
# Università degli studi di Napoli Parthenope

Progetto Green Pass

Corso di Reti di Calcolatori e

Laboratorio di Reti di Calcolatori

A.A. 2022/2023

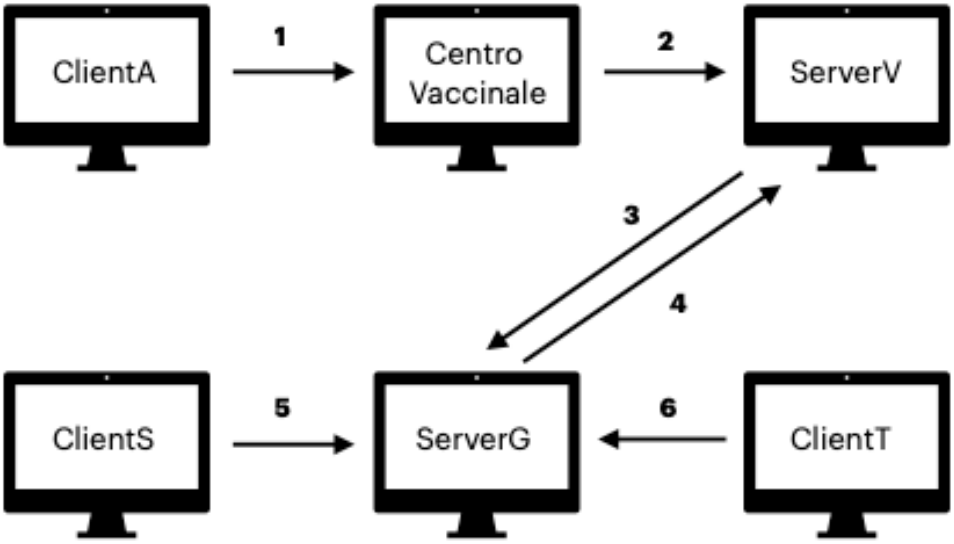
Beniamino Nardone 0124002440
Enrico Madonna 0124002279
Lorenzo Mazza 0124002003

Professore Alessio Ferone - Professore Aniello Castiglione

## Descrizione del progetto

Il progetto ha come obiettivo la creazione di un servizio per la gestione del Green Pass.
Il funzionamento del servizio prevede che un utente, dopo aver effettuato la vaccinazione, utilizzi il ClientA per collegarsi ad un centro vaccinale e comunicare il codice della propria tessera sanitaria.
Il centro vaccinale, a sua volta, comunica al ServerV il codice ricevuto dal client e il periodo di validità del Green Pass. Per verificare la validità di un Green Pass, un ClientS invia il codice di una tessera sanitaria al ServerG, il quale richiede al ServerV il controllo della validità.
Inoltre, il ClientT 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. Infine, il ServerG comunica con il ServerV per invalidare o ripristinare il Green Pass.

## Schema dell’architettura




I numeri (da 1 a 6) non indicano l’ordine delle operazioni. Essi sono puramente indicativi e vengono solo usati per numerare le frecce.

# Schema del protocollo applicazione:

-1: ClientA invia la tessera sanitaria al CentroVaccinale.

-2: CentroVaccinale invia la tessera sanitaria ricevuta dal clientA e periodo validità del Green Pass al ServerV.

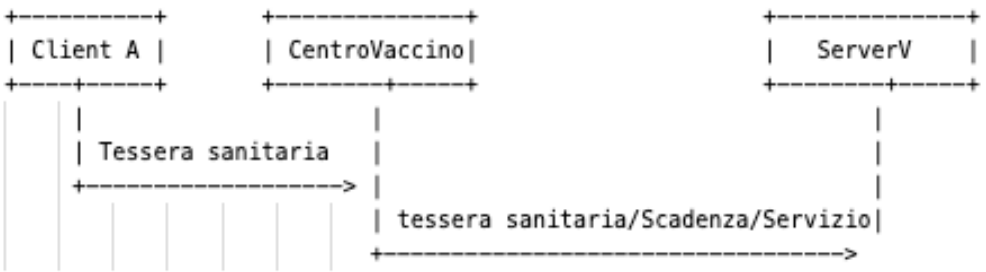
-3: ServerV invia il messaggio “Valido” oppure il messaggio “Non valido” (entrambi riferiti al Green Pass) al ServerG.

