

# Università degli Studi di Napoli 'Parthenope'

Dipartimento di Scienze e Tecnologie



Relazione del progetto di Reti di Calcolatori

## Gestione dei Green Pass

Github

Proponenti:  
**Giuseppe Fiorillo**  
**Raffaele Ventriglia**

Professori:  
**Aniello Castiglione**  
**Alessio Ferone**

Matricole:  
**0124002248**  
**0124002249**

Data di consegna:  
**24/02/2023**

**Anno Accademico 2022/2023**



# Indice

<b>1</b>	<b>Descrizione del progetto</b>	<b>5</b>
1.1	Schemi dell'architettura . . . . .	5
1.2	Schemi del protocollo applicazione . . . . .	6
<b>2</b>	<b>Implementazione</b>	<b>8</b>
2.1	Dettagli implementativi del client . . . . .	8
2.1.1	client . . . . .	8
2.1.2	clientT . . . . .	8
2.1.3	clientS . . . . .	8
2.1.4	GreenPass . . . . .	9
2.2	Dettagli implementativi del server . . . . .	9
2.2.1	centro_vaccinale . . . . .	9
2.2.2	serverV . . . . .	9
2.2.3	serverG . . . . .	10
2.2.4	Gestione delle connessioni e delle interruzioni . . . . .	10
<b>3</b>	<b>Manuale utente</b>	<b>11</b>
3.1	Istruzioni per la compilazione . . . . .	11
3.2	Istruzioni per l'esecuzione . . . . .	12
<b>4</b>	<b>Simulazione dell'applicazione</b>	<b>13</b>
4.1	Inserimento . . . . .	13
4.2	Controllo di validità . . . . .	13
4.3	Validazione . . . . .	14

# Elenco delle figure

1.1	Schema dell'architettura del progetto . . . . .	6
4.1	Inserimento di un green pass tramite CLI . . . . .	13
4.2	Controllo di validità di un green pass tramite CLI . . . . .	13
4.3	Invalidazione di un green pass tramite CLI . . . . .	14

# Traccia

Progettare ed implementare un servizio di gestione dei **green pass** secondo le seguenti specifiche. Un utente, una volta effettuata la vaccinazione, tramite un **client** si collega ad un centro vaccinale e comunica il codice della propria tessera sanitaria. Il centro vaccinale comunica al **ServerV** il codice ricevuto dal **client** ed il periodo di validità del green pass. Un **ClientS**, per verificare se un green pass è valido, invia il codice di una tessera sanitaria al **ServerG** il quale richiede al **ServerV** il controllo della validità. Un **ClientT**, inoltre, può invalidare o ripristinare la validità di un green pass comunicando al **ServerG** il contagio o la guarigione di una persona attraverso il codice della tessera sanitaria.

# Capitolo 1

## Descrizione del progetto

Il progetto è stato interamente scritto in C, usando delle socket per gestire la comunicazione tra thread. Il progetto è stato realizzato usando sistemi Unix-like, in particolare MacOS e Linux Ubuntu, mentre l'IDE utilizzato è CLion.

### 1.1 Schemi dell'architettura

- **client**: usato dall'utente per collegarsi al centro vaccinale al fine di comunicare il codice della propria tessera sanitaria.
- **centro\_vaccinale**: comunica al **serverV** il codice della tessera sanitaria ricevuta dal **client** insieme al suo periodo di validità.
- **serverV**: riceve il green pass dal centro vaccinale, potendo inoltre entrare in contatto con **serverG** per eseguire le operazioni di validazione e di controllo dei green pass.
- **serverG**: fa da tramite tra il **serverV** e i client T ed S.
- **clientT**: comunicando al **serverG** un'eventuale guarigione o contagio, può invalidare o ripristinare la validità di un green pass.
- **clientS**: può richiedere al **serverG** un controllo sulla validità di un green pass.

