevuti: \n-> codice tessera {}, \n-> tempo validità {}, \n-> stato {}".format(tessera_sanitaria,
58          save_to_database(tessera_sanitaria, tempo_val, stato)
59          # per sapere come proseguire dopo il primo invio di dati
60          message2 = clientCV_socket.recv(1024).decode()
61          mess2 = message2
62      else:
63          print("Non è stato ricevuto un messaggio coerente.\n")
64          return False
```

Se invece il primo messaggio è “fine”, o il secondo messaggio è “”fine”, allora si prosegue con la chiusura della connessione e la variabile running diventa False, per cui anche il server viene chiuso.

```
66      # per uscire dal ciclo di gestione dei client e finire l'operazione
67      if mess1.lower() == "fine" or mess2.lower() == "fine":
68          print("      --------------------")
69          print("Chiusura della connessione richiesta dal client.")
70          clientCV_socket.close()
71          return False
72
73      return True
```

Sono presenti poi due funzioni relative al database, la prima per salvare le informazioni ricevute e creare quindi un nuovo record, la seconda invece per mostrare il contenuto del database.

La funzione di salvataggio delle informazioni nel database.

```python
# funzione per poter salvare i dati nel database
def save_to_database(tessera_sanitaria, tempo_val, stato):
    # per creare o connettersi al database
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()

    # per creare la tabella nel caso non esista
    cursor.execute('''
                   CREATE TABLE IF NOT EXISTS greenpass (
                   id INTEGER PRIMARY KEY,
                   tessera_sanitaria TEXT,
                   tempo_val TEXT,
                   stato TEXT)
                   ''')

    # per inserire i dati nella tabella
    cursor.execute('''
                   INSERT INTO greenpass (tessera_sanitaria, tempo_val, stato)
                   VALUES (?, ?, ?)''',
                   (tessera_sanitaria, tempo_val, stato,))

    # per salvare le modifiche
    conn.commit()
    # per chiudere la connessione al database
    conn.close()
    print("I dati sono stati inseriti correttamente nel database.")
```

La funzione per mostrare il contenuto del database.

```python
# funzione per visualizzare il contenuto del database
def display_database_contents():
    # per connettersi o creare il database
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()

    # per selezionare tutto il contenuto del database
    cursor.execute('SELECT * FROM greenpass')
    rows = cursor.fetchall()
    # per stampare a video il contenuto
    print("\n*****************************")
    print("Contenuto del database:")
    for row in rows:
        print("ID: ", row[0])
        print("Codice tessera: ", row[1])
        print("Tempo validità del green pass: ", row[2])
        print("Stato green pass: ", row[3])
        print("---------------------------")

    # per chiudere la connessione al database
    conn.close()
```

Infine viene specificato il metodo di esecuzione, ovvero che in primis viene eseguito il codice relativo alla funzione del server, poi una volta chiuso viene mostrato il contenuto del database.

```
# per specificare il metodo di esecuzione
if __name__ == '__main__':
    # prima viene eseguito il programma relativo al server
    serverC_program()
    # quando il server viene chiuso, viene visualizzato il contenuto del database
    display_database_contents()
```

3.2 - Dettagli implementativi serverV

Il serverV rappresenta il server di controllo della validità dei vaccini, che è in ascolto delle richieste da parte dei clientS e dei clientT.

Per tanto, la sua implementazione risulta simile a quella del serverC fino alla parte in cui si mette in ascolto dei client.

```
import socket
import sqlite3

# funzione principale del serverV
def serverV_program():
    # per ottenere il nome dell'host
    host = socket.gethostname()
    # inizializzare la porta ad un numero superiore a 1024 (le precedenti sono riservate)
    port = 5050

    # per ottenere un istanza
    serverV_socket = socket.socket()
    # per legare l'indirizzo dell'host alla porta
    serverV_socket.bind((host, port))

    # configurare il numero di client che il server può ascoltare simultaneamente
    serverV_socket.listen(2)
    print("      --------------------")
    print("Server controllo validità green pass in ascolto.")
```

Anche in questo caso la variabile running serve a capire lo stato del server, poiché il controllo sulla connessione è stato implementato allo stesso modo di come è stato implementato nell'altro server.
Fin quando la variabile running resta True il server rimane attivo, quando, a seguito delle operazioni scelte dal client, questa viene posta a False il server viene chiuso insieme alla connessione.

All'interno del while, poiché i client che deve servire sono di due tipologie diverse, si accetta inizialmente una richiesta generica di connessione.