-4: ServerG invia il codice della tessera sanitaria (per verificare la validità o per sovrascrivere delle informazioni) al ServerV.

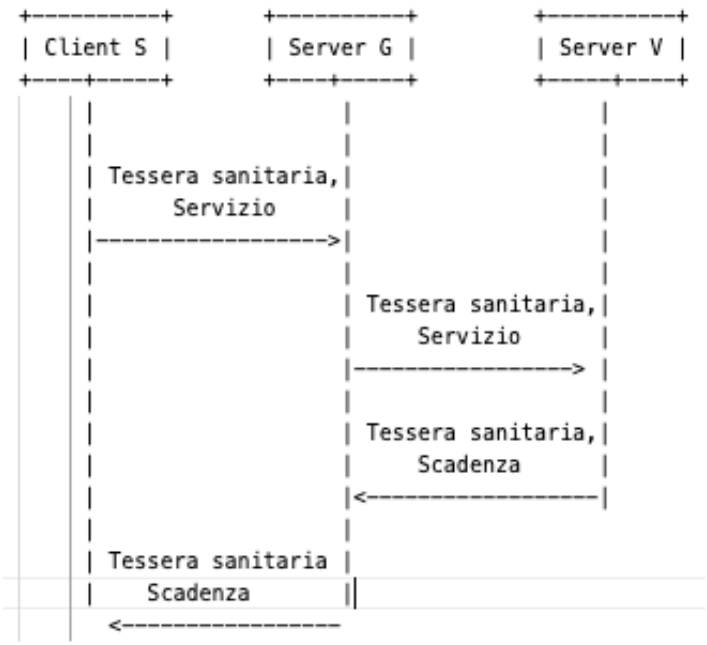
-5: ClientS invia il codice della tessera sanitaria per verificare la validità del Green Pass al ServerG (che ha bisogno del ServerV per ottenere la risposta desiderata: Valido/Non Valido).

-6: ClientT invia una informazione sul Green Pass (per sovrascriverne una già esistente) al ServerG.

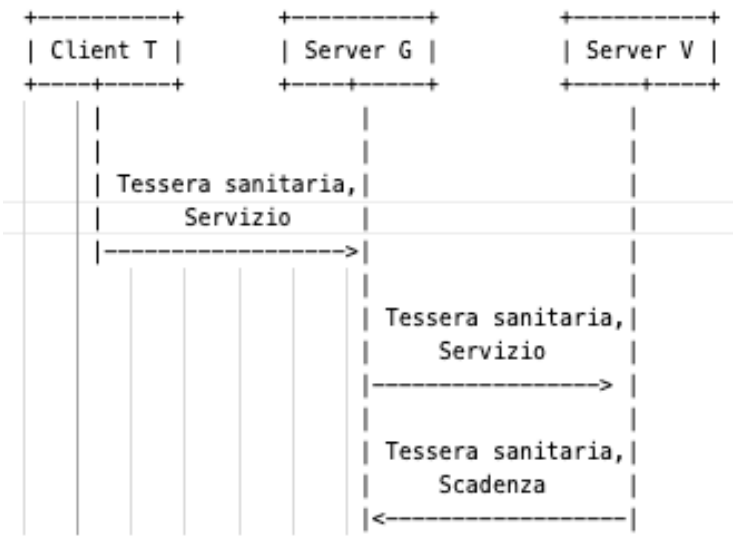
Schema 1:



Schema 2:




Schema 3:



La logica del pacchetto applicazione consiste in tre campi di dati distinti. Il primo campo è una stringa di 20 caratteri che rappresenta il numero della tessera sanitaria. Il secondo campo contiene la data di scadenza del Green Pass associato alla tessera sanitaria. Il terzo campo è il campo "servizio" e contiene un valore che indica il tipo di operazione richiesta al ServerV.