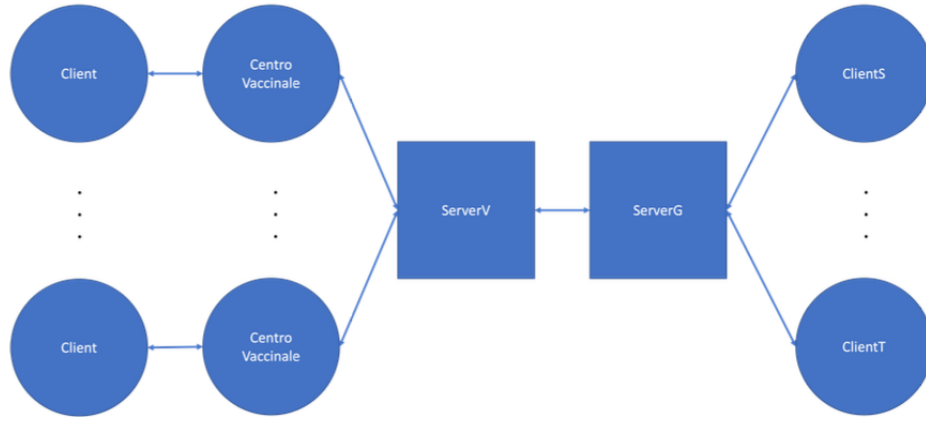


Figura 1.1: Schema dell'architettura del progetto

## 1.2 Schemi del protocollo applicazione

Il progetto è stato sviluppato utilizzando il protocollo **TCP** (Transmission Control Protocol), che è un protocollo del livello trasporto, ed è anche il protocollo più utilizzato nelle reti dei calcolatori dato l'elevato numero di vantaggi che offre nella comunicazione delle reti.

Uno dei primi e più importanti vantaggi è l'affidabilità che questo protocollo offre, in quanto vengono utilizzati numeri di sequenza e di riconoscimento che assicurano che tutti i dati siano ricevuti nell'ordine corretto e senza alcuna perdita. Un ulteriore vantaggio è dato dal fatto di essere un protocollo connection - oriented, ovvero viene stabilita una connessione logica tra due endpoint prima della trasmissione dei dati: ciò consente di trasferire in maniera efficiente i dati, in quanto entrambi gli endpoint possono ottimizzare e gestire il flusso dei dati. Come ultimo vantaggio che ci ha consentito di scegliere, e quindi utilizzare questo tipo di protocollo, è dato dal fatto che fornisce meccanismi di controllo della congestione che prevengono la congestione e garantiscono una equa condivisione delle risorse di rete.

I server vengono attivati all'indirizzo localhost, ovvero 127.0.0.1, mentre le porte utilizzate sono:

- CENTER\_PORT 8888
- SERVERV\_PORT 8890

- SERVERG\_PORT 8100



## Capitolo 2

# Implementazione

### 2.1 Dettagli implementativi del client

#### 2.1.1 client

Il `client` ha il compito di avviare la procedura di registrazione di un nuovo GreenPass, inviando il codice fiscale della tessera al centro vaccinale; il codice fiscale viene richiesto come secondo parametro quando si vuole avviare il `client`. Controlla se il codice fiscale inserito sia della lunghezza desiderata, ovvero 16 caratteri, altrimenti l'esecuzione viene terminata.

#### 2.1.2 clientT

Il `clientT` ha il compito di validare o invalidare il green pass desiderato, e lo fa inviando il codice fiscale al `serverV`; l'intera procedura va ovviamente avviata all'avvenimento di un contagio oppure di una guarigione. Il `clientT`, dopo aver inviato il codice fiscale, riceve una risposta da parte del `serverG` che gli indica se l'operazione è avvenuta con successo o meno in base al valore della variabile `response`.

