

UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE
SCUOLA INTERDIPARTIMENTALE DELLE SCIENZE, DELL'INGEGNERIA E DELLA SALUTE
INFORMATICA
CORSO DI RETI DI CALCOLATORI E LABORATORIO DI RETI DI CALCOLATORI



Vanilla Green Pass

Proponenti:	
Calcopietro Francesco	0124002090
Caruso Denny	0124002062

Data di Consegna:
14/02/2022

Anno Accademico:
2021 – 2022

Categoria:
Green Pass

Questa pagina è stata lasciata bianca di proposito

Questa pagina è stata lasciata bianca di proposito

Indice

1 – Descrizione del progetto.....	1
2 – Descrizione e schemi dell'architettura.....	5
3 – Descrizione e schemi del protocollo applicazione.....	8
4 – Dettagli implementativi comuni.....	21
5 – Dettagli implementativi dei client.....	25
6 – Dettagli implementativi dei server.....	?
7 - Manuale utente.....	?
7.1 – Istruzioni per la compilazione.....	?
7.2 – Istruzioni per l'esecuzione.....	?
8 – Sviluppi futuri.....	?

Elenco delle figure

Figura 2.1: Diagramma Architettura di Rete Vanilla Green Pass.....	7
Figura 3.1: Diagramma delle sequenze ClientCitizen-CentroVaccinale-ServerV.....	12
Figura 3.2: Diagramma delle sequenze ClientS-ServerG-ServerV.....	16
Figura 3.3: Diagramma delle sequenze ClientT-ServerG-ServerV.....	17
Figura 3.4: Istanza del diagramma delle sequenze ClientCitizen-CentroVaccinale-ServerV.....	17
Figura 3.5: Istanza del diagramma delle sequenze ClientS-ServerG-ServerV.....	18
Figura 3.6: Istanza del diagramma delle sequenze ClientT-ServerG-ServerV.....	19
Figura 4.1: Dipendenze sorgenti e librerie Vanilla Green Pass.....	23
Figura 7.1: Esecuzione di ServerV.....	?
Figura 7.2: Esecuzione di ServerG.....	?
Figura 7.3: Esecuzione di CentroVaccinale.....	?
Figura 7.4: Esecuzione di ClientCitizen.....	?
Figura 7.5: Esecuzione di ClientS.....	?
Figura 7.6: Esecuzione di ClientT.....	?

Bibliografia

GaPiL – Guida alla Programmazione in Linux, Simone Piccardi
Dhananjay M. Dhamdhare - Sistemi Operativi, McGraw-Hill Education

Sitografia

<https://www.un.org/en/file/45419>
<https://github.com/dennewbie/VanillaGreenPass>
<https://www.dgc.gov.it/web/>
<https://gapil.gnulinux.it/>
<https://valgrind.org/>
<https://staruml.io/>

Note Aggiuntive

- Per informazioni aggiornate e complete in merito all'attuale regolamentazione del Green Pass si faccia riferimento al terzo link riportato nella sitografia di questo documento.
- Il repository GitHub verrà reso pubblico solo a partire da gennaio 2023.
- Per una migliore visualizzazione delle figure riportate in elenco, consultare i file in formato PDF in allegato.

Descrizione del progetto

1 - Descrizione del progetto

La Certificazione verde “COVID-19 - *EU digital COVID certificate*” (più semplicemente detta Green Pass) nasce su proposta della Commissione europea per agevolare la libera circolazione in sicurezza dei cittadini nell’Unione europea durante la pandemia di COVID-19.

È una certificazione digitale e stampabile (cartacea), che contiene un codice a barre bidimensionale (QR Code) e un sigillo elettronico qualificato. In Italia, viene emessa soltanto attraverso la Piattaforma nazionale DGC del Ministero della Salute. Il Green Pass facilita i viaggi in Europa e nel mondo. Nel nostro Paese rende più sicuri i cittadini al lavoro, a scuola e in molte attività quotidiane. Il Green Pass presenta le seguenti caratteristiche:

- formato digitale e/o cartaceo
- verificabile con QR code
- gratis per tutti
- in italiano e in inglese, più francese o tedesco
- sicura e protetta
- valida in tutta l’UE e altri Paesi non-UE

Il Green Pass attesta una delle seguenti condizioni:

- aver fatto la vaccinazione anti COVID-19 (in Italia viene emessa dopo ogni dose di vaccino);
- essere negativi al test antigenico rapido nelle ultime quarantotto ore o al test molecolare nelle ultime settantadue ore;
- essere guariti dal COVID-19 da non più di sei mesi.

Il Regolamento europeo sulla Certificazione è entrato in vigore il 1 luglio 2021 in tutti i Paesi dell’Unione e avrà durata di un anno. L’Italia ha anticipato l’emissione della Certificazione verde COVID-19 al 17 giugno 2021 e ne ha esteso progressivamente l’utilizzo sul territorio nazionale. Per maggiori informazioni consultare il terzo link riportato all’interno della sitografia di questo documento.

Ora che è stato introdotto cosa sia il Green Pass e come viene utilizzato nella vita reale, si propone l’esemplificazione a scopo didattico dello stesso. Al fine di non creare confusione si farà riferimento al Green Pass semplificato realizzato nell’ambito di questo progetto come “Vanilla Green Pass” ovvero un Green Pass senza particolari personalizzazioni e semplificato nel suo funzionamento. Un Vanilla Green Pass è valido a partire dal momento del suo rilascio dopo la vaccinazione fino al giorno in cui passeranno sei mesi rispetto al primo giorno del mese nel quale è stato rilasciato. Ciò vuol dire che nel momento in cui un Vanilla Green Pass è rilasciato nel giorno 15/01/2022, verrà preso in considerazione il primo giorno del mese di gennaio (ovvero 01/01/2022) e sommati sei mesi (01/06/2022). A partire dalla data risultante sarà possibile effettuare una nuova dose di vaccino. Si precisa che con Vanilla Green Pass non si fa distinzione tra quello ottenuto dopo la prima dose, la seconda dose o la dose booster. Inoltre, un Vanilla Green Pass è ottenibile solo mediante vaccinazione.

In base a quanto detto finora un Vanilla Green Pass rilasciato l’ultimo giorno del mese di gennaio, avrà la stessa data limite di scadenza di un Vanilla Green Pass rilasciato nei primi giorni del mese. Queste sono le prime semplificazioni più importanti apportate a un Vanilla Green Pass rispetto al Green Pass reale. In base a ciò, si noti che non sempre la validità del Vanilla Green Pass risulterà essere di sei mesi, bensì oscillante tra i cinque ed i sei mesi a seconda se il vaccino è stato inoculato rispettivamente a fine mese oppure ad inizio mese. Tale semplificazione è stata apportata così da velocizzare e facilitare il

calcolo del periodo di validità di un Vanilla Green Pass evitando inoltre l'ottenimento di date il cui giorno non è realmente esistente; si pensi al 29/02/2022 o al 31/04/2022.

La traccia del progetto richiede di progettare e implementare un servizio di gestione dei Green Pass. La traccia consultabile dal file "tracciaProgetto.pdf" è stata ampliata ed arricchita. Di seguito si riportano le specifiche risultanti di Vanilla Green Pass in quanto sistema software.

- Un utente, una volta effettuata la vaccinazione, tramite un client si collega ad un centro vaccinale e comunica il codice della propria tessera sanitaria. Si suppone che ogni utente, che chiameremo da qui in poi cittadino, abbia un codice di tessera sanitaria univoco, permanente e senza una scadenza associata alla relativa tessera sanitaria. Si suppone che tale codice non segua un particolare formato, ma abbia un numero minimo di caratteri pari a venti. I caratteri possono essere lettere, numeri, simboli.
- Il centro vaccinale comunica al ServerV il codice ricevuto dal client ed il periodo di validità del Vanilla Green Pass. Si suppone che il periodo di validità venga indicato mediante una data che corrisponderà alla data a partire dalla quale sarà possibile essere sottoposti all'inoculazione di una nuova dose di vaccino. Di conseguenza, rappresenterà anche il periodo di validità del Vanilla Green Pass associato a quel codice di tessera sanitaria. Si suppone che il formato della data sia del tipo DD-MM-YYYY dove DD rappresenta il giorno espresso da un intero non negativo che va da "01" a "31" in accordo con i giorni dei singoli mesi, MM rappresenta il mese espresso da un intero non negativo che va da "01" a "12" e YYYY rappresenta l'anno espresso in quattro cifre. Il centro vaccinale oltre a comunicare col ServerV, dialoga anche con il client del cittadino che sta cercando di sottoporsi all'inoculazione del vaccino. Quest'ultimo attenderà l'esito della richiesta di vaccinazione. Qualora l'esito sarà positivo, allora il vaccino può essere inoculato, altrimenti non sarà possibile inoculare il vaccino.
- Un ClientS, per verificare se un Vanilla Green Pass è valido, invia il codice di una tessera sanitaria al ServerG il quale richiede al ServerV il controllo della validità. Il ClientS può essere considerato come quello installato presso il terminale di un ristorante, il cui proprietario o l'addetto apposito utilizza per verificare i Vanilla Green Pass dei clienti all'ingresso. Si suppone che l'esito della verifica di un Vanilla Green Pass dipenda da due fattori: la data di scadenza del Vanilla Green Pass (o periodo di validità) in base alla definizione che ne è stata data nei paragrafi precedenti e uno stato che può assumere due valori, attivo oppure non attivo (rispettivamente uno o zero). Questo vuol dire che un Vanilla Green Pass sebbene sia ancora valido in termini di date perché non è ancora scaduto, può risultare non valido in quanto il cittadino possessore del codice di tessera sanitaria associata a quel Vanilla Green Pass è risultato positivo al (o alternativamente è stato contagiato dal) COVID-19.

In maniera del tutto analoga ad un Vanilla Green Pass può essere associato uno stato valido, sebbene sia scaduto in base al periodo di validità ad esso associato. Ciò può accadere nel momento in cui si supera il giorno di scadenza previsto. Però non rappresenta comunque un problema. Infatti, un Vanilla Green Pass per essere considerato valido deve presentare entrambe le caratteristiche. Tali caratteristiche sono adeguatamente controllate dal ServerV incaricato.