In particolare:

- *Il valore "-1" viene utilizzato per richiedere l'aggiunta di un nuovo Green Pass.*
- *Il valore "0" viene utilizzato per verificare la validità di un Green Pass esistente.*
- *Il valore "1" viene utilizzato per confermare la validità di un Green Pass.*
- *Il valore "2" viene utilizzato per annullare un Green Pass.*

Quando ServerV riceve una richiesta di convalida (codice "1"), utilizza la funzione fseek per spostarsi all'inizio del file contenente i dati del Green Pass e cerca l'elemento corrispondente alla richiesta.

Se il Green Pass viene trovato, ServerV lo rinnova con una nuova data di scadenza. Nel caso in cui il servizio richiesto sia "3", il Green Pass viene rinnovato per 2 giorni, mentre se il servizio è diverso da "3", il Green Pass viene rinnovato per 6 mesi. Se il Green Pass non viene trovato, ServerV aggiunge un nuovo Green Pass con una validità di 2 giorni.

Nel caso in cui ServerV riceva una richiesta di annullamento (codice "2"), cerca il Green Pass corrispondente all'interno del file. Se viene trovato, il Green Pass viene annullato; se non viene trovato, viene aggiunto un nuovo Green Pass scaduto per la tessera sanitaria corrispondente. Questo Green Pass scaduto verrà attivato una volta che la persona sarà guarita.

Per quanto riguarda il ClientT, oltre alla tessera sanitaria, riceve un ulteriore parametro "V" o "I" per scegliere se convalidare o annullare il Green Pass.

## Dettagli implementativi ServerV

La struttura server implementa un server che gestisce la comunicazione tra client e server attraverso una socket TCP.

Il server accetta connessioni da client, legge dati dalla connessione e in base ai dati ricevuti esegue una delle quattro operazioni possibili sui dati di un file contenente informazioni su tessere sanitarie e relative scadenze.

Il codice inizia con l'inclusione di alcune librerie standard per la gestione di socket, file e semafori. In seguito, vengono definite due costanti, una per indicare la scadenza di sei mesi (in secondi) e l'altra per indicare la scadenza di due giorni (in secondi). Viene quindi dichiarata la struttura "green_pass" che rappresenta le informazioni su una tessera sanitaria, tra cui il numero di tessera sanitaria, la data di scadenza e il tipo di servizio.

Dentro la funzione principale, vengono dichiarate diverse variabili, tra cui list_fd e conn_fd che rappresentano rispettivamente il file descriptor della socket del server e del client, serv_add che rappresenta l'indirizzo del server, received e temp che rappresentano la struttura del green pass ricevuto e quella temporanea, pid che rappresenta il processo figlio e zero che viene inizializzato a 0.

Successivamente, viene aperto un file in modalità lettura/scrittura binaria e viene creato un semaforo con nome "semaphore" con permessi di accesso in lettura e scrittura. Il codice crea un socket TCP IPv4 utilizzando la funzione Socket().

Successivamente, utilizzando la struct serv_add, definisce l'indirizzo IP e la porta associata alla socket tramite la funzione Bind(). Infine, con Listen() viene messa in ascolto la socket per accettare eventuali richieste di connessione in entrata.

Questo codice esegue un ciclo continuo di ascolto per nuove connessioni in ingresso sul socket list_fd. Quando una nuova connessione viene accettata, il processo figlio viene creato tramite la funzione fork(). Se la creazione del processo figlio fallisce, viene stampato un messaggio di errore e il programma termina.

