

Università degli studi di Napoli  
Parthenope

**Reti Dei Calcolatori**

**Green Pass**



D'ANGELO SIMONE 0124/002601

VETRANO ALESSIO 0124/002326

GALLUCCIO LUIGI 0124/002293

## Sommario

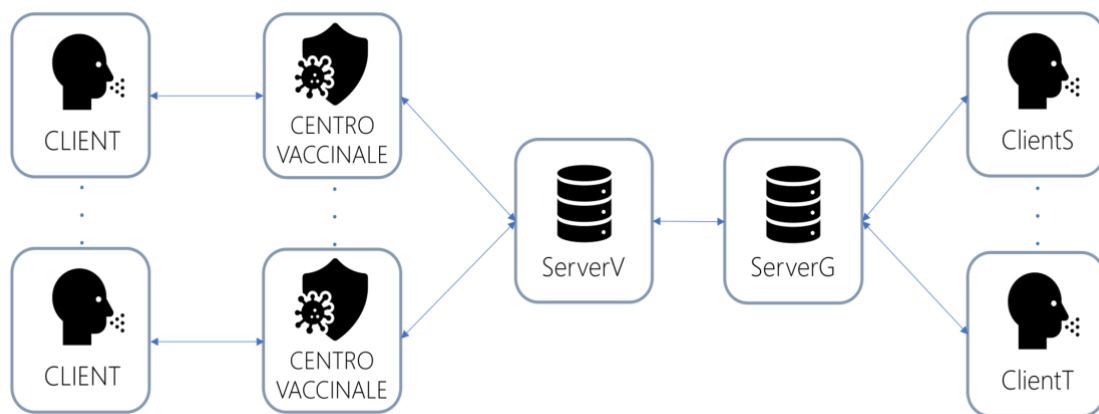
1.Descrizione del progetto .....	2
2. Descrizione e schemi dell'architettura.....	2
3.Descrizione e schemi del protocollo applicazione .....	3
4.Dettagli implementativi del client.....	3
4.1 Utente.....	3
4.2 Centro Vaccinale.....	4
4.3 App Verifica .....	4
4.4 ASL .....	4
5.Dettagli implementativi del server.....	5
5.1 Server Vaccinale .....	5
5.2 Server Verifica .....	5
6.Manuale utente.....	6
6.1 Istruzioni per la compilazione .....	6
6.2 Istruzioni per l'esecuzione.....	6
7. Simulazione dell'applicazione .....	7

## 1.Descrizione del progetto

La traccia del progetto ha lo scopo di progettare e implementare un servizio di gestione dei Green Pass secondo determinate specifiche. L'utente che usa il servizio, una volta effettuata la vaccinazione, si collega al centro vaccinale a cui comunica il codice della propria tessera sanitaria, il centro provvederà a fornire il cliente di un green pass che può essere validato o meno. Il cliente, inoltre, può invalidare o ripristinare la validità del Green Pass comunicando un eventuale contagio o guarigione.

Il progetto in questione è scritto in linguaggio C, utilizzando i socket per la comunicazione tra i processi. La realizzazione del progetto è avvenuta utilizzando esclusivamente piattaforme UNIX, nello specifico Arch Linux e MacOS.

## 2. Descrizione e schemi dell'architettura



- **CLIENT** (*utente.out*) → L'utente, una volta effettuata una vaccinazione, tramite un client si collega ad un centro vaccinale a cui comunica il codice della propria tessera sanitaria.
- **CENTRO VACCINALE** (*centro\_vaccinale.out*) → Comunica al ServerV il codice ricevuto dal client ed il periodo di validità del Green Pass.
- **ClientS** (*app\_verifica.out*) → Verificare se il Green Pass è effettivamente valido, invia il codice della tessera sanitaria da controllare al ServerG, il quale richiede al ServerV il controllo della validità del Green Pass.
- **ClientT** (*asl.out*) → Può invalidare o ripristinare la validità del **Green Pass** comunicando al ServerG un eventuale contagio o un eventuale guarigione identificandola attraverso il codice della tessera sanitaria.
- **ServerV** (*serverV.out*) → Modifica lo stato di un **Green Pass** secondo le direttive dell'ClientT e inoltre, si occupa dell'invio del **Green Pass** al ServerG.
- **ServerG** (*serverG.out*) → Si occupa di fare da tramite tra il Server Vaccinale e l'ASL in modo da applicare le direttive di quest'ultimo. Successivamente comunica con ClientS per comunicargli l'esito della validità di un determinato **Green Pass**.