#### 2.1.3 clientS

Il `clientS` ha il compito di verificare in che stato si trova un green pass, e lo fa inviando il codice fiscale della tessera sanitaria al `serverV`. La procedura di controllo della validità del green pass avviene in maniera simile a quello che accade nel `clientT`, ovvero il `clientS`, dopo aver inviato il codice fiscale al `serverV`, riceve una risposta da esso che gli indica se il green pass è valido,

non valido, scaduto oppure non presente all'interno del file che contiene tutti i green pass.

#### 2.1.4 GreenPass

I vari client utilizzano una struct chiamata **GreenPass**, che contiene 4 campi:

- **tessera\_sanitaria**, che è un'array di caratteri di lunghezza **TESSERA\_LENGTH** (16) con un carattere aggiuntivo che rappresenta il terminatore.
- **valid\_from** e **valid\_until** di tipo **time\_t** che indicano la data di inizio della validità e quella di scadenza del green pass preso in considerazione.
- **service** di tipo **int** che indica il tipo di servizio che un client richiede sulla struct presa in considerazione.

```
struct GreenPass {  
    char tessera_sanitaria[TESSERA_LENGTH + 1];  
    time_t valid_from;  
    time_t valid_until;  
    int service;  
};
```

## 2.2 Dettagli implementativi del server

### 2.2.1 centro\_vaccinale

Il **centro\_vaccinale** ha il compito di servire da collegamento tra il **client** e il **serverV** per quanto riguarda l'operazione di inserimento di un nuovo green pass. Dopo aver ricevuto il codice fiscale collegato al green pass, viene instaurata una connessione con il **serverV** per richiedere il servizio che permette di registrare il nuovo green pass.

### 2.2.2 serverV

Il **serverV** ha un compito molto importante all'interno dell'intero progetto, in quanto può essere considerato come il punto cardine della comunicazione dei vari servizi richiesti dai client, utilizzando gli altri due server **centro\_vaccinale** e il **serverG** come punto di appoggio e canale di comunicazione per far arrivare le varie risposte ai client. Il **serverV** riceve dai due server una struct green pass contenente i dati su cui devono essere effettuate

le operazioni richieste dal campo `service` contenuto all'interno della stessa struct.

### 2.2.3 serverG

Il `serverG` ha il compito di servire da collegamento tra `clientT`, `clientS` e il `serverV` per le procedure di controllo e validazione/invalidazione di un green pass.

### 2.2.4 Gestione delle connessioni e delle interruzioni

Per quanto riguarda la gestione delle connessioni in entrata, all'avvio di ogni server viene inizializzato un pool di thread, e ogni volta che un client effettua una richiesta ad uno dei server, viene prima di tutto cercato un thread libero all'interno del pool, che si occuperà di gestire il client. Ogni thread è programmato per effettuare un'uscita controllata nel caso in cui avvenissero degli errori nell'utilizzo delle socket: vengono chiuse tutte le connessioni, viene sbloccato il mutex e viene chiamata la funzione `pthread_exit()`.

Per la gestione delle interruzioni si è reso necessario creare un handler che permetta, in caso di un segnale `SIGINT` catturato dal programma, di impostare a 0 il valore della variabile `isRunning` che viene utilizzata all'interno di un ciclo while infinito che permette di effettuare tutte le operazioni dei vari server. Facendo in questo modo, abbiamo la possibilità di chiudere tutte le socket che erano state aperte precedentemente e sbloccare il mutex che è stato usato per l'accesso in mutua esclusione al file che contiene tutti i green pass registrati.

## Capitolo 3

# Manuale utente

### 3.1 Istruzioni per la compilazione

Per la **compilazione** dei file, i comandi da eseguire sono:

```
gcc client.c -o client
gcc centro_vaccinale.c -o centro_vaccinale
gcc serverV.c -o serverV
gcc serverG.c -o serverG
gcc clientT.c -o clientT
gcc clientS.c -o clientS
```