```
    running = True
    # per poter gestire le connessioni in modo corretto in base al tipo di client
    while running:
        # per accettare una nuova connessione dal client (ancora generico)
        client_socket, address = serverV_socket.accept()
        print("Connessione da parte di: " + str(address))
        print("      --------------------")
```

Successivamente, per capire in che modo gestire il client, analizza l'informazione client che gli è stata inviata, a seconda che sia "clients" o "clientt" deciderà quale funzione di gestione dei client utilizzare.

```
        client = client_socket.recv(1024).decode()
        if client.lower() == "clients":
            #gestire la connessionedel clientB
            print("La connessione è stata effettuata da un clientB.\n")
            running = handle_clientS(client_socket)
        else:
            if client.lower() == "clientt":
                #gestire la connessione da parte dei client B2
                print("La connessione è stata effettuata da un clientB2.\n")
                running = handle_clientT(client_socket)
            else:
                print("Tipo do client sconosciuto, la connessione verrà chiusa. \n")
                client_socket.close()
                continue
```

Seguono le righe dedicate alla chiusura della connessione e del server, che sono la fine della funzione principale del server.

```
    # a seguito della fine delle operazioni da parte del client
    print("Il programma client è stato chiuso.\n")
    client_socket.close()

    #termina il programma chiudendo la socket del server
    print("Il server verrà chiuso.")
    print("      --------------------")
    serverV_socket.close()
```

La prima funzione che compare successivamente è quella di gestione del clientS, ovvero il client che richiede il controllo della validità del green pass. Per effettuare il controllo, il server acquisisce il codice della tessera sanitaria che gli è stato inviato, richiama la funzione per verificare se sia o meno presente nel database dei green pass, e restituisce la validità al client.

Successivamente controlla il secondo messaggio inviato per capire come continuare l'esecuzione.

```
def handle_clientS(clientS_socket):
    # per essere sicuri che si stia utilizzando il corretto handle
    print("Si sta utilizando la funzione per la gestione del clientS che ha richiesto la verifica di validità. \n")
    # ricevi le informazioni (pacchetti più grandi di 1024 bytes non verranno accettati)
    message = clientS_socket.recv(1024).decode()
    mess1 = message

    # controllo sul messaggio ricevuto
    if mess1.lower() == "continua":
        # ricevi dati dal client
        data = clientS_socket.recv(1024).decode()
        tessera_sanitaria = data
        print("Dati ricevuti: \n-> codice tessera {}".format(tessera_sanitaria))

        # richiama il controllo su quella tessera sanitaria
        stato = check_greenpass(tessera_sanitaria)
        # invia la risposta al client che ha effettuato la richiesta
        clientS_socket.send(stato.encode())

        # per sapere come proseguire
        message2 = clientS_socket.recv(1024).decode()
        mess2 = message2
    else:
        print("Non è stato ricevuto un messaggio corretto.\n")
        return False
```

L'ultima parte è dedicata alla chiusura della connessione e del server, in quanto alla fine delle operazioni la variabile running viene posta a False.

```
    #per uscire dal ciclo di gestione dei client
    if mess1.lower() == "fine" or mess2.lower() == "fine":
        print("      --------------------")
        print("Chiusura della connessione richiesta dal client.")
        clientS_socket.close()
        return False

    return False
```

Per quanto riguarda la funzione per gestire i clientT, ovvero i client che notificano il tipo di modifica da effettuare alla validità del green pass, la parte iniziale è simile alla funzione di gestione del clientS, in cui si prende il messaggio iniziale inviato dal client.

```
def handle_clientT(clientT_socket):
    # per essere ricuri dell'handle che si utilizza
    print("Si sta utilizando la funzione per la gestione dei clientT, che ha inviato una modifica allo stato del green
    #ricevi le informazioni (pacchetti più grandi di 1024 bytes non verranno accettati)
    message = clientT_socket.recv(1024).decode()
    mess1 = message
```

Una volta preso il messaggio, si controlla il tipo di modifica da effettuare e si richiama in entrambi i casi la funzione di modifica dei dati nel database (la scelta su quale operazione effettuare verrà presa da questa funzione).
Successivamente il secondo messaggio dirà come proseguire l'esecuzione.

Anche in questo caso ci sono i controlli necessari e nel caso in cui non è stato inserito un messaggio coerente con gli input possibili si prosegue con il resto del codice tornando alla funzione principale del server.