Tutti i file del nostro progetto presentano un fondamento in comune caratterizzato dalla presenza dei file `wrapper.c` e `wrapper.h`. Tali file conterranno tutte le funzioni utilizzate per la connessione socket tra i vari server e client e per l'invio e la ricezione dei vari dati tra quest'ultimi.

La scelta di utilizzare funzioni wrapper ha lo scopo di poter gestire opportunamente gli errori per ogni funzione utilizzata, risparmiando anche linee di codice.

### 3.Descrizione e schemi del protocollo applicazione

Gli schemi del protocollo applicazione in C che utilizzano i socket sono regole e convenzioni per la comunicazione tra processi o applicazioni attraverso una rete informatica. I socket in C forniscono un'interfaccia per la comunicazione tra processi e possono essere utilizzati per implementare diversi schemi di protocollo applicazione.

Il tipo di comunicazione adottato è socket con specifica TCP, poiché un protocollo connection oriented, prima di poter trasmettere dati deve stabilire una connessione.

Nello specifico il nostro progetto è basato sulla comunicazione Client-Server basato sulle funzioni `full_write` e `full_read`:

Tali funzioni sono state scelte a differenza degli standard `write` e `read` poiché garantiscono che **tutti** i dati vengano effettivamente scritti e letti sul socket. Queste funzioni continuano a chiamare `write` e `read` finché tutti i dati non sono stati scritti sul socket. Ciò garantisce che tutti i dati vengano inviati al server, anche se ciò richiede più chiamate a `write` e `read`.

Lo schema generale del progetto si basa, come già specificato, su una connessione socket TCP. L'Indirizzo scelto per la gestione della connessione socket è stato "127.0.0.1" comunemente chiamato localhost.

Associate a tal indirizzo abbiamo tre porte che definiscono la connessione:

- **1024** → Per la connessione con *centro\_vaccinale*
- **1025** → Per la connessione con *server\_vaccinale*
- **1026** → Per la connessione con *server\_verifica*

Tali porte ci permettono di utilizzare un indirizzo comune per tutti i file protagonisti della connessione sezionandolo in specifiche porte per specifiche connessioni.

### 4.Dettagli implementativi del client

Nel nostro mini-mondo sono presenti diversi attori che svolgono il ruolo di **Client**.

#### 4.1 Utente

Il Client (*Utente*) si connette al *centro\_vaccinale* ed attende il messaggio di benvenuto dal centro che assicura il corretto collegamento.

Successivamente l'Utente invierà al centro un pacchetto mediante la funzione `create_package()`.

Nello specifico il pacchetto risulterà essere:

```
typedef struct{
    char name[MAX_SIZE];
    char surname[MAX_SIZE];
    char ID[ID_SIZE];
} vax_request;
```

Tale `struct` comprende 3 campi principali dove i primi due sono "char" che rappresentano rispettivamente nome e cognome dell'Utente (dove `MAX_SIZE` è 1024) e come terzo elemento un "char" ID (dove `ID_SIZE` è 21, considerando anche il carattere terminatore).

La scelta di utilizzare l'identificatore di tessera sanitaria e non il semplice codice fiscale è stata presa per evitare omonimi e trattare l'identificatore come chiave univoca.

## 4.2 Centro Vaccinale

Il *centro\_vaccinale* è un Client che ottiene da *Utente* i dati di quest'ultimo. Associati a questi il *centro\_vaccinale* genera un **GreenPass** con una data di inizio e una data di fine validità, la prima definita nel momento stesso in cui il certificato è creato mentre la seconda è data dalla data di inizio con l'aggiunta di nove mesi creando così un periodo di validità di certificazione verde.