- Un ClientT, inoltre, può invalidare o ripristinare la validità di un Vanilla Green Pass comunicando al ServerG il contagio o la guarigione di una persona attraverso il codice della tessera sanitaria. Il ServerG provvederà a mettersi in comunicazione con il ServerV, il quale a sua volta aggiornerà lo stato del corrispettivo Vanilla Green Pass se esistente. Da quest'ultimo aspetto si intende che così come tante altre operazioni effettuate a livello implementativo, anche la richiesta di invalidare o ripristinare la validità di una Vanilla Green Pass può non andare a buon fine. Per esempio, come detto poc'anzi, se non esiste un Vanilla Green Pass associato al codice di tessera sanitaria inviato dal ClientT. Per il ClientS, il controllo della validità di un Vanilla Green Pass può altresì fallire.

Inoltre, si mettono in luce i seguenti aspetti del sistema software realizzato.

- Si consideri la circostanza secondo la quale un cittadino possa tentare di richiedere nuovamente il vaccino dopo qualche giorno dalla precedente inoculazione, o comunque prima che passi il tempo stabilito in base alla regolamentazione vigente. Il sistema software da realizzare deve prevedere un controllo tale per cui, in tali circostanze non venga inoculato al cittadino un'ulteriore dose.
- Si consideri la circostanza in cui possano esservi errori di input da parte dell'utente. In tali circostanze il sistema software deve riconoscere, individuare e segnalare l'errore evitando di ritrovarsi uno stato inconsistente.
- Si consideri la circostanza in cui possano esservi indirizzi IP di configurazione non validi. In tali circostanze il sistema software deve riconoscere, individuare e segnalare l'errore evitando di ritrovarsi uno stato inconsistente. In maniera del tutto analoga il sistema software in tutto il suo ciclo di vita deve intercettare e riconoscere stati d'errore generati da chiamate a funzioni o altro.
- La persistenza dei dati relativi ai Vanilla Green Pass, ovvero codice di tessera sanitaria, periodo di validità e stato di validità, sono assicurati mediante il salvataggio degli stessi sul file system, più in particolare su un supporto di memorizzazione di massa in un apposito file "serverV.dat", situato nella sottocartella "data". I file di configurazione delle varie entità del sistema software che prenderemo in considerazione in maniera più approfondita nelle prossime sezioni, sono nel formato ".conf" e sono memorizzati sul supporto di memorizzazione di massa nella sottocartella "conf". Infine, i sorgenti sono memorizzati nella sottocartella "src", mentre tutti gli eseguibili da testare nella sottocartella "bin". Questa documentazione è invece salvata nella cartella "doc".
- Le entità che verranno individuate sono eseguibili anche su macchine differenti, andando a modificare opportunamente i relativi file di configurazione.
- Non si gestisce la risoluzione dei nomi a dominio.
- Non si gestisce la risoluzione degli indirizzi IPv6, né la doppia gestione di indirizzi sia IPv4 che IPv6, ma soltanto IPv4.
- Non si gestisce l'implementazione della persistenza dei dati mediante database.
- Non si gestisce l'implementazione di tecniche di crittografia per i dati scambiati via rete, né di serializzazione dei dati stessi all'interno dei dispositivi di memorizzazione di massa.

Descrizione e schemi dell'architettura

2 - Descrizione e schemi dell'architettura

Vanilla Green Pass sfrutta un'architettura di rete di tipo Client/Server multilivello in cui i vari client e server possono essere in esecuzione sulla stessa macchina oppure su macchine differenti. Un'architettura Client/Server è un'architettura all'interno della quale vi sono una o più entità Client che richiedono un servizio e una o più entità Server che offrono un servizio. Se si tratta di un'architettura Client/Server multilivello, questo implica che vi possano essere una o più entità intermedie costituenti dei livelli per l'appunto tra il Server che offre un servizio e il Client che lo richiede. Nel caso di Vanilla Green Pass si individuano tre livelli.

In particolare, un primo livello è quello al quale accedono i client che a vario titolo possono segnalare la propria volontà di effettuare il vaccino per ricevere il Vanilla Green Pass e attendere l'esito della richiesta, oppure possono verificare la validità di un Vanilla Green Pass associato a un codice di tessera sanitaria fornito in input, o ancora invalidare o riattivare un Vanilla Green Pass associato a un codice di tessera sanitaria fornito in input. Un secondo livello dell'architettura è rappresentato da quelli che sono il ServerG e il CentroVaccinale. Il primo dei due svolge una funzione da "gateway" da e verso il ServerV, mettendo in comunicazione il ClientT ed il ClientS con il ServerV. Invece, il CentroVaccinale è come se svolgesse la stessa funzione del ServerG ma questa volta mettendo in comunicazione il client del cittadino che richiede di vaccinarsi col ServerV. Il terzo livello, infine, è dato dal nodo centrale e dominante dell'architettura ovvero il ServerV stesso: il "cuore" di Vanilla Green Pass.

A questo punto si può stabilire che vengono individuate le seguenti entità all'interno di Vanilla Green Pass:

- Client del cittadino che richiede di vaccinarsi (da ora in poi verrà fatto riferimento a quest'ultimo col nome di ClientCitizen);
- ClientS;
- ClientT;
- CentroVaccinale;
- ServerG;
- ServerV.

Un ClientCitizen comunica solo con un CentroVaccinale. Un CentroVaccinale, invece, comunica sia con più ClientCitizen che con il ServerV. Si noti come si faccia riferimento al ServerV con l'articolo determinativo "il" e non con quello indeterminativo "un" ad indicare che in Vanilla Green Pass si suppone l'esistenza di un solo ServerV, e grazie a questa caratteristica sarà poi possibile effettuare un controllo di tipo centralizzato sui vari Vanilla Green Pass memorizzati mediante file system. Se questo comporta delle semplificazioni da un lato, è anche vero che lo rende un punto debole dell'architettura: un single point of failure.

Un ClientS comunica solo con un ServerG. Un ClientT comunica solo con un ServerG. Un ServerG, invece, comunica sia con più ClientS, sia con più ClientT, che con il ServerV. Si noti anche in questo caso la scelta degli articoli determinativi e indeterminativi. Di conseguenza è possibile avere più istanze di un ServerG in esecuzione, più istanze di un CentroVaccinale in esecuzione, così come di ClientCitizen, ClientS e ClientT, ma non più istanze del ServerV in esecuzione.

Il ServerV comunica sia con più CentriVaccinali, sia con più ServerG. Quindi l'architettura a tre livelli nasce esattamente in questo punto, ovvero il ServerV comunica mediante il ServerG con il ClientS e con il ClientT e mediante il CentroVaccinale con il ClientCitizen.

Di seguito si allega uno schema riassuntivo dell'architettura di rete di Vanilla Green Pass illustrato mediante diagramma delle componenti.

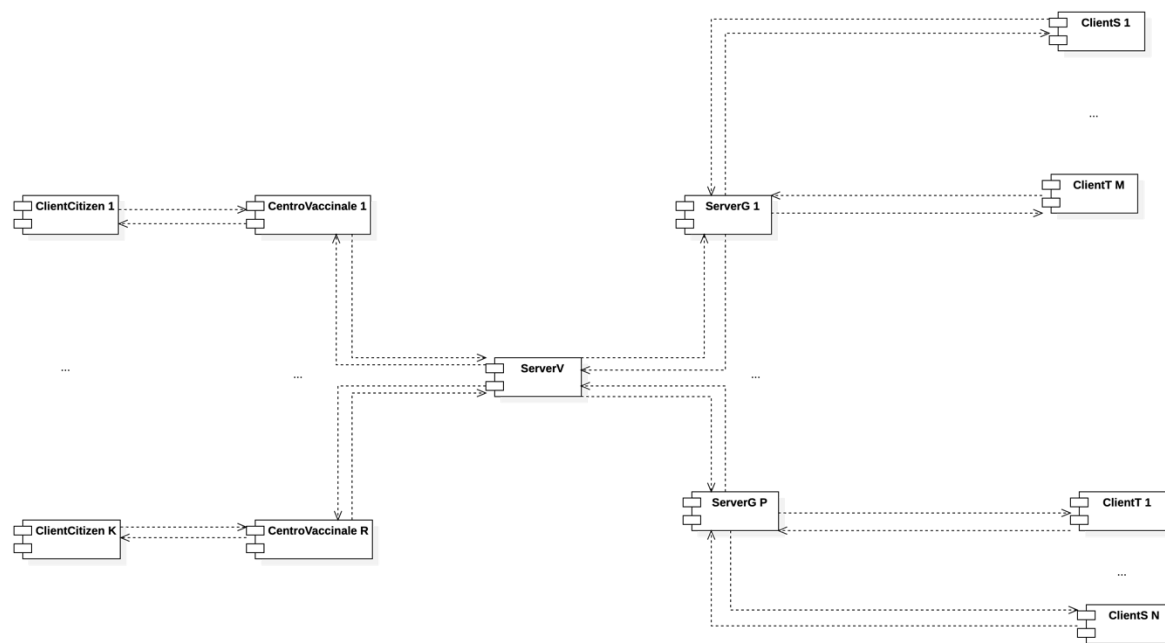


Figura 2.1: Diagramma Architettura di Rete Vanilla Green Pass

Un ClientCitizen, un ClientS e un ClientT svolgono soltanto un ruolo di client: ovvero richiedono un servizio sulla rete (tipicamente a un server). Invece, il ServerV svolge solo un ruolo di server: ovvero offre un servizio sulla rete (tipicamente ai client). Infine, un ServerG e un CentroVaccinale svolgono sia un ruolo da server che da client. Questi ultimi svolgono un ruolo da client quando richiedono un servizio al ServerV, mentre svolgono un ruolo da server quando offrono un servizio ai client coi quali hanno stabilito una connessione. ServerG e CentroVaccinale possono anche essere visti come dei gateway (o addirittura degli intermediari tra il ServerV e il client di turno) verso i quali i client indirizzano le proprie richieste e dai quali gli stessi client ricevono delle risposte alle loro richieste.

C'è da sottolineare però che ServerG e CentroVaccinale offrono servizi differenti e non sono la stessa entità a livello di rete. Un ServerG offre un servizio ai ClientS e ai ClientT (rispettivamente verifica validità Vanilla Green Pass e gestione rinnovo/invalidazione Vanilla Green Pass), mentre il CentroVaccinale offre un servizio ai ClientCitizen (richiesta di vaccinazione e ottenimento periodo di validità).

Si noti come nello schema si utilizzino alcune variabili quali: K, M, N, P, R. Tali lettere stanno a significare che possono esservi rispettivamente più istanze di ClientCitizen, più istanze di ClientT, più istanze di ClientS, più istanze di ServerG, più istanze di CentroVaccinale. Inoltre, non necessariamente i valori numerici strettamente positivi assegnati a tali variabili sono uguali tra di loro.