```
    # controllo sul messaggio ricevuto
    if mess1.lower() == "continua":
        # ricevi dati dal client
        data = clientT_socket.recv(1024).decode()
        tessera_sanitaria, modifica = data.split(',')
        print("Dati ricevuti: \n-> codice tessera {}, \n-> modifica {}".format(tessera_sanitaria, modifica))
        # per capire il tipo di modifica da effettuare
        if modifica.lower() == "contagio" or modifica.lower() == "guarigione":
            modify_greenpass(tessera_sanitaria, modifica)
        else:
            # ulteriore controllo
            if modifica.lower() != "contagio" and modifica.lower() != "guarigione":
                print("Le uniche parole accettabili per lo stato sono 'contagio' oppure 'guarigione'. \n")

        # per sapere come proseguire
        message2 = clientT_socket.recv(1024).decode()
        mess2 = message2
    else:
        print("Non è stato consegnato un messaggio coerente.\n")
        return False
```

Infine si trova l'ultimo controllo sul messaggio inviato e la chiusura della connessione, che porta lo stato del server ad essere False e quindi a chiudersi.

```
    # per uscire dal ciclo di gestione dei client
    if mess1.lower() == "fine" or mess2.lower() == "fine":
        print("      --------------------")
        print("Chiusura della connessione richiesta dal client.")
        clientT_socket.close()
        return False

    return False
```

Ci sono poi le tre funzioni relative al database.

La prima è la funzione che, preso il codice di tessera sanitaria, verifica la validità controllando in primis se è presente all'interno del database e poi, in caso sia presente, controllando la variabile dello stato del green pass, inviando poi la risposta al programma di gestione del clientS che la invierà come risposta alla richiesta del client.

```python
# funzione per verificare la validità del green pass
def check_greenpass(tessera_sanitaria):
    # per creare o connettersi al database
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()

    # per verificare se il green pass è presente nel database
    tessera = tessera_sanitaria
    print("Si verificherà la validità del green pass appartenente alla tessera sanitaria {}\n".format(tessera))
    cursor.execute("SELECT stato FROM greenpass WHERE tessera_sanitaria=?", (tessera,))
    result = cursor.fetchone()
    # per mandare il risultato al client
    if result:
        return result[0]
    else:
        return "Non presente nel database"

    # per chiudere la connessione al database
    conn.close()
```

La seconda funzione è quella che si occupa di modificare lo stato dei green pass.
Presi in input il codice della tessera sanitaria e il tipo di modifica da effettuare, controllerà il tipo di modifica da effettuare e sostituirà lo stato in "non valido" se si è stati contagiati, o in "valido" se si tratta di guarigione.

```python
# funzione per modificare lo stato di validità di un green pass
def modify_greenpass(tessera_sanitaria, modifica):
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()
    tessera = tessera_sanitaria

    # per scegliere l'azione in base alla modifica richiesta
    if modifica.lower() == "contagio":
        nuovo_stato = "non valido"
        cursor.execute("UPDATE greenpass SET stato = ? WHERE tessera_sanitaria = ?", (nuovo_stato, tessera,))
        conn.commit()
    else:
        if modifica.lower() == "guarigione":
            nuovo_stato = "valido"
            cursor.execute("UPDATE greenpass SET stato = ? WHERE tessera_sanitaria = ?", (nuovo_stato, tessera,))
            conn.commit()
        else:
            if modifica.lower() != "contagio" and modifica.lower() != "guarigione":
                print("Non è stata inserita una modifica di stato accettabile. \n")

    # per chiudere la connessione
    conn.close()
```

Infine la funzione che permette di mostrare il contenuto del database.

```python
# funzione per mostrare il contenuto del database
def display_database_contents():
    # connessione al database
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()

    # selezionare tutto il contenuto
    cursor.execute('SELECT * FROM greenpass')
    rows = cursor.fetchall()
    # stampare il contenuto
    print("\n*****************************")
    print("Contenuto del database:")
    for row in rows:
        print("ID: ", row[0])
        print("Codice tessera: ", row[1])
        print("Tempo validità greenpass: ", row[2])
        print("Stato del greenpass: ", row[3])
        print("---------------------------")

    # chiudere la connessione al database
    conn.close()
```

Alla fine del file si trova il metodo di esecuzione. Prima viene eseguito il codice principale del server, quando il server viene chiuso, quindi alla fine di tutte le operazioni, viene eseguita la stampa del contenuto del database.