Se invece la creazione del processo figlio ha successo, il figlio viene utilizzato per gestire la nuova connessione e il processo padre torna alla fase di ascolto per nuove connessioni. Nel figlio, la connessione

accettata viene gestita con delle operazioni specifiche, come ad esempio la ricezione e l'invio di dati. Quando il figlio ha finito di gestire la connessione, il socket della connessione viene chiuso tramite la funzione close(), e il processo figlio termina.

## Dettagli implementativi ServerG

Il codice riguarda l’implementazione di un server che accetta connessioni TCP sulla porta 1026 e gestisce richieste di servizio "green pass" da parte di client.

Il server dichiara una struttura chiamata "green_pass" che contiene tre campi: una stringa chiamata "num_tesseraSanitaria" lunga 21 caratteri, un valore "data_Scadenza" di tipo time_t e un intero “tipo_Servizio”.

La funzione "main" è definita con tre argomenti: un intero "argc" e un array di stringhe "argv". Viene dichiarata una serie di variabili che verranno utilizzate per la creazione del socket, l'accettazione delle connessioni e la gestione delle richieste.

La funzione accetta solo 1 argomento, ovvero l'indirizzo IP del server a cui connettersi, se l'utente non fornisce tale argomento, il programma stampa un messaggio di errore su stderr che indica come utilizzare il programma utilizzando il primo argomento come nome del programma, e poi termina con un codice di uscita 1.

Il codice crea un socket di tipo AF_INET per accettare le connessioni in entrata tramite il protocollo TCP (SOCK_STREAM) sulla porta 1026. Il socket viene quindi associato all'indirizzo IP locale e alla porta specificata utilizzando la funzione Bind().

La funzione Listen() viene utilizzata per mettere in ascolto il socket. Il server utilizza un loop while(1) per accettare le connessioni in entrata. Quando una connessione viene accettata, il server crea un nuovo processo figlio per gestire la richiesta del client.

Il figlio legge il pacchetto di richiesta del client dalla socket e invia la richiesta a un altro server tramite una connessione TCP sulla porta 1025. Se la richiesta è per il servizio 0, il figlio legge la risposta del server remoto e la invia al client originale. Il codice utilizza le funzioni di sistema della libreria di rete BSD per creare socket, accettare connessioni, leggere e scrivere dati sulla rete.

## Dettagli implementativi ClientA

Il programma inizia dichiarando un descrittore di socket "sockfd" e una struttura "sockaddr_in" "servaddr" contenente le informazioni sull'indirizzo del server.

Successivamente, vengono verificati due requisiti: il primo è che il programma deve essere avviato con esattamente due argomenti dalla riga di comando (l'indirizzo IP del server e il numero di tessera sanitaria), altrimenti il programma emette un messaggio di errore e termina l'esecuzione.

Il secondo requisito è che il numero di tessera sanitaria deve essere esattamente di 20 caratteri, altrimenti viene emesso un messaggio di errore e il programma termina l'esecuzione. Viene quindi creato un socket utilizzando la funzione "Socket" definita nella libreria "wrapper.h", specificando l'indirizzo della famiglia di protocolli (AF_INET per IPv4), il tipo di socket (SOCK_STREAM per una connessione TCP) e il protocollo (0 per il protocollo di default).

La struttura "sockaddr_in" viene inizializzata con l'indirizzo della famiglia di protocolli (AF_INET per IPv4) e la porta a cui il server è in ascolto (la porta 1024). L'indirizzo IP passato come primo argomento viene convertito da stringa a formato binario tramite la funzione "inet_pton" e assegnato al campo "sin_addr" della struttura "servaddr".

Se la conversione fallisce, viene stampato un messaggio di errore e il programma termina l'esecuzione. Viene quindi stabilita la connessione al server tramite la funzione "Connect", specificando il descrittore del socket, l'indirizzo IP e la porta del server.

La stringa contenente il numero di tessera sanitaria viene inviata al socket tramite la funzione "write" e ne viene verificato l'avvenuto invio. Se l'invio fallisce, viene emesso un messaggio di errore e il programma termina l'esecuzione. Infine, viene stampato un messaggio di conferma dell'avvenuta consegna della tessera sanitaria e il socket viene chiuso.