Descrizione e schemi del protocollo applicazione

3 - Descrizione e schemi del protocollo applicazione

I protocolli di livello applicazione adottati da ognuna delle entità software precedentemente individuate sono semplici e veloci. Si prendono ora in considerazione i vari protocolli di livello applicazione adottati per ogni entità presente, ovvero l'insieme delle regole di comunicazione adottate da ogni entità (pacchetti eventuali inviati, dati inviati non all'interno di pacchetti, ordine delle operazioni, etc.). Si suppone che la fase di connessione iniziale all'entità con la quale è necessario effettuare il collegamento e la fase di disconnessione finale vadano a buon fine.

Il protocollo di livello applicazione di un ClientCitizen prevede come prima operazione quella di collegarsi a un CentroVaccinale. A questo punto il ClientCitizen invia il codice della tessera sanitaria al CentroVaccinale al quale si è collegato. Dopodiché si mette in attesa di una risposta da parte del CentroVaccinale che consisterà in un pacchetto di livello applicazione strutturato nel seguente modo:

```
Pacchetto centroVaccinaleReplyToClientCitizen {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    char greenPassExpirationDate[DATE_LENGTH]  
    unsigned short int requestResult  
}
```

Tale pacchetto contiene il codice della tessera sanitaria del ClientCitizen che ha fatto la richiesta di vaccinazione, la data di scadenza del Vanilla Green Pass associato a tale codice di tessera sanitaria e l'esito della richiesta di vaccinazione. Quest'ultimo campo è fondamentale, infatti potrebbe capitare che un ClientCitizen abbia già effettuato la vaccinazione di recente e benché richieda di farne un'altra, non gli si può effettivamente inoculare un altro vaccino a distanza così ridotta. Ragion per cui, per ora ci limitiamo a dire che "lato server si verifica se il ClientCitizen può vaccinarsi" per poi analizzare in seguito come avviene tale controllo.

In ogni caso l'esito di richiesta della vaccinazione rappresenta un tipico campo booleano (vero o falso), dove "vero" sta a significare che la richiesta di vaccinazione è stata accettata e il cittadino può vaccinarsi. Mentre "falso" sta a significare che la richiesta di vaccinazione non è stata accettata e il cittadino non può vaccinarsi. In quest'ottica, quando l'esito è positivo allora la data all'interno del pacchetto è la data di scadenza del Vanilla Green Pass, mentre se l'esito è negativo, allora la data all'interno del pacchetto rappresenta la data a partire dalla quale è possibile effettuare una nuova inoculazione del vaccino (o alternativamente la data di scadenza del Vanilla Green Pass collegata all'ultima richiesta di vaccinazione andata a buon fine e conseguente vaccinazione andata a buon fine).

Una volta arrivato tale pacchetto al ClientCitizen da parte del CentroVaccinale, allora il protocollo di livello applicazione del ClientCitizen termina e si procede con operazioni interne proprie del ClientCitizen come la verifica dell'esito della richiesta di vaccinazione, alcune stampe, il rilascio delle risorse allocate, la chiusura della connessione col CentroVaccinale e la terminazione. Per realizzare una nuova richiesta di vaccinazione, è necessario riavviare nuovamente il ClientCitizen.

Il protocollo di livello applicazione di un CentroVaccinale prevede come prima operazione quella di impostare tutto il necessario al fine di accettare connessioni da più ClientCitizen. Dopodiché il CentroVaccinale all'infinito accetta una connessione da un ClientCitizen, si mette in collegamento con il ServerV e gestisce la richiesta del ClientCitizen in maniera opportuna in termini di risorse come vedremo successivamente. In quanto a comunicazione effettiva, il CentroVaccinale si mette innanzitutto in attesa di ricevere un codice di tessera sanitaria dal ClientCitizen che si è collegato.

Una volta ricevuto il codice di tessera sanitaria dal ClientCitizen, è necessario che il CentroVaccinale si identifichi presso il ServerV con un opportuno identificativo. A tal proposito invia al ServerV il valore "0" che, come vedremo successivamente, identificherà in un ServerV le richieste ricevute dei centri vaccinali. A questo punto il CentroVaccinale crea e invia un pacchetto di livello applicazione strutturato nel seguente modo al ServerV, dove all'interno del primo campo si immette il codice di tessera sanitaria ricevuto dal ClientCitizen e all'interno del secondo campo si immette la data prevista di scadenza del Vanilla Green Pass che eventualmente sarà generato (eventualmente dal momento che non è detto che la richiesta di vaccinazione del ClientCitizen vada a buon fine).

```
Pacchetto centroVaccinaleRequestToServerV {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    char greenPassExpirationDate[DATE_LENGTH]  
}
```

Ora il CentroVaccinale si metterà in attesa di una risposta da parte del ServerV che consisterà in un pacchetto di livello applicazione strutturato nel seguente modo, dove il codice di tessera sanitaria è sempre quello del ClientCitizen che ha effettuato la richiesta di vaccinazione, la data è in generale la data di scadenza del Vanilla Green Pass associato a quel codice di tessera sanitaria, ma più nello specifico può essere la data di scadenza del Vanilla Green Pass in seguito a una richiesta di vaccinazione andata a buon fine e conseguente vaccinazione appena richiesta dal ClientCitizen, oppure la data di scadenza del Vanilla Green Pass in seguito all'ultima ma non attuale richiesta di vaccinazione andata a buon fine e conseguente vaccinazione. Infine, l'esito di richiesta della vaccinazione è un valore booleano che assume valore "vero" nel momento in cui la richiesta di vaccinazione del ClientCitizen va a buon fine e valore "falso" nel momento in cui la richiesta di vaccinazione del ClientCitizen non va a buon fine.

```
Pacchetto serverV_ReplyToCentroVaccinale {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    char greenPassExpirationDate[DATE_LENGTH]  
    unsigned short int requestResult  
}
```

Una volta ricevuta la risposta dal ServerV, il CentroVaccinale crea e invia al ClientCitizen col quale è collegato un pacchetto di livello applicazione strutturato nel modo visto prima e denominato "Pacchetto CentroVaccinaleReplyToClientCitizen" i cui campi sono compilati coerentemente con la risposta ottenuta poc'anzi dal ServerV.

Ora che la risposta al ClientCitizen è stata inviata, il CentroVaccinale procede a liberare le risorse acquisite durante l'elaborazione della risposta, chiude le relative connessioni aperte e passa ad attendere una successiva eventuale richiesta di un ClientCitizen. Il CentroVaccinale quindi non termina in quanto deve adempiere le sue mansioni da "server" nei confronti dei ClientCitizen.

Il protocollo di livello applicazione del ServerV prevede come prima operazione quella di impostare tutto il necessario al fine di accettare connessioni da più CentriVaccinali e da più ServerG. Altre operazioni effettuate dal ServerV saranno commentate nella sezione riguardante i dettagli implementativi lato server, in quanto non riguardano la descrizione del protocollo di livello applicazione usato dal ServerV. A questo punto il ServerV esegue tre operazioni fondamentali all'infinito:

- accettare una nuova connessione da parte di un'entità che può essere un CentroVaccinale oppure un ServerG;
- identificare chi si è messo in collegamento mediante l'attesa di un identificativo seriale;
- gestire il collegamento e le conseguenti richieste proprie del collegamento in maniera opportuna.

Per capire con chi “sta comunicando”, il ServerV una volta accettata una nuova connessione, si mette in attesa di ricevere un identificativo che può corrispondere al valore “0” se la nuova connessione proviene da un CentroVaccinale, al valore “1” se la nuova connessione proviene da un ServerG che ha bisogno di trovare risposta a una richiesta inviata da un ClientS, oppure al valore “2” se la nuova connessione proviene da un ServerG che ha bisogno di trovare risposta a una richiesta inviata da un ClientT.

Una volta che il ServerV ha identificato la tipologia di “pari della comunicazione” col quale si è messo in collegamento, lo stesso ServerV gestisce la richiesta pervenuta in maniera adeguata. Si analizza prima il caso in cui l'identificativo ricevuto (che corrisponde sostanzialmente ad un “unsigned short int”) sia pari a “0”, ovvero la nuova connessione che è stata accettata, è stabilita con un CentroVaccinale.

In tal caso, il ServerV si mette subito in attesa del pacchetto di livello applicazione visto precedentemente denominato “Pacchetto centroVaccinaleRequestToServerV”. Una volta ricevuto quest'ultimo e dopo una verifica interna per stabilire se la richiesta del ClientCitizen inoltrata al ServerV mediante il CentroVaccinale può essere accettata o meno, operazioni interne di modifica, salvataggio e aggiornamento dei Vanilla Green Pass, il ServerV crea e invia un pacchetto di livello applicazione visto precedentemente e denominato “Pacchetto serverV_ReplyToCentroVaccinale”. Quest'ultimo pacchetto viene “costruito” in base agli esiti delle verifiche condotte dal ServerV. A questo punto il ServerV rilascia le risorse acquisite e allocate per gestire questa richiesta ricevuta dal CentroVaccinale, chiude le relative connessioni aperte e passa ad attendere una successiva eventuale richiesta. Il ServerV, quindi, non termina in quanto deve adempiere le sue mansioni da “server” nei confronti dei CentriVaccinali e dei ServerG.

Sicuramente il ServerV poteva essere implementato diversamente per esempio come un server multi-processo anziché server multi-thread. In maniera analoga il protocollo di livello applicazione poteva essere progettato diversamente, per esempio l'identificativo inviato dall'entità che si pone in collegamento con il ServerV poteva essere incluso nei vari pacchetti di livello applicazione che vengono scambiati, oppure il ServerV si poteva “mettere in ascolto su socket differenti” a seconda delle varie entità che possono stabilire una comunicazione. Di questo e di altri aspetti si approfondirà nella sezione riguardante i dettagli implementativi dei server.

Di seguito si allega uno schema riassuntivo utile a chiarire i protocolli di livello applicazione delle entità finora considerate: ClientCitizen, CentroVaccinale, ServerV (nel suo interfacciamento con un CentroVaccinale).