Il *centro\_vaccinale* dopo aver creato un **GreenPass** associato ad uno specifico utente lo invia al *server\_Vaccinale*, tutte le azioni gestite dal centro vaccinale sono invocate dalla funzione *answer\_user()*. La struttura del **GreenPass** inviato risulterà essere la seguente:

```
typedef struct{
    char ID[ID_SIZE];
    DATE start_date;
    DATE end_date;
}GreenPass_Date;
```

Utilizziamo l'identificativo della tessera sanitaria (dove `ID_SIZE` è 21, considerando anche il carattere terminatore) come chiave univoca dell'Utente associato, i due elementi di tipo `DATE`, invece, rappresentano l'intervallo della validità del Green Pass come definito.

La struttura `DATE` è data da una `struct` in cui saranno presenti 3 elementi interi che risulteranno essere il giorno, il mese e l'anno.

Le date sono state gestite da due costrutti:

```
time_t time_tick;
time_tick = time(NULL);
```

**Time\_tick** permette di estrapolare il time dal computer.

```
struct tm *t_date = localtime(&time_tick);
```

Struttura che permette di convertire il formato da data a intero, per poi manipolarlo attraverso incrementi dei vari elementi della struttura (`tm_mon`, `tm_year`).

## 4.3 App Verifica

Questo Client è visto come una app per la verifica del Green Pass.

*App Verifica* comunicherà con il *server\_verifica* dal quale riceverà in primis un messaggio di benvenuto, l'utente che utilizza la nostra app inserirà l'identificativo di tessera sanitaria da inviare al *server\_verifica*.

L'applicazione di verifica resterà in attesa che il *server\_verifica* e il *server\_vaccinale* facciano le specifiche operazioni per analizzare il Green Pass associato all'identificativo di tessera inserito e dopo aver ottenuto l'esito il *server\_verifica* comunicherà quest'ultimo all'applicazione.

## 4.4 ASL

ASL rappresenta un ente pubblico generico che ha la possibilità di sospendere o ripristinare un Green Pass a seconda delle necessità.

L'ASL comunicherà l'esito di un tampone al *server\_verifica*, il quale interfacciandosi con *server\_vaccinale* andrà a modificare il package associato all'identificativo della persona che ha effettuato il tampone.

In particolare si andrà a modificare il valore `report` della struct `package` come segue:

```
if(package.report == '1'){
    printf("Invio richiesta di ripristino del Green Pass\n");
}else if(package.report == '0'){
    printf("Invio richiesta di sospensione del Green Pass\n");
}
```

La struct `package` risulterà essere di tipo `Report_Asl` definito nel seguente modo:

```
typedef struct{
    char ID[ID_SIZE];
    char report;
}Report_Asl;
```

Un'associazione tra identificativo di tessera sanitaria e report associato a quest'ultimo.

## 5. Dettagli implementativi del server

### 5.1 Server Vaccinale

Il Server Vaccinale comunica con *centro\_vaccinale* per ricavare il Green Pass e creare un file associato a quest'ultimo.

Il file creato è univoco in relazione al Green Pass e sarà rinominato con il numero di identificativo della tessera sanitaria ricevuta dal *centro\_vaccinale* e contenente data inizio e fine validità del Green Pass.

L'insieme dei file creati consiste nel database del nostro progetto.

Il Server Vaccinale comunica con *server\_verifica* effettuando due operazioni, quest'ultime verranno scelte a seconda di un ulteriore bit di comunicazione inviato da parte del *server\_verifica*.

Le due operazioni sono le seguenti:

- `modify_report`: funzione che modifica il report secondo le direttive dell'ASL
- `send_gp`: funzione che invia un Green Pass richiesto dal *server\_verifica*

Nella comunicazione tra il Server Vaccinale e i rispettivi *server\_verifica* e *centro\_vaccinale* un ruolo fondamentale è quello svolto dal `bit_communication`, dove a seconda del valore verrà gestita una connessione.