## Dettagli implementativi centroVaccinale

La struttura di codice fornita è un programma in C che implementa un server che accetta connessioni da client e riceve dati dal client.

Il server legge i dati inviati dal client in una struttura chiamata "green_pass" e invia questi dati a un altro server V specificato come argomento sulla riga di comando. Il programma utilizza socket per la comunicazione tra il server e il client, nonché tra il server e il server V.

Il server utilizza la funzione fork per creare un nuovo processo figlio che gestisce ogni connessione in entrata. Il processo figlio legge i dati inviati dal client e li memorizza nella struttura "green_pass". Il processo figlio quindi crea un nuovo socket e si connette al server V utilizzando l'indirizzo IP e la porta specificati sulla riga di comando. I dati memorizzati nella struttura "green_pass" vengono quindi inviati al server V utilizzando il nuovo socket.

Il server utilizza una costante definita come SCADENZASEIMESI, che corrisponde a 15552000 secondi, ovvero 6 mesi. Viene assegnato un valore alla variabile "data_Scadenza" della struttura "green_pass" aggiungendo alla funzione time() il valore di SCADENZASEIMESI.

Il server viene eseguito con un argomento IPaddress che deve essere specificato sulla riga di comando. Se il programma viene eseguito senza un argomento IPaddress, viene stampato un messaggio di errore e il programma termina.

## Dettagli implementativi ClientS

Il codice implementa un client che si connette ad un server per verificare la validità di un Green Pass.

Nel dettaglio, il client prende in input dalla riga di comando l'indirizzo IP del server e il numero della tessera sanitaria da verificare. Viene quindi verificato che il numero di argomenti sia esattamente 3 e che il numero della tessera sanitaria abbia una lunghezza valida.

Successivamente, viene inizializzata la struttura green_pass con il numero della tessera sanitaria passato come argomento e il valore 0 per il campo tipo_Servizio.

Viene poi creato il socket utilizzando la funzione Socket definita nel file wrapper.h, specificando il protocollo AF_INET e il tipo di socket SOCK_STREAM.

Viene poi inizializzata la struttura servaddr con l'indirizzo IP e la porta del server, specificati come argomenti dalla riga di comando.

Viene quindi chiamata la funzione Connect definita nel file wrapper.h per stabilire la connessione con il server, passando il socket creato in precedenza e l'indirizzo del server. Viene quindi inviata la struttura green_pass al server utilizzando la funzione write. Se la scrittura sul socket non ha successo, viene stampato un messaggio di errore.

Successivamente, viene letto il risultato della verifica del green pass dal server utilizzando la funzione read. Se la lettura dal socket non ha successo, viene stampato un messaggio di errore.

Infine, vengono stampati i risultati della verifica: il numero della tessera sanitaria, la data di scadenza e se il green pass è valido o meno in base alla data di scadenza. Viene quindi chiusa la connessione e il programma termina con un codice di uscita 0.

## Dettagli implementativi ClientT

La struttura client in questione implementa una connessione di rete con un server utilizzando il protocollo TCP. La connessione viene stabilita creando un socket di tipo SOCK_STREAM e collegandosi all'indirizzo IP e alla porta specificati come argomenti in input alla linea di comando.

La struttura del programma utilizza una struttura personalizzata chiamata green_pass che contiene tre campi: un numero di tessera sanitaria, una data di scadenza e un tipo di servizio. I valori di questi campi vengono letti dalla linea di comando e inseriti nella struttura.

Il programma controlla se il numero di argomenti passati in input è corretto e se la lunghezza della tessera sanitaria è valida. In caso contrario, viene stampato un messaggio di errore e il programma termina.

Il programma crea un socket e imposta la famiglia di indirizzi e la porta nella struttura sockaddr_in utilizzata per la connessione. Inoltre, l'indirizzo IP passato come argomento viene convertito in formato binario utilizzando la funzione inet_pton della libreria arpa/inet.h.