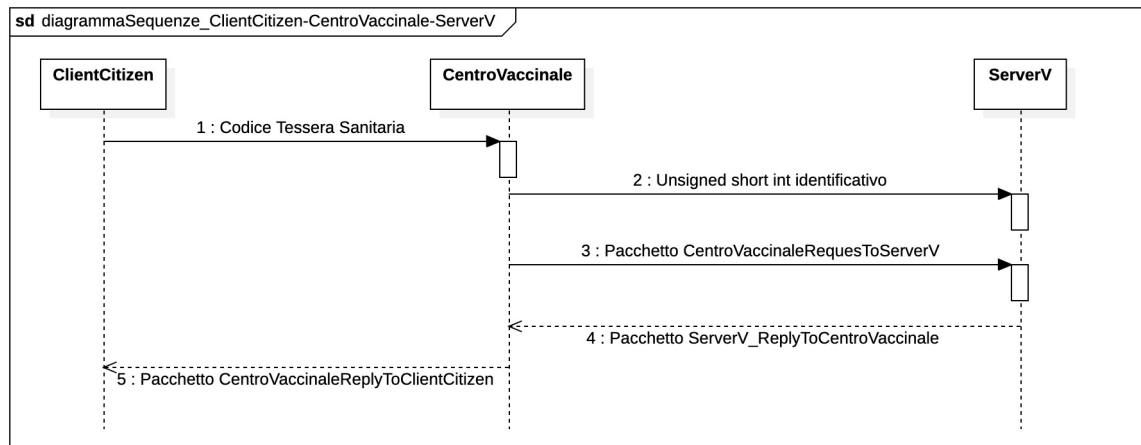


Figura 3.1: Diagramma delle sequenze ClientCitizen-CentroVaccinale-ServerV

Si analizza ora il caso in cui l'identificativo ricevuto (che corrisponde sostanzialmente ad un “unsigned short int”) sia pari a “1”, ovvero la nuova connessione che è stata accettata, è stabilita con un ServerG che ha necessità di rispondere a sua volta ad una richiesta proveniente da un ClientS. In tal caso il ServerV si mette subito in attesa di un codice di tessera sanitaria da parte del ServerG. Una volta ricevuto quest'ultimo e dopo una verifica interna per stabilire se la richiesta del ClientS inoltrata al ServerV mediante il ServerG può avere responso positivo o meno, il ServerV crea e invia un pacchetto di livello applicazione riportato qui di seguito al ServerG in collegamento. In particolare, il codice della tessera sanitaria è quello in base al quale si verifica se vi è un Vanilla Green Pass associato, non scaduto e attualmente attivo. L'esito della richiesta è un valore booleano e corrisponde alla validità del Vanilla Green Pass (valore “1”) o alla sua non validità (valore “0”).

```

Pacchetto serverV_ReplyToServerG_clientS {
  char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]
  unsigned short int requestResult
}
  
```

Una volta inviato tale pacchetto di livello applicazione, il ServerV rilascia le risorse acquisite e allocate per gestire questa richiesta ricevuta dal ServerG, chiude le relative connessioni aperte e passa ad attendere una successiva eventuale richiesta.

Si analizza ora il caso in cui l'identificativo ricevuto (che corrisponde sostanzialmente ad un “unsigned short int”) sia pari a “2”, ovvero la nuova connessione che è stata accettata, è stabilita con un ServerG che ha necessità di rispondere a sua volta ad una richiesta proveniente da un ClientT. In tal caso il ServerV si mette subito in attesa di un pacchetto di livello applicazione illustrato qui di seguito da parte del ServerG. In particolare, il codice della tessera sanitaria è quello del cittadino del quale si vuole andare a modificare lo stato di attivazione del Vanilla Green Pass (se presente).

Per esempio, il cittadino è risultato positivo al COVID-19 e quindi va invalidato per un certo periodo di tempo il suo Vanilla Green Pass, senza andare ad agire sul periodo di validità. Oppure il cittadino è risultato nuovamente negativo e quindi il suo Vanilla Green Pass può essere riattivato, nuovamente agendo solo sullo stato di attivazione e non sul periodo di validità del Vanilla Green Pass. Tale valore è evidente che può essere pari a “0” se si vuole invalidare il Vanilla Green Pass del cittadino che ha codice di tessera sanitaria pari al primo campo del seguente pacchetto, oppure valore pari a “1” se si vuole riattivare il Vanilla Green Pass del cittadino che ha codice di tessera sanitaria pari al primo campo del seguente pacchetto.

```
Pacchetto serverG_RequestToServerV_onBehalfOfClientT {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    unsigned short int updateValue  
}
```

Una volta ricevuto quest’ultimo pacchetto e dopo una verifica interna per stabilire se la richiesta del ClientT inoltrata al ServerV mediante il ServerG può avere responso positivo o meno, operazioni interne di modifica, salvataggio e aggiornamento dei Vanilla Green Pass, il ServerV crea e invia un pacchetto di livello applicazione riportato qui di seguito al ServerG in collegamento.

In particolare, il codice della tessera sanitaria è quello associato ad un Vanilla Green Pass di cui ne è stata richiesta la riattivazione o l’invalidazione. L’esito dell’aggiornamento è un valore booleano e corrisponde al fatto che è stato possibile effettuare l’aggiornamento dello stato di validità del Vanilla Green Pass (valore “1”) o al fatto che non è stato possibile effettuare l’aggiornamento dello stato di validità del Vanilla Green Pass (valore “0”).

```
Pacchetto serverV_ReplyToServerG_clientT {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    unsigned short int updateResult  
}
```

Una volta inviato tale pacchetto di livello applicazione, il ServerV rilascia le risorse acquisite e allocate per gestire questa richiesta ricevuta dal ServerG, chiude le relative connessioni aperte e passa ad attendere una successiva eventuale richiesta.

Sebbene gli aspetti seguenti siano propri della sezione riguardante i dettagli implementativi del ServerV, è necessario dare un rapido cenno a ciò che viene utilizzato per la verifica interna e alle operazioni di gestione del ServerV sia per gestire richieste dei ClientCitizen, sia quelle dei ClientS che quelle dei ClientT. In particolare, si impiegano:

- thread per una gestione rapida, veloce ed economica delle singole richieste ricevute dal ServerV;
- file dedicato memorizzato su supporto di memorizzazione di massa e persistente (file system) per tenere traccia e conservare i dati relativi a tutti i Vanilla Green Pass dei ClientCitizen, gli aggiornamenti dei ClientT e i controlli dei ClientS;
- un mutex per l’accesso in mutua esclusione al file contenente i dati relativi a tutti i Vanilla Green Pass dei ClientCitizen e un altro mutex per la generazione e l’assegnazione del “descrittore” affidato al singolo thread.

Il protocollo di livello applicazione del ServerG prevede come prima operazione quella di impostare tutto il necessario al fine di accettare connessioni da più ClientS e da più ClientT. Altre operazioni effettuate dal ServerG saranno commentate nella sezione riguardante i dettagli implementativi lato server, in quanto non riguardano la descrizione del protocollo di livello applicazione usato dal ServerG.

A questo punto il ServerG esegue tre operazioni fondamentali all'infinito:

- accettare una nuova connessione da parte di un'entità che può essere un ClientS oppure un ClientT;
- identificare chi si è messo in collegamento mediante l'attesa di un identificativo seriale;
- gestire il collegamento e le conseguenti richieste proprie del collegamento in maniera opportuna.

Per capire con chi “sta comunicando”, il ServerG una volta accettata una nuova connessione, si mette in attesa di ricevere un identificativo che può corrispondere al valore “1” se la nuova connessione proviene da un ClientS che ha bisogno di verificare la validità di un determinato Vanilla Green Pass, oppure al valore “2” se la nuova connessione proviene da un ClientT che ha bisogno di aggiornare lo stato di validità di un determinato Vanilla Green Pass.

Una volta che il ServerG ha identificato la tipologia di “pari della comunicazione” col quale si è messo in collegamento, lo stesso ServerG gestisce la richiesta in maniera adeguata. Si analizza prima il caso in cui l'identificativo ricevuto (che corrisponde sostanzialmente ad un “unsigned short int”) sia pari a “1”, ovvero la nuova connessione che è stata accettata, è stabilita con un ClientS. In tal caso il ServerG si mette subito in attesa di un codice di tessera sanitaria da parte del ClientS.

Una volta ricevuto quest'ultimo, il ServerG invia l'identificativo “1” al ServerV per fargli intendere che il collegamento che si sta effettuando è relativo a un ServerG che ha necessità di dare risposta ad un ClientS. A questo punto il ServerG invia il codice di tessera sanitaria ricevuto dal ClientS al ServerV e si mette in attesa di un pacchetto di livello applicazione da parte del ServerV corrispondente al pacchetto denominato “Pacchetto serverV_ReplyToServerG_clientS” visto prima. Infine, crea e invia un pacchetto illustrato qui di seguito denominato “Pacchetto serverG_ReplyToClientS” al ClientS collegato.

```
Pacchetto serverG_ReplyToClientS {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    unsigned short int requestResult  
}
```

In particolare, il codice di tessera sanitaria è quello relativo al Vanilla Green Pass di cui si vuole controllare la validità, mentre l'esito della richiesta è un valore booleano per stabilire se il Vanilla Green Pass eventualmente associato al codice di tessera sanitaria inviato, risulta essere attivo (valido sia per data che per stato) o meno. Una volta inviato tale pacchetto di livello applicazione, il ServerG rilascia le risorse acquisite e allocate per gestire questa richiesta ricevuta dal ClientS, chiude le relative connessioni aperte e passa ad attendere una successiva eventuale richiesta.

Si analizza prima il caso in cui l'identificativo ricevuto (che corrisponde sostanzialmente ad un "unsigned short int") sia pari a "2", ovvero la nuova connessione che è stata accettata, è stabilita con un ClientT. In tal caso il ServerG si mette subito in attesa di un pacchetto applicazione clientT_RequestToServerG parte del ClientS. La struttura del pacchetto poc'anzi menzionato è illustrata qui di seguito.

```
Pacchetto clientT_RequestToServerG {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    unsigned short int updateValue  
}
```

In particolare, il codice della tessera sanitaria è quello del cittadino del quale si vuole andare a modificare lo stato di attivazione del Vanilla Green Pass (se presente), mentre il valore di aggiornamento è il nuovo stato di validità da impostare a quel determinato Vanilla Green Pass. Tale valore è fornito dal ClientT. Una volta ricevuto quest'ultimo pacchetto, il ServerG invia l'identificativo "2" al ServerV per fargli intendere che il collegamento che si sta effettuando è relativo a un ServerG che ha necessità di dare risposta ad un ClientT.

A questo punto il ServerG si mette in comunicazione col ServerV inviando un pacchetto di livello applicazione visto prima, ovvero serverG_RequestToServerV_onBehalfOfClientT. Dopodiché il ServerG attende un pacchetto di livello applicazione visto prima, ovvero serverV_ReplyToServerG_clientT dal ServerV. Infine, crea e invia un pacchetto serverG_ReplyToClientT illustrato qui di seguito al ClientS collegato.

```
Pacchetto serverG_ReplyToClientT {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    unsigned short int updateResult  
}
```

In particolare, il codice di tessera sanitaria è quello relativo al Vanilla Green Pass che si vuole riattivare o invalidare, mentre l'esito dell'aggiornamento è un valore booleano per stabilire se l'aggiornamento dello stato di validità del Vanilla Green Pass eventualmente associato al codice di tessera sanitaria inviato, risulta essere andato a buon fine o meno. Una volta inviato tale pacchetto di livello applicazione, il ServerG rilascia le risorse acquisite e allocate per gestire questa richiesta ricevuta dal ClientT, chiude le relative connessioni aperte e passa ad attendere una successiva eventuale richiesta.