In codice:

```
if(bit_communication == '0'){modify_report(connect_fd);
}else if(bit_communication == '1'){send_gp(connect_fd);
}else{printf("bit_communication NOT FOUND\n\n");}
```

### 5.2 Server Verifica

Il Server Verifica comunica con ASL inviandogli un `package` contenente identificativo di tessera associato ad un report. Successivamente il Server Verifica si conatterà al *server\_vaccinale* che effettuerà la modifica effettiva del Green Pass.

Il Server Verifica comunica anche con l'*app\_verifica* dove attenderà la ricezione di un numero di tessera sanitaria associata ad un GP. Dopo aver ottenuto tale identificativo il Server Verifica si collegherà al *server\_vaccinale* inviandogli l'identificativo ricevuto dall'*app\_verifica*.

Il *server\_vaccinale* invierà il Green Pass associato all'identificativo se esiste al Server Verifica che a sua volta comunicherà all'*app\_verifica* se il certificato verde è valido o non esiste.

Nella comunicazione tra il Server Verifica e i rispettivi ASL e *app\_verifica* un ruolo fondamentale è quello svolto dal `bit_communication`, dove a seconda del valore verrà gestita una connessione. In codice:

```
if(bit_communication == '1') receive_report(connect_fd); //connessione con ASL
else if(bit_communication == '0') receive_ID(connect_fd); //connessione con app_verifica
else printf("client non riconosciuto\n");
```

## 6. Manuale utente

### 6.1 Istruzioni per la compilazione

Per la compilazione tramite terminale si dovrebbero compilare i diversi file, nello specifico:

- `gcc -o ServerV ServerV.c`
- `gcc -o CentroVaccinale CentroVaccinale.c`
- `gcc -o ServerG ServerG.c`
- `gcc -o Utente Utente.c`
- `gcc -o ClientS ClientS.c`
- `gcc -o ClientT ClientT.c`

Per facilitare il compito, per quanto concerne l'applicazione è stato creato un *BashScript* di compilazione automatica, in modo da ottenere i file \*.out. Il *BashScript* in questione è "compilazione.sh".

```
#!/bin/bash
mkdir eseguibili
gcc centro_vaccinale/centro_vaccinale.c centro_vaccinale/function_centVacc.c wrapper.c -o eseguibili/centro_vaccinale.out
gcc 'clientS[app_verifica]'/app_verifica.c wrapper.c -o eseguibili/app_verifica.out
gcc 'clientT[asl]'/asl.c wrapper.c -o eseguibili/asl.out
gcc 'serverG[server_verifica]'/server_G.c 'serverG[server_verifica]'/server_verifica.c wrapper.c -o eseguibili/serverG.out
gcc 'serverV[server_vaccinale]'/server_V.c 'serverV[server_vaccinale]'/server_Vaccinale.c wrapper.c -o eseguibili/serverV.out
gcc utente/utente.c wrapper.c -o eseguibili/utente.out
cd eseguibili
chmod +x serverG.out serverV.out centro_vaccinale.out utente.out app_verifica.out asl.out
```

### 6.2 Istruzioni per l'esecuzione


Dopo l'utilizzo del *BashScript* di compilazione, basterà eseguire da terminale i seguenti comandi:

- `./serverG.out`
- `./serverV.out`
- `./centro_vaccinale.out`
- `./app_verifica.out`
- `./asl.out`
- `./utente.out`

Nello specifico, l'ordine raccomandato è quello citato precedentemente.

Per facilitare il compito di esecuzione è stato creato un ulteriore *BashScript* che si occupa di eseguire i file attraverso il pacchetto *xterm*. Nello specifico basterà lanciare il *BashScript* "launcher.sh" per eseguire contemporaneamente tutti i file eseguibili precedentemente creato.

```
#!/bin/bash
xterm -title "serverG" -e bash -c "./eseguibili/serverG.out;exec bash" &
sleep 0.2
xterm -title "serverV" -e bash -c "./eseguibili/serverV.out;exec bash" &
sleep 0.2
xterm -title "centro_vaccinale" -e bash -c "./eseguibili/centro_vaccinale.out;exec bash" &
sleep 0.2
xterm -title "app_verifica" -e bash -c "./eseguibili/app_verifica.out;exec bash" &
sleep 0.2
xterm -title "asl" -e bash -c "./eseguibili/asl.out;exec bash" &
sleep 0.2
xterm -title "utente" -e bash -c "./eseguibili/utente.out;exec bash"
```

 Per gli utilizzatori MacOS è necessario scaricare i seguenti pacchetti: *xterm*, *x11*.