Una volta stabilita la connessione, la struttura green_pass viene inviata al server utilizzando la funzione write. Il server elabora la richiesta e invia una risposta al client, che viene ricevuta utilizzando la funzione read.

Infine, il socket viene chiuso e il programma termina.

## Manuale di compilazione

Ognuno dei seguenti comandi compila un file sorgente (in formato .c) e lo unisce a eventuali librerie (come la libreria pthread per il ServerV).

Di seguito una spiegazione di ogni comando:

- *gcc -c -o wrapper.o wrapper.c*
- Il comando crea un file oggetto wrapper.o dal sorgente wrapper.c, che verrà poi utilizzato per creare i file eseguibili per ClientA, CentroVaccinale, ServerV, ServerG, ClientS, e ClientT. Il flag -c indica che si vuole compilare il sorgente in un file oggetto, mentre il flag -o indica il nome del file oggetto che verrà creato.

- *gcc -o ClientA clientA.c wrapper.o*
- Il comando crea l'eseguibile ClientA dal sorgente clientA.c e dal file oggetto wrapper.o. Il flag -o indica il nome del file eseguibile che verrà creato.

- *gcc -o CentroVaccinale centroVaccinale.c wrapper.o*
- Il comando crea l'eseguibile CentroVaccinale dal sorgente centroVaccinale.c e dal file oggetto wrapper.o.

- *gcc -pthread -o ServerV serverV.c wrapper.o*
- Il comando crea l'eseguibile ServerV dal sorgente serverV.c, dal file oggetto wrapper.o e dalla libreria pthread (necessaria per il supporto ai thread in C).

- *gcc -o ServerG serverG.c wrapper.o*
- Il comando crea l'eseguibile ServerG dal sorgente serverG.c e dal file oggetto wrapper.o.

- *gcc -o ClientS clientS.c wrapper.o*
- Il comando crea l'eseguibile ClientS dal sorgente clientS.c e dal file oggetto wrapper.o.

- *gcc -o ClientT clientT.c wrapper.o*
- Il comando crea l'eseguibile ClientT dal sorgente clientT.c e dal file oggetto wrapper.o.

# Manuale di esecuzione

*Dopo la corretta compilazione, per eseguire il progetto, l'utente deve seguire queste istruzioni:

1) Avviare il ServerV con il comando

*./ServerV*

Il ServerV viene eseguito in background.

2) Avviare il CentroVaccinale con il comando

*./CentroVaccinale “indirizzo ServerV”*

L'indirizzo del ServerV deve essere passato come parametro al CentroVaccinale.

3) Avviare il ClientA con il comando

*./ClientA “indirizzo di CentroVaccinale” “tessera sanitaria”*

L'indirizzo del CentroVaccinale e il numero della tessera sanitaria devono essere passati come parametri al ClientA.

4) Avviare il ServerG con il comando

*./ServerG "indirizzo ServerV”*

L'indirizzo del ServerV deve essere passato come parametro al ServerG.

5) Avviare il ClientS con il comando

*./ClientS "indirizzo ServerG" "tessera sanitaria”*

L'indirizzo del ServerG e il numero della tessera sanitaria devono essere passati come parametri al ClientS.

6) Avviare il ClientT con il comando

*./ClientT "indirizzo ServerG" "tessera sanitaria" "V  I”*

L'indirizzo del ServerG, il numero della tessera sanitaria e la lettera "V" o "I" (per indicare la validità o l'invalidità del Green Pass) devono essere passati come parametri al ClientT.

In sintesi, le istruzioni per la compilazione sono utilizzate per generare gli eseguibili per il progetto, mentre le istruzioni per l'esecuzione sono utilizzate per avviare i componenti del progetto e farli interagire tra loro per gestire il Green Pass.

*I doppi apici non vanno usati, sono puramente indicativi.

Separare semplicemente i parametri da passare utilizzando il backspace da tastiera.