Il protocollo di livello applicazione di un ClientS prevede come prima operazione quella di collegarsi a un ServerG. A questo punto il ClientS invia il codice della tessera sanitaria al ServerG al quale si è collegato. Dopodiché si mette in attesa di una risposta da parte del ServerG che consisterà del pacchetto di livello applicazione serverG_ReplyToClientS analizzato precedentemente. Una volta arrivato tale pacchetto al ClientS da parte del ServerG, allora il protocollo di livello applicazione del ClientS termina e si procede con operazioni interne proprie del ClientS come la verifica dell'esito relativo al controllo richiesto, alcune stampe, il rilascio delle risorse allocate, la chiusura della connessione col ServerG e la terminazione. Per realizzare una nuova verifica di un Vanilla Green Pass dato, è necessario riavviare nuovamente il ClientS.

Il protocollo di livello applicazione di un ClientT prevede come prima operazione quella di collegarsi a un ServerG. A questo punto il ClientT invia un pacchetto al ServerG al quale si è collegato. Tale pacchetto consiste di un "Pacchetto clientT_RequestToServerG" ed è stato analizzato precedentemente. Dopodiché si mette in attesa di una risposta da parte del ServerG che consisterà nel pacchetto di livello applicazione serverG_ReplyToClientT anch'esso analizzato precedentemente. Una volta arrivato tale pacchetto al ClientT da parte del ServerG, allora il protocollo di livello applicazione del ClientT termina e si procede con operazioni interne proprie del ClientT come la verifica dell'esito dell'aggiornamento richiesto di un Vanilla Green Pass, alcune stampe, il rilascio delle risorse allocate, la chiusura della connessione col ServerG e la terminazione. Per realizzare una nuova verifica di un Vanilla Green Pass dato, è necessario riavviare nuovamente il ClientT.

Di seguito si allega uno schema riassuntivo utile a chiarire i protocolli di livello applicazione delle seguenti entità: ClientS, ServerG, ServerV (nel suo interfacciamento con un ServerG dovuto a un ClientS). Successivamente si allega un ulteriore schema riassuntivo utile a chiarire i protocolli di livello applicazione delle seguenti entità: ClientT, ServerG, ServerV (nel suo interfacciamento con un ServerG dovuto a un ClientT).

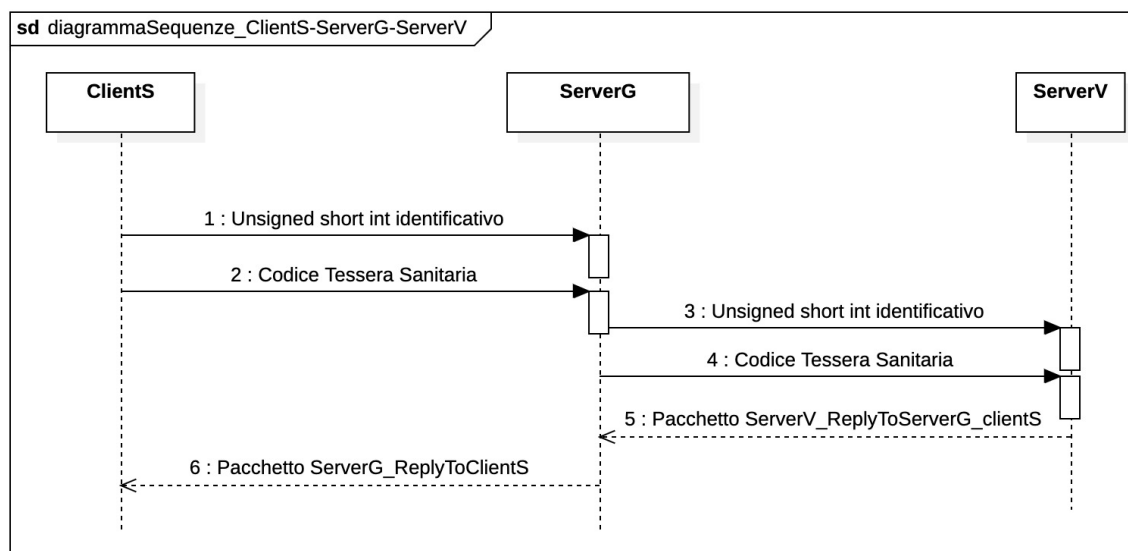


Figura 3.2: Diagramma delle sequenze ClientS-ServerG-ServerV

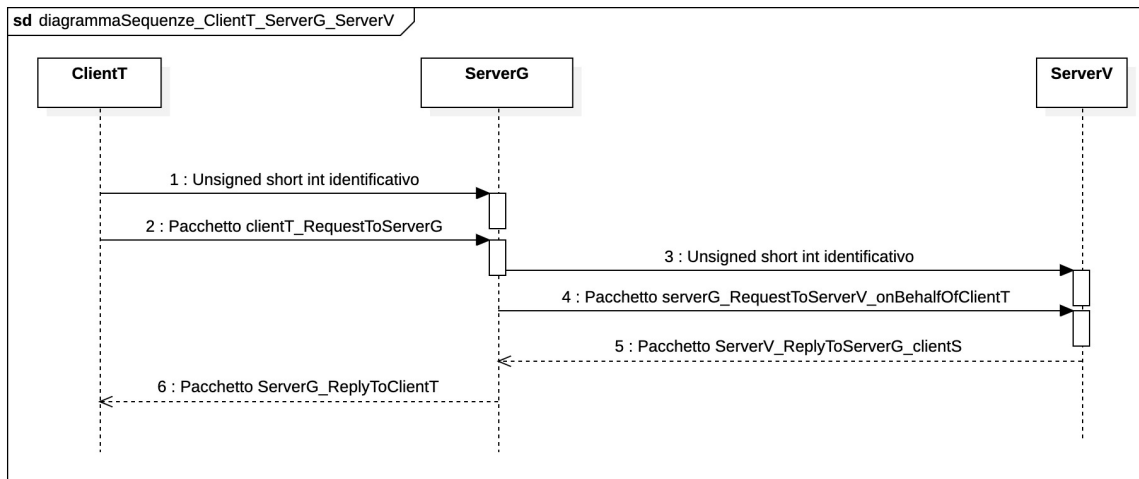


Figura 3.3: Diagramma delle sequenze ClientT-ServerG-ServerV

A questo punto, al fine di chiarire meglio il funzionamento di tutti i protocolli di livello applicazione utilizzati a vario titolo dalle varie entità realizzate, si allegano anche delle “istanze” dei diagrammi delle sequenze realizzati presenti in questo documento.

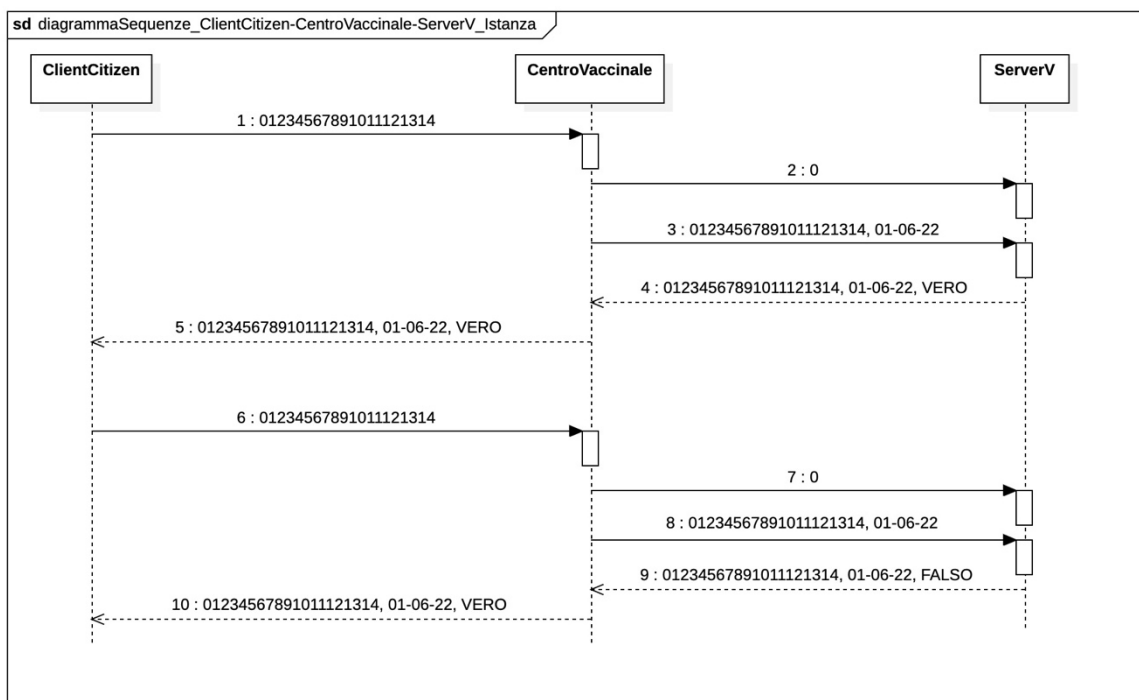


Figura 3.4: Istanza del diagramma delle sequenze ClientCitizen-CentroVaccinale-ServerV