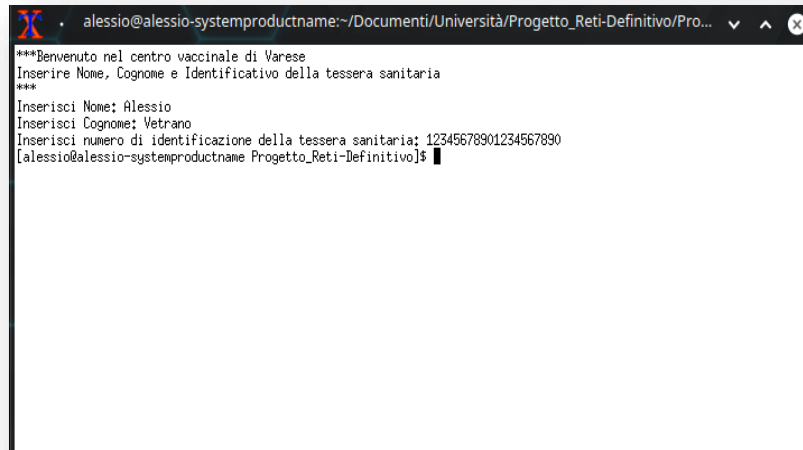
I comandi possono essere installati attraverso i seguenti comandi:

- `brew install xterm`

- `brew install x11`

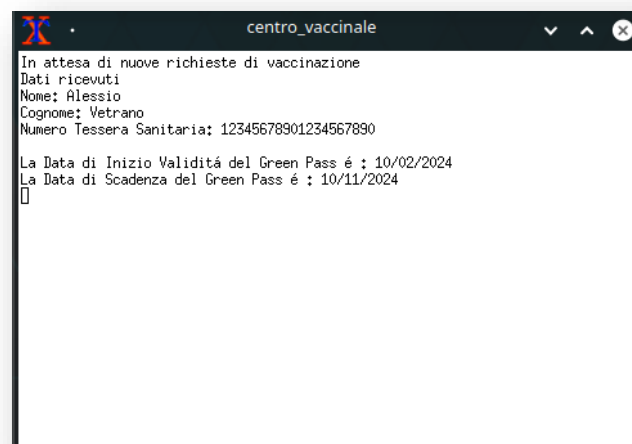
## 7. Simulazione dell'applicazione

Dopo aver eseguito i file, l'utente ha la possibilità di vaccinarsi utilizzando i dati della Tessera Sanitaria.



```
alessio@alessio-systemproductname:~/Documenti/Università/Progetto_Reti-Definitivo/Pro...  
***Benvenuto nel centro vaccinale di Varese  
Inserire Nome, Cognome e Identificativo della tessera sanitaria  
***  
Inserisci Nome: Alessio  
Inserisci Cognome: Vetrano  
Inserisci numero di identificazione della tessera sanitaria: 12345678901234567890  
[alessio@alessio-systemproductname Progetto_Reti-Definitivo]$
```

Di seguito, il centro vaccinale riceve i dati dal cliente appena vaccinato e consegna il Green Pass con la data di inizio e di fine di validità.