Per facilitare la compilazione, tuttavia, ci siamo avvalsi di un makefile, così costituito:

```
# compilatore utilizzato
CC = gcc
# abilita tutti i warning del compilatore
CFLAGS = -g -Wall

all : clean client clientT clientS centro_vaccinale serverV serverG

client : client.c green_pass.h addresses.h
        $(CC) $(CFLAGS) client.c -o client

clientT : clientT.c green_pass.h addresses.h
        $(CC) $(CFLAGS) clientT.c -o clientT

clientS : clientS.c green_pass.h addresses.h
```

```
$(CC) $(CFLAGS) clientS.c -o clientS

centro_vaccinale : centro_vaccinale.c green_pass.h addresses.h
$(CC) $(CFLAGS) centro_vaccinale.c -o centro_vaccinale

serverV : serverV.c green_pass.h addresses.h
$(CC) $(CFLAGS) serverV.c -o serverV

serverG : serverG.c green_pass.h addresses.h
$(CC) $(CFLAGS) serverG.c -o serverG

clean :
    rm -f *.o all
    rm -f clientT
    rm -f clientS
    rm -f client
    rm -f centro_vaccinale
    rm -f serverG
    rm -f serverV
```

Grazie a questo makefile, sarà sufficiente digitare `make` per poter compilare tutti i file correttamente.

## 3.2 Istruzioni per l'esecuzione

Per poter eseguire i file senza troppi problemi è raccomandato innanzitutto avviare i server, in quest'ordine:

```
./serverV
./serverG
./centro_vaccinale
```

Dopodiché, possiamo avviare i singoli client. Notare che `client` necessita l'avvio pregresso di `serverV` e `centro_vaccinale`, mentre gli altri due client richiedono che siano prima stati avviati `serverV` e `serverG`.

```
./client <codice_fiscale>
./clientT <codice_fiscale>
./clientS <codice_fiscale>
```

## Capitolo 4

# Simulazione dell'applicazione

### 4.1 Inserimento

Come spiegato prima, usando la sintassi `./client <codice_fiscale>` possiamo richiedere l'inserimento di un green pass.

```
giuseppefiorillo@MacBook-Pro-di-Giuseppe Progetto-Reti-dei-calcolatori % ./client RSSMRA80A01H501U
Green pass valido
```

Figura 4.1: Inserimento di un green pass tramite CLI

Causando la scrittura nel file `green_pass.txt` della seguente stringa:  
`RSSMRA80A01H501U : dd/mm/yyyy : dd/mm/yyyy : 1`

### 4.2 Controllo di validità

Usando la sintassi `./clientS <codice_fiscale>` invece possiamo richiedere il controllo di validità su un green pass inserito.

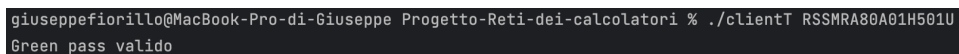
```
giuseppefiorillo@MacBook-Pro-di-Giuseppe Progetto-Reti-dei-calcolatori % ./clientS RSSMRA80A01H501U
1
Green pass valido
```

Figura 4.2: Controllo di validità di un green pass tramite CLI

Controllando la validità del green pass inserito in precedenza, notiamo che esso è ancora valido.

### 4.3 Validazione

Usando la sintassi `./clientT <codice_fiscale>` possiamo richiedere la validazione/invalidazione di un green pass inserito.

A terminal window with a dark background. The prompt is 'giuseppesforillo@MacBook-Pro-di-Giuseppe Progetto-Reti-dei-calcolatori %'. The command entered is './clientT RSSMRA80A01H501U'. The output is 'Green pass valido' on the next line.

```
giuseppesforillo@MacBook-Pro-di-Giuseppe Progetto-Reti-dei-calcolatori % ./clientT RSSMRA80A01H501U
Green pass valido
```

Figura 4.3: Invalidazione di un green pass tramite CLI

In questo caso, essendo il green pass inserito valido, stiamo andando ad annullarlo. Nella riga in cui è presente il green pass nel file `green_pass.txt`, troveremo adesso lo 0 come ultimo carattere invece dell'1.