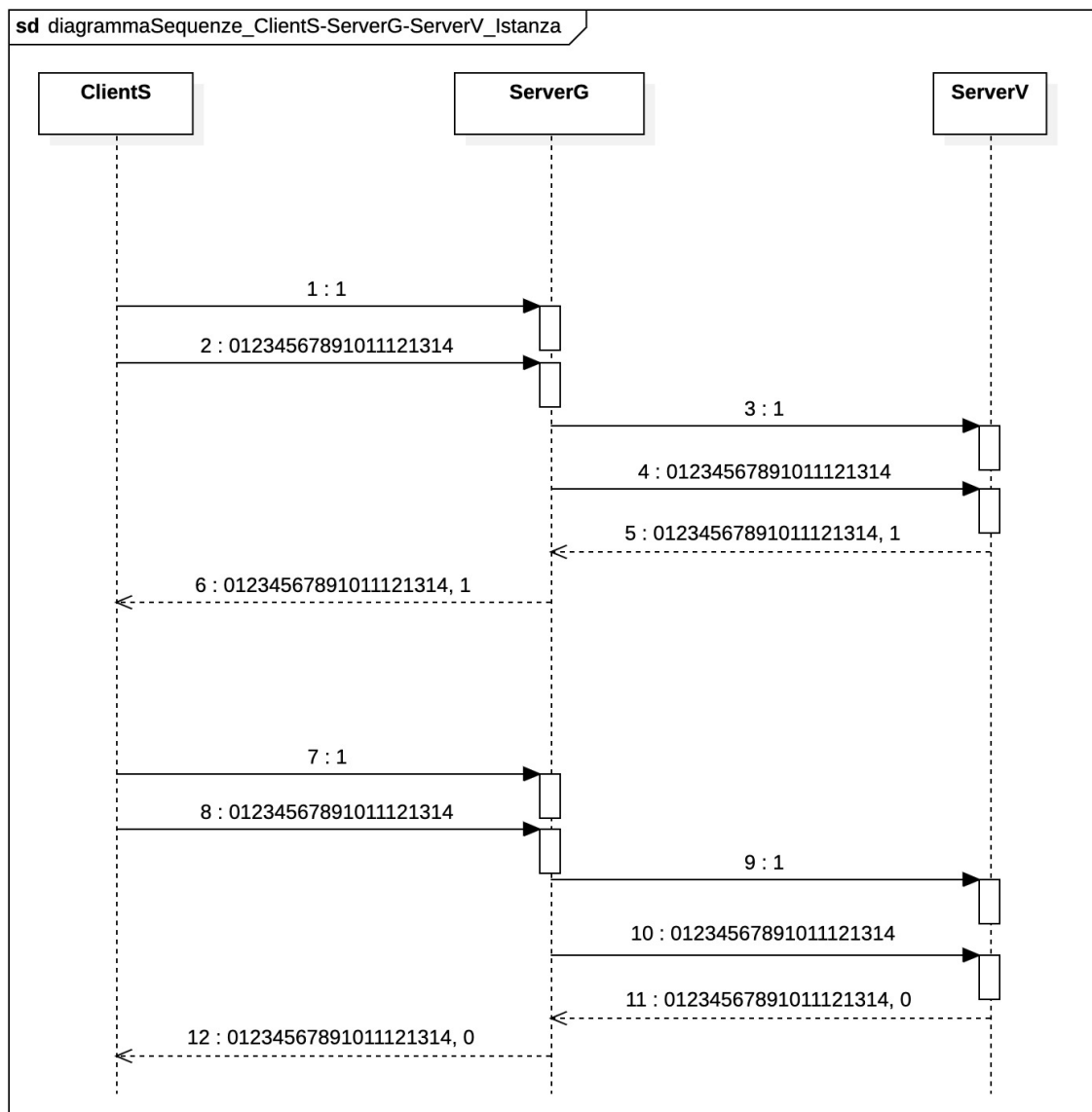


Figura 3.5: Istanza del diagramma delle sequenze ClientS-ServerG-ServerV

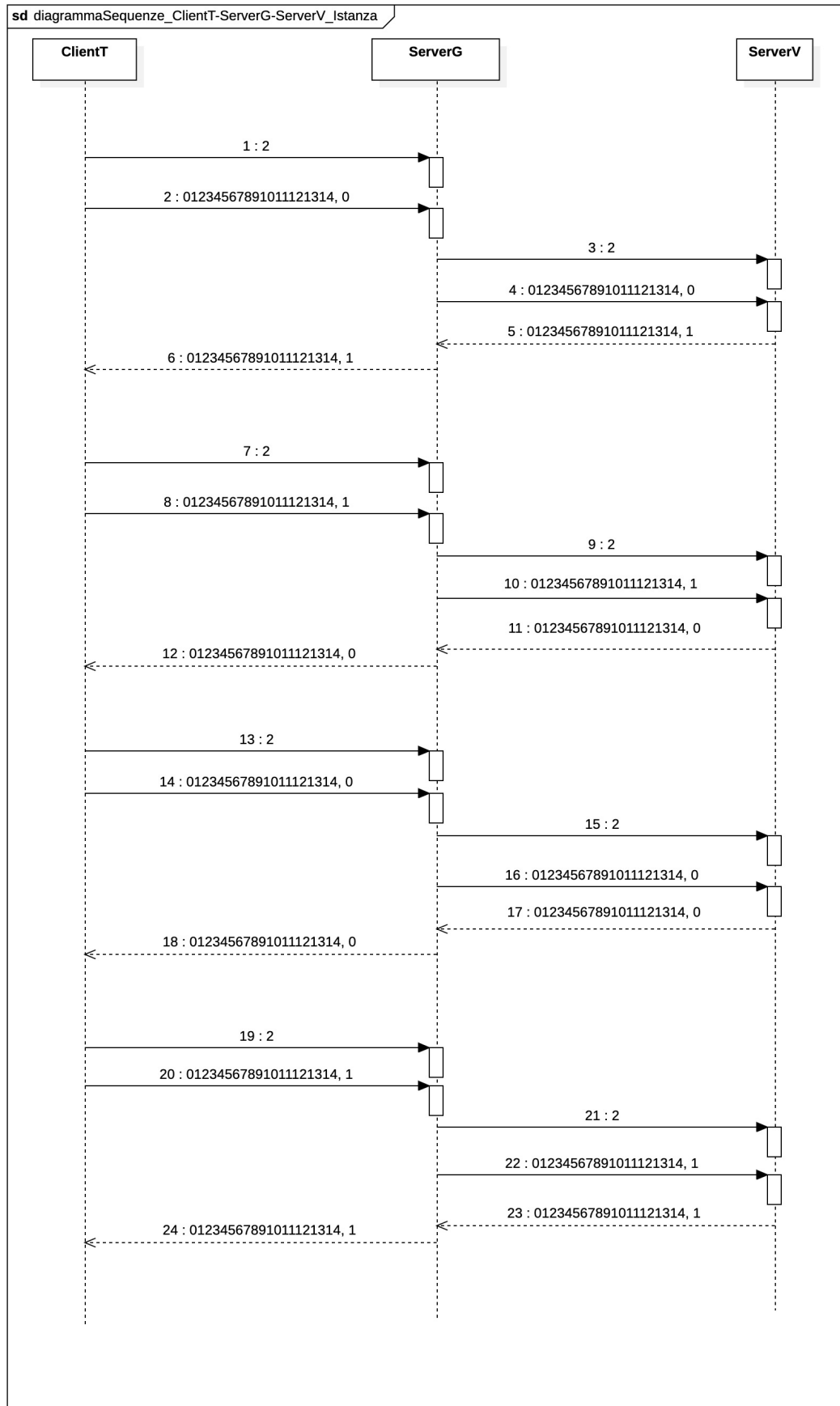


Figura 3.6: Istanza del diagramma delle sequenze ClientT-ServerG-ServerV

Al fine di chiarire meglio quali sono i differenti pacchetti di livello applicazione utilizzati a vario titolo dalle entità realizzate, si riportano qui di seguito tutti i pacchetti con la loro struttura in memoria.

```
Pacchetto centroVaccinaleReplyToClientCitizen {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    char greenPassExpirationDate[DATE_LENGTH]  
    unsigned short int requestResult  
}
```

```
Pacchetto centroVaccinaleRequestToServerV {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    char greenPassExpirationDate[DATE_LENGTH]  
}
```

```
Pacchetto serverV_ReplyToCentroVaccinale {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    char greenPassExpirationDate[DATE_LENGTH]  
    unsigned short int requestResult  
}
```

```
Pacchetto serverV_ReplyToServerG_clientS {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    unsigned short int requestResult  
}
```

```
Pacchetto serverG_RequestToServerV_onBehalfOfClientT {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    unsigned short int updateValue  
}
```

```
Pacchetto serverV_ReplyToServerG_clientT {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    unsigned short int updateResult  
}
```

```
Pacchetto serverG_ReplyToClientS {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH];  
    unsigned short int requestResult  
}
```

```
Pacchetto clientT_RequestToServerG {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    unsigned short int updateValue  
}
```

```
Pacchetto serverG_ReplyToClientT {  
    char healthCardNumber[HEALTH_CARD_NUMBER_LENGTH]  
    unsigned short int updateResult  
}
```

Dettagli implementativi comuni

4 - Dettagli implementativi comuni

Tutte le entità software individuate sono state realizzate in linguaggio C. Gli standard di riferimento sono POSIX e ISO C99. I sorgenti sono stati organizzati in file header (.h) e file d'implementazione (.c). In questa sezione si mettono in luce alcuni aspetti, funzioni e librerie condivise sia dai client che dai server individuati e realizzati. In particolare, la maggior parte del codice utilizza molto spesso funzioni e procedure presenti nelle seguenti librerie scritte appositamente da uno dei proponenti e analizzate dall'altro durante il corso di "Reti di Calcolatori e Laboratorio di Calcolatori". In particolare "UsageUtility.h" è una libreria che contiene e implementa delle funzioni e delle procedure di "utility" come ben fa intuire il nome. Vi è la procedura "checkUsage(...)" per controllare gli argomenti con i quali vengono lanciati gli eseguibili, la "raiseError(...)" e la "threadRaiseError(...)" rispettivamente per "segnalare e terminare il programma" con un errore in un processo e in un thread. Poi vi sono le classiche "fullRead(...)" e "fullWrite(...)" che hanno il compito di completare letture e scritture anche nel caso in cui la variabile "errno" sia posta ad "EINTR". Per maggiori informazioni su "fullRead(...)" e "fullWrite(...)" si consulti "GaPiL – Guida alla Programmazione in Linux, Simone Piccardi". Inoltre, la libreria "UsageUtility.h" include tutte le librerie necessarie come stdio.h, pthread.h, stdlib.h, unistd.h, e così via. Per una visione completa di tutte le librerie importate in "UsageUtility.h" si consulti il relativo sorgente. Infine, "UsageUtility.h" definisce la maggior parte dei codici di errori e di stringhe ad essi associati, utili per la stampa dettagliata dell'errore avvenuto e definisce l'enumerazione per generare il "tipo" booleano simulato.

Un'altra libreria comune ai server e ai client individuati è "NetWrapper.h". Questa libreria, come si intuisce dal nome, contiene tutte le funzioni wrapper per la gestione dei socket e della connessione quali "socket(...)", "listen(...)", "bind(...)", "accept(...)", "connect(...)", "close(...)". Infine, è presente una procedura per il controllo della validità di un indirizzo IPv4, a partire da una stringa data. All'interno di questa libreria sono presenti ulteriori codici di errore propri delle funzioni e procedure "di rete", da cui il nome della libreria "NetWrapper.h". Sia la libreria "UsageUtility.h" che la libreria "NetWrapper.h" vedono la loro implementazione all'interno dei rispettivi file ".c": "UsageUtility.c" e "NetWrapper.c".