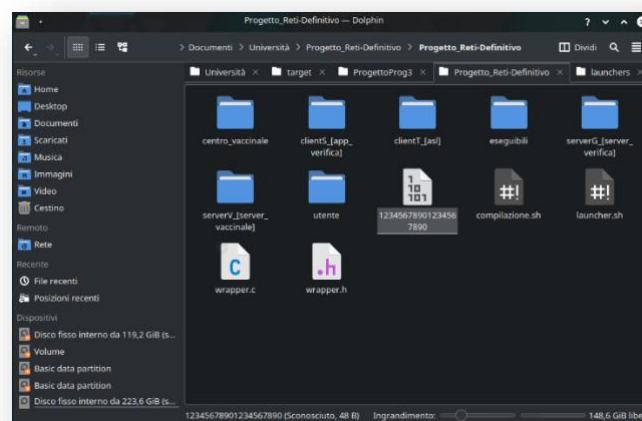
```
centro_vaccinale  
In attesa di nuove richieste di vaccinazione  
Dati ricevuti  
Nome: Alessio  
Cognome: Vetrano  
Numero Tessera Sanitaria: 12345678901234567890  
  
La Data di Inizio Validità del Green Pass é : 10/02/2024  
La Data di Scadenza del Green Pass é : 10/11/2024  
□
```



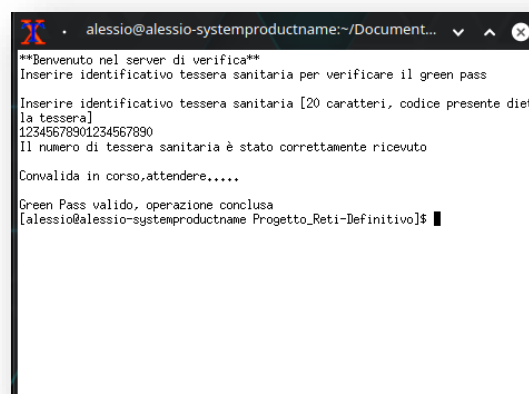


```
serverV
***In attesa di nuove connessioni***
***In attesa di nuove connessioni***
Nome file creato: 12345678901234567890
***In attesa di nuove connessioni***
***In attesa di nuove connessioni***
***In attesa di nuove connessioni***
```

Il serverV, dopo aver ricevuto i dati, crea il file nella directory corrente rinominato con il numero di Tessera Sanitaria.



In questo momento, è possibile verificare la validità del Green Pass, inserendo il numero di Tessera Sanitaria nell'eseguibile app\_verifica. In questo caso poiché il Green Pass è stato appena creato e non modificato risulterà valido.



```
alessio@alessio-systemproductname:~/Document...
**Benvenuto nel server di verifica**
Inserire identificativo tessera sanitaria per verificare il green pass
Inserire identificativo tessera sanitaria [20 caratteri, codice presente dietro la tessera]
12345678901234567890
Il numero di tessera sanitaria è stato correttamente ricevuto
Convalida in corso, attendere.....
Green Pass valido, operazione conclusa
[alessio@alessio-systemproductname Progetto_Reti-Definitivo]$
```

L'eseguibile ASL ha il compito di ripristinare o sospendere un Green Pass tramite l'inserimento della Tessera sanitaria e dell'input riferito all'operazione da effettuare.

```
alessio@alessio-systemproductname:~/Documenti/U...
***ASL***
Inserire identificativo tessera sanitaria ed il referto del tampone per sospensione o ripristinare del green_pass

Inserisci identificativo tessera sanitaria [20 caratteri]
12345678901234567890
Inserire 0 [Green Pass NON valido]
Inserire 1 [Green Pass valido]
INPUT: 0
Invio richiesta di sospensione del Green Pass
***Operazione avvenuta con successo***
[alessio@alessio-systemproductname Progetto_Reti-Definitivo]$
```

Se ricontrollassimo la validità dopo aver sospeso un determinato Green Pass, attraverso l'app\_verifica comunicando con il serverG riceveremo un messaggio in entrambe le console che ci indica la non validità dello stesso.

```
Inserire identificativo tessera sanitaria [20 caratteri, codice presente dietro la tessera]
12345678901234567890
Il numero di tessera sanitaria è stato correttamente ricevuto

Convalida in corso,attendere.....

Green Pass non valido, uscita in corso
[alessio@alessio-systemproductname eseguibili]$
```

```
serverG
In attesa di Green Pass da scansionare
In attesa di Green Pass da scansionare
In attesa di Green Pass da scansionare
Green Pass valido, operazione conclusa

***Operazione avvenuta con successo***
In attesa di Green Pass da scansionare
Green Pass non valido, uscita in corso

[]
```