```python
# per specificare il metodo di esecuzione
if __name__ == '__main__':
    # prima viene eseguito il programma principale del server
    serverV_program()
    # alla fine del programma principale viene mostrato il contenuto del database
    display_database_contents()
```

4 - Manuale utente

In questa sezione vengono illustrate le istruzioni per eseguire il codice e alcuni esempi di come dovrebbe essere visualizzato l'output per diverse operazioni.

4.1 - Istruzioni per l'esecuzione

Poiché Python è un linguaggio interpretato, non ha bisogno dell'operazione di compilazione prima di essere eseguito, per cui basterà il solo comando di compilazione.

Per poter eseguire il programma bisognerà, in primis, aver installato sul proprio computer il linguaggio Python, più precisamente Python3, per poter eseguire il codice proposto senza rischiare di avere errori di interpretazione.

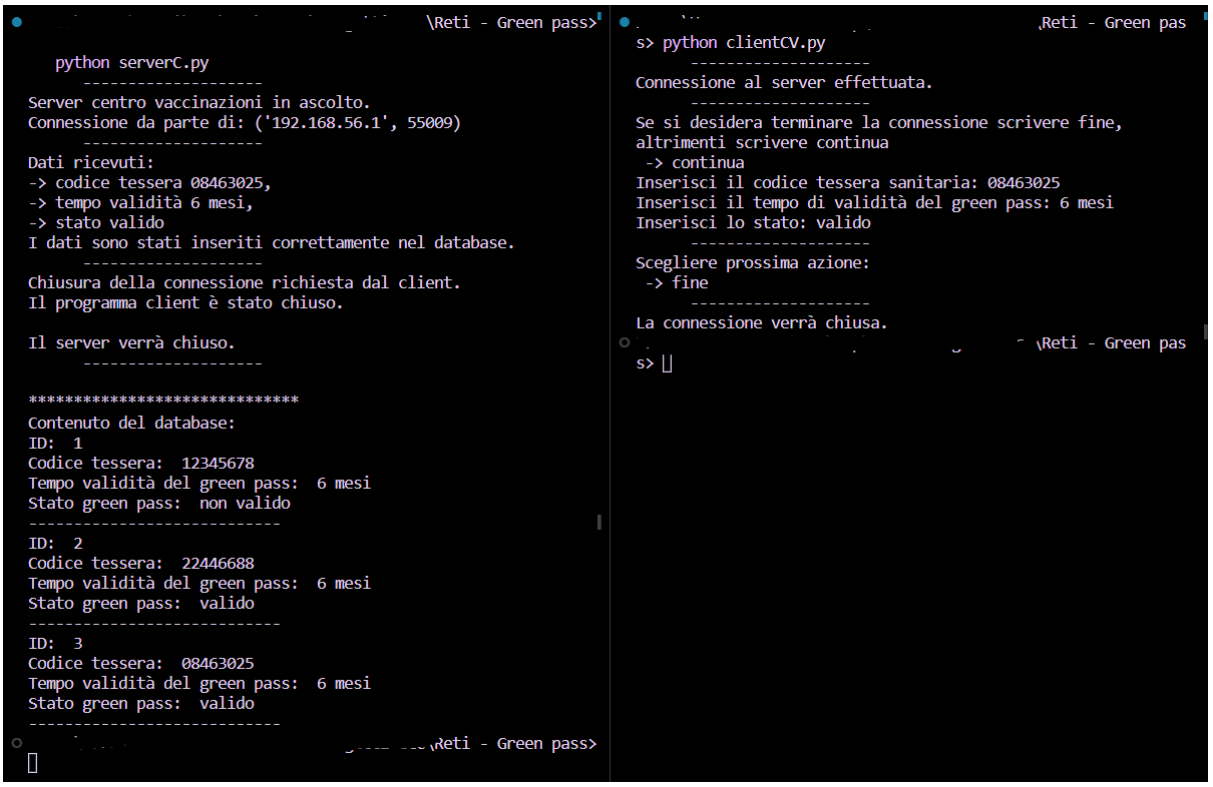