Un'ultima libreria comune ai server e ai client individuati è "GreenPassUtility.h" che vede la propria implementazione in "GreenPassUtility.c". Questa libreria fornisce alcune funzioni e procedure fondamentali come "checkHealtCardNumber(...)" per controllare che una data stringa fornita che rappresenti un codice di tessera sanitaria, rispetti il formato previsto; "retrieveConfigurationData(...)" per ricavare a partire da un determinato file di configurazione, i parametri fondamentali per contattare un preciso server (o comunque entità) all'interno dell'architettura finora presa in considerazione; "getVaccineExpirationDate(...)" per calcolare la data di scadenza del vaccino e di conseguenza del Vanilla Green Pass ad esso associato in formato stringa; "getNowDate(...)" per ottenere la data di sistema attuale in formato stringa; "createConnectionWithServerV(...)" per ottenere un descrittore di socket al fine di mettersi in comunicazione con il ServerV.

Inoltre, "GreenPassUtility.h" fornisce anche la definizione di tutti i pacchetti di livello applicazione utilizzati e fin qui presi in considerazione e un "enumerazione" importante che elenca i vari tipi di mittenti con cui il ServerV e il ServerG possono ritrovarsi in collegamento. Infine, definisce utili codici di errori, stringhe ad essi associate e costanti proprie del contesto "GreenPass", vaccinazione, etc.

Di seguito si allega uno schema utile al fine di capire quali sorgenti dell'architettura includono quali librerie. Vengono omesse le librerie di sistema.

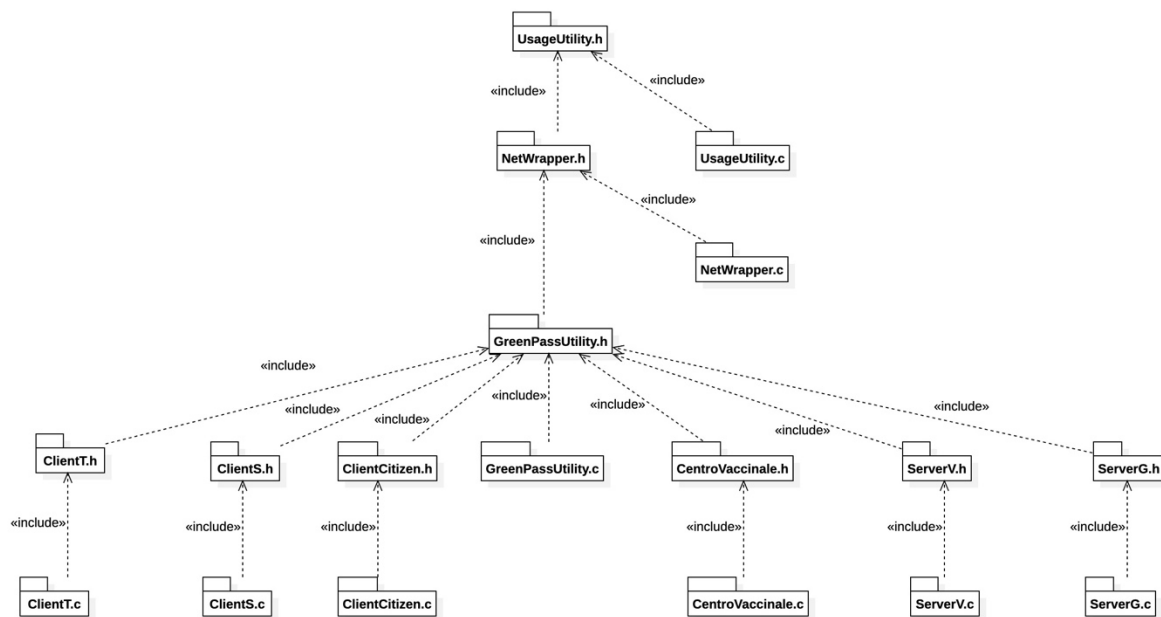


Figura 4.1: Dipendenze sorgenti e librerie Vanilla Green Pass

Si definisce ora il formato dei file di configurazione e il formato dei file contenenti i dati dei Vanilla Green Pass. Per quanto riguarda un generico file di configurazione, è costituito da tre righe:

- la prima riga contiene l'IP dell'host al quale l'entità il cui nome è dato dal nome del file deve connettersi;
- la seconda riga contiene il numero di porta alla quale l'entità il cui nome è dato dal nome del file deve connettersi;
- la terza riga è vuota ed è ottenuta a partire dal carattere new line utilizzato nella seconda riga.

Un esempio del contenuto del file di configurazione è il seguente:

```
127.0.0.1
20000
```

Per quanto riguarda il file contenente i dati dei Vanilla Green Pass, è strutturato nel seguente modo. Per ogni riga si riporta una tripla che rappresenta un Vanilla Green Pass in cui ogni componente della tripla è separato dall'altra componente mediante il carattere ":". Le componenti della tripla sono le seguenti:

- codice della tessera sanitaria;
- data di scadenza del Vanilla Green Pass (DD-MM-YYYY);
- stato di validità del Vanilla Green Pass (0/1).

Un esempio del contenuto del file che conserva i dati dei Vanilla Green Pass è il seguente:

```
...
000000000000000000000004:01-06-2022:1
000000000000000000000003:01-06-2022:1
00000000000000000000001000:01-06-2022:1
00000000000000000000001001:01-06-2022:1
000000000000000000000002:01-06-2002:1
000000000000000000000001:01-06-2022:1
000000000000000000000000:01-06-2002:0
...
```

Si precisa che nei due capitoli che seguono, rispettivamente “Dettagli implementativi dei client” e “Dettagli implementativi dei server”, verranno descritti nel primo dei due capitoli il ClientCitizen, il ClientT e il ClientS approfondendo quello che è il loro funzionamento e la parte implementativa loro riguardante. Mentre, nel secondo dei due capitoli menzionati poc’anzi, verranno presi in considerazione il CentroVaccinale, il ServerG e il ServerV.

Dettagli implementativi dei client

5 - Dettagli implementativi dei client

Temp text

Dettagli implementativi dei server

6 - Dettagli implementativi dei server

Si inizia la trattazione di questo capitolo considerando un generico CentroVaccinale e le peculiarità della sua implementazione. CentroVaccinale sebbene sia inserito in questa categoria relativa ai server, svolge anche un ruolo da client nei confronti del ServerV, così come il ServerG svolge anche un ruolo da client nei confronti del medesimo ServerV.

Si prenda in considerazione la funzione “main(...)” di CentroVaccinale. Inizialmente si dichiarano alcune variabili che saranno utili nel corso di vita dell’entità software. Si dichiarano tre descrittori: uno per la connessione con il ServerV, uno per ascoltare le connessioni in entrata dei vari ClientCitizen e un altro per gestire la singola connessione con un singolo ClientCitizen. Poi sono presenti due strutture “sockaddr_in” una da “riempire” con le informazioni del CentroVaccinale in quanto server, e una che verrà riempita dalla “waccept(...)” nel momento in cui vi è un nuovo ClientCitizen collegato. Inoltre, si dichiara una variabile “unsigned short int” per memorizzare la porta sulla quale chi avvia l’entità sceglie di avviare il CentroVaccinale e una variabile “pid_t” per gestire i processi figli di un ClientCitizen che andranno ad occuparsi di una singola richiesta di un ClientCitizen. Il CentroVaccinale rappresenta quindi un server multi-processo, tale per cui, al fine di soddisfare una nuova richiesta di un ClientCitizen pervenuta, crea un processo figlio dedicato e la gestisce in maniera opportuna.

Prima di passare a tale aspetto però c’è da dire che il CentroVaccinale invoca “checkUsage(...)” al fine di verificare che è stato avviato con i parametri che si aspetta di avere. Dopodiché si cerca di ricavare il numero di porta a partire dal valore passato come argomento all’avvio di CentroVaccinale via terminale. A questo punto si utilizza la “setsockopt(...)” per impostare l’opzione di riutilizzo degli indirizzi durante l’applicazione del meccanismo di interprocess communication via socket, si puliscono le variabili che andranno a contenere la struttura “sockaddr_in” del CentroVaccinale e dei ClientCitizen, si “compila” la struttura “sockaddr_in” del CentroVaccinale per erogare il servizio server del CentroVaccinale, dopodiché si effettua una chiamata a funzione wrapper di “bind(...)” e poi “listen(...)”.

Si entra ora nel vivo del CentroVaccinale, avviando un ciclo infinito che svolge le seguenti operazioni.

1. Ottieni il socket file descriptor col quale comunicare con il ClientCitizen che si è appena collegato a partire dal socket file descriptor “di ascolto” sul quale il CentroVaccinale si era per l’appunto messo in ascolto e riempi la struttura “sockaddr_in” del client con le informazioni relative al “ClientCitizen” collegatosi.
2. Crea un processo figlio dedicato mediante “fork()” che, come prima operazione, chiuderà il file descriptor relativo “all’ascolto” delle nuove connessioni in arrivo per il CentroVaccinale, realizza un collegamento con il ServerV all’interno del processo figlio generato e invoca la routine per gestire la richiesta del ClientCitizen collegatosi. Una volta che il processo figlio ha terminato l’esecuzione di quest’ultima routine, terminerà.
3. Intanto, il processo padre chiude il socket file descriptor che realizza la connessione con il ClientCitizen collegatosi e si rimette in ascolto di nuove connessioni e qualora dovesse arrivarne una nuova, si passa nuovamente al punto 1.

Si noti come il codice seguente questo ciclo infinito, non verrà mai eseguito. Inoltre, il CentroVaccinale può terminare la sua esecuzione nel momento in cui si verifichi un errore che causi la terminazione, ovvero nel momento in cui venga invocata la procedura “raiseError(...)” dal processo padre. Si considera ora nel dettaglio le azioni svolte dal processo figlio generato. La funzione invocata per mettersi in collegamento con il ServerV ha il seguente prototipo “int createConnectionWithServerV(char * configFilePathCentroVaccinale)” e il suo scopo è quello di “ritornare” il socket file descriptor di una connessione impostata con il ServerV. Per fare ciò però c’è bisogno di fare il “retrieve” dei dati di contatto che sono stati forniti al CentroVaccinale per mettersi in contatto con il ServerV. Tali dati si trovano all’interno del rispettivo file di configurazione “centroVaccinale.conf”. Poi è necessario effettuare il classico “setup” della struttura “sockaddr_in”, invocare un’opportuna “socket(...)”, una “inet_pton(...)”, una “connect(...)”, per non parlare della gestione della memoria dinamica annessa a tali operazioni. Quindi siccome sono operazioni laboriose e svolte anche dal ServerG, si è realizzato tale funzione che si occupa di fare tutto ciò.