Una volta verificata la presenza di Python nel proprio computer, basterà inserire il prompt per l'esecuzione, una volta essere passati nella directory della cartella contenente tutti i file del progetto.

Nel caso di Python, il prompt è semplicemente: *python* (oppure python3) *nomefile.py*

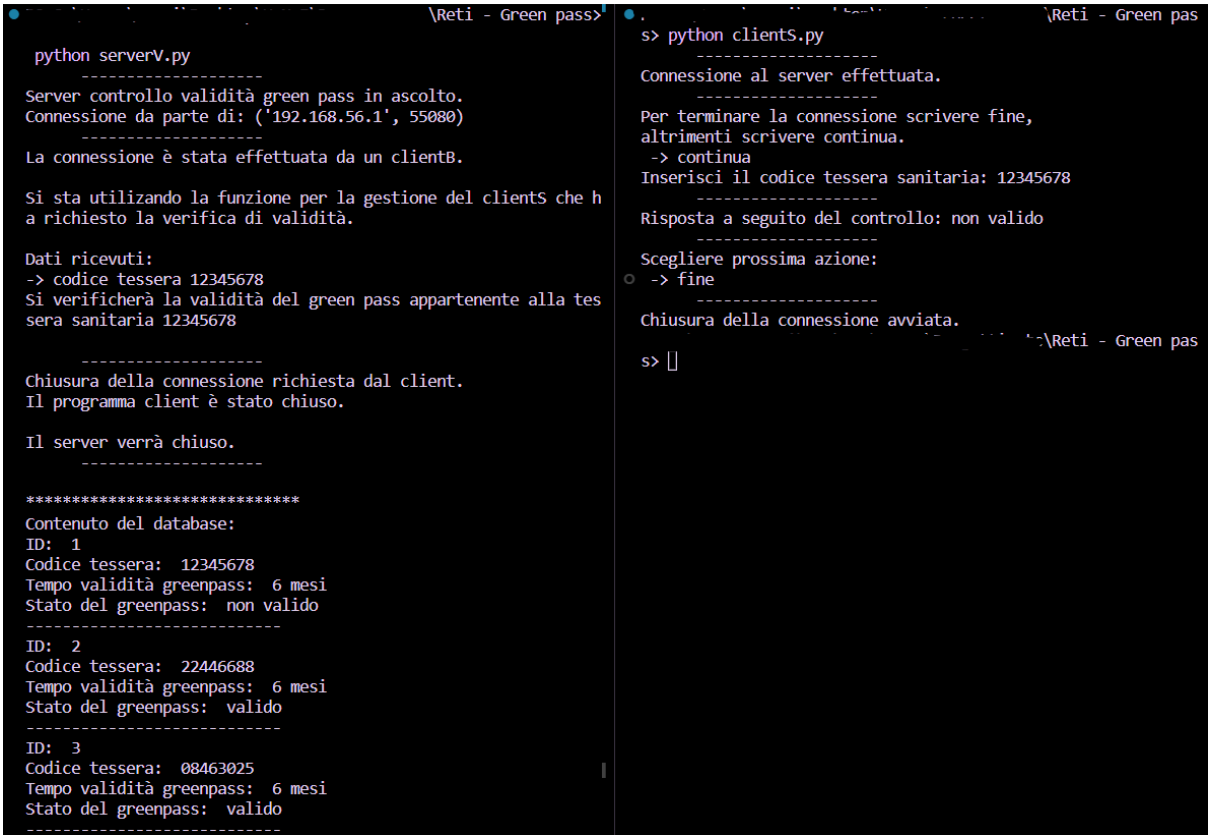
Nota: é importante che siano eseguiti prima i server e poi i client, altrimenti ci saranno errori di connessione poiché i client non hanno nulla a cui connettersi.

4.2 - Esempi di output

Di seguito è mostrato un esempio di output lato serverC e lato ClientCV nel caso in cui il client richiede l'inserimento di un nuovo green pass all'interno del database.



Questo è invece un esempio di come dovrebbe vedersi l'output nel caso in cui un clientS richiede la verifica di validità di un green pass al serverV.



Infine l'output nel caso in cui un clientT comunica la modifica di un green pass a seguito di guarigione al serverV.