A questo punto il processo figlio invoca la routine di gestione della richiesta del ClientCitizen. Per capire cosa e in che ordine viene scambiato si consultì il capitolo 3 di questo documento. Per quanto riguarda il lato implementativo, si utilizza la memoria dinamica per allocare i pacchetti da inviare al ClientCitizen, da inviare al ServerV e quello da ricevere dal ServerV. Anche in questo caso l’error-handling fa da padrone e accompagna, così come in tutte le altre operazioni di tutte le altre entità software realizzate, il flusso del programma stesso. Si usa un buffer per memorizzare il codice di tessera sanitaria dell’utente e si imposta l’identificativo del CentroVaccinale, che sarà poi utilizzato per farsi identificare dal ServerV. La comunicazione tra le varie entità software coinvolte avviene mediante “fullRead(...)” e “fullWrite(...)”. Mentre le assegnazioni e/o copie di stringhe sono effettuate mediante “strncpy(...)” ben controllate in modo tale che date le debolezze di tale funzione, non generi problemi all’esecuzione del programma. Il periodo di validità che il CentroVaccinale è incaricato di inviare al ServerV, viene ottenuta grazie alla funzione “getVaccineExpirationDate()” che prende la data di sistema attuale, la tronca al primo del mese corrente, aggiunge sei mesi verificando l’eventuale nuovo anno, la converte sottoforma di stringa (char *) e la “ritorna” al chiamante. Infine, la routine di gestione della richiesta del ClientCitizen, genera la risposta per il ClientCitizen e la invia ad esso. Dopodiché si libera tutta la memoria dinamica allocata per la gestione con opportune “free(...)” e successive chiusure dei socket file descriptor ormai inutilizzati.

Si allega la gestione server all’interno del main di CentroVaccinale:

```
while (TRUE) {
    socklen_t clientAddressLength = (socklen_t) sizeof(client);
    connectionFileDescriptor = waccept(listenFileDescriptor, (struct sockaddr *) & client, (socklen_t *) &
                                      clientAddressLength);

    if ((childPid = fork()) == -1) {
        raiseError(FORK_SCOPE, FORK_ERROR);
    } else if (childPid == 0) {
        wclose(listenFileDescriptor);
        serverV_SocketFileDescriptor = createConnectionWithServerV(configFilePathCentroVaccinale);
        clientCitizenRequestHandler(connectionFileDescriptor, serverV_SocketFileDescriptor);
        exit(0);
    }
    wclose(connectionFileDescriptor);
}
```

Si prosegue la trattazione con l'analisi dell'implementazione del ServerG. Le differenze tra l'implementazione e realizzazione del ServerG rispetto al CentroVaccinale sono minime. Ciò che cambia è il contenuto del ciclo infinito tipico del server e le routine invocate per la gestione delle richieste che in questo caso possono pervenire dai ClientS e dai ClientT. La parte precedente al ciclo infinito resta concettualmente quasi del tutto invariata rispetto al CentroVaccinale, a meno degli adattamenti dovuti alla differente entità software. Una volta entrati nel ciclo infinito, il ServerG accetta una nuova connessione, si mette in ascolto dell'identificativo del mittente col quale si è messo in collegamento e crea un processo figlio all'interno del quale si determina chi sia il mittente e si invoca l'opportuna routine di gestione della richiesta, dopo aver chiuso il socket file descriptor per l'ascolto e dopo aver instaurato una connessione col ServerV.

Se il collegamento è stato instaurato con un ClientS, allora il processo figlio invoca la procedura "clientS_RequestHandler(...)", mentre se il collegamento è stato instaurato con un ClientT, allora il processo figlio invoca la procedura "clientT_RequestHandler(...)". Le due procedure non differiscono molto tra di loro se non per il diverso protocollo applicazione utilizzato descritto nel capitolo 3 di questo documento e la gestione di più o meno blocchi di memoria dinamica. Entrambe le procedure prendono in input due socket file descriptor: connectionFileDescriptor e il serverV_SocketFileDescriptor. Il primo è il socket file descriptor col quale è possibile comunicare con un ClientS o un ClientT, mentre il secondo è il socket file descriptor col quale è possibile comunicare con il ServerV.

Si precisa che anche ServerG per la parte del server, è implementato come un server multi-processo. Si allega la gestione server all'interno del main di ServerG:

```
while (TRUE) {
    ssize_t fullReadReturnValue;
    socklen_t clientAddressLength = (socklen_t) sizeof(client);
    connectionFileDescriptor = waccept(listenFileDescriptor, (struct sockaddr *) & client, (socklen_t *) &
                                      clientAddressLength);
    if ((fullReadReturnValue = fullRead(connectionFileDescriptor, (void *) & requestIdentifier,
                                       sizeof(requestIdentifier))) != 0) raiseError(FULL_READ_SCOPE, (int) fullReadReturnValue);

    if ((childPid = fork()) == -1) {
        raiseError(FORK_SCOPE, FORK_ERROR);
    } else if (childPid == 0) {
        wclose(listenFileDescriptor);
        serverV_SocketFileDescriptor = createConnectionWithServerV(configFilePathServerG);

        switch (requestIdentifier) {
            case clientS_viaServerG_Sender:
                clientS_RequestHandler(connectionFileDescriptor, serverV_SocketFileDescriptor);
                break;
            case clientT_viaServerG_Sender:
                clientT_RequestHandler(connectionFileDescriptor, serverV_SocketFileDescriptor);
                break;
            default:
                raiseError(INVALID_SENDER_ID_SCOPE, INVALID_SENDER_ID_ERROR);
                break;
        }

        wclose(connectionFileDescriptor);
        wclose(serverV_SocketFileDescriptor);
        exit(0);
    }
    wclose(connectionFileDescriptor);
}
```

Si prosegue la trattazione con l'analisi dell'implementazione del ServerV. Si tratta di un'entità che svolge soltanto il ruolo di server ed è implementata come un server multi-thread, cioè per ogni nuova richiesta pervenuta da un CentroVaccinale oppure da un ServerG (che a sua volta può ricevere richiesta da un ClientS o da un ClientT), si crea un thread apposito che permette di gestire la richiesta in maniera efficiente. Un thread in base a "Dhananjay M. Dhamdhare - Sistemi Operativi" è un flusso di esecuzione di un programma che usa le risorse di un processo. Un processo al proprio interno può avere più thread e quindi più flussi di esecuzione. Un server multi-thread permette di avere i seguenti vantaggi.

- Minore overhead dovuto alla creazione e alla commutazione: lo stato del thread consiste solo dello stato dell'elaborazione. Lo stato dell'allocazione delle risorse e lo stato delle comunicazioni non sono parte dello stato del thread, per cui la creazione e la commutazione dei thread produce un overhead inferiore.
- Comunicazione più efficiente: i thread di un processo possono comunicare tra loro attraverso dati condivisi, evitando in questo modo l'overhead di comunicazione dovuto alle chiamate di sistema.
- Progettazione semplificata: l'uso dei thread può semplificare la progettazione e la codifica delle applicazioni che servono le richieste concorrentemente.

La gestione di un server-multithread comporta anche degli svantaggi come la gestione delle "race condition", eventuali problematiche relative alla sicurezza dovute alla condivisione del comparto dati e così via. I punti chiave in base ai quali per il ServerV si è scelto un server-multithread anziché un server multi-processo come è accaduto nel caso di ServerG e CentroVaccinale, sono le seguenti.

- Il ServerV rappresenta un SPOF per l'architettura realizzata ed è soggetto potenzialmente a un grandissimo quantitativo di richieste da parte dei ServerG e da parte dei CentriVaccinali. Ogni richiesta ricevuta, richiede delle risorse per essere gestita e di conseguenza man mano che arrivano centinaia se non migliaia di richieste, avere dei thread anziché dei processi permette di risparmiare le risorse eventualmente impiegate per la creazione dei processi a favore di un maggior carico di richieste supportabile.
- Si è supposto che il ServerV sia monoprocesso e non multiprocesso. In quest'ultimo caso sarebbe stato altrettanto valida la scelta di utilizzare una soluzione multi-processo eventualmente con l'applicazione del calcolo parallelo.
- Scopo didattico e sperimentale: al fine di combinare le conoscenze acquisite precedentemente nel corso di laurea, esplorando differenti alternative per realizzare una medesima entità software.
- Si suppone che un ServerG e un CentroVaccinale possano essere soggetti a meno richieste in quanto presenti in numero maggiore rispetto ad un singolo ServerV. In tal caso si adotta un server multi-processo in quanto non vi è la problematica delle risorse limitate. Inoltre, sono maggiormente robusti in termini di sicurezza dei server multi-processo dal momento che rappresentano, nelle ipotesi dei progettisti, dei bersagli più esposti per eventuali attaccanti che potrebbero finire per corrompere i dati condivisi di tutti i thread se CentroVaccinale e ServerG fossero implementati come server multi-thread, anziché come server multi-processo. Inoltre, supponendo che il ServerV sia un bersaglio meno esposto e con l'aggiunta futura di tecniche crittografiche e serializzazione, è come se si creasse un "tunneling" sicuro di comunicazione tra le varie entità considerate "server" all'interno di questo schema architetturale.

Si analizza ora il codice implementativo di ServerV. Si omette il commento delle operazioni iniziali di “setup” del ServerV in quanto sono banali e molto simili, con i dovuti adattamenti, a quelle già viste per il ServerG e per il CentroVaccinale. Si passa quindi all’inizializzazione dei mutex e dei loro attributi. Vi è un mutex su un puntatore a intero che punta a un intero rappresentante il socket file descriptor che il singolo thread userà per mettersi in collegamento con il ServerG o con il CentroVaccinale. Tale variabile è soggetta a race condition in quanto più thread potrebbero accedere alla stessa variabile e comunicare con un’entità di rete non inaspettata. Per questo motivo i thread dereferenziano il puntatore ottenendo il valore effettivo dopo aver acquisito il mutex corrispettivo, lo rilasciano e proseguono senza problemi. L’altro mutex utilizzato è quello per accedere in mutua esclusione per operazioni di lettura e scrittura al file “serverV.dat” memorizzato sul file system.

A questo punto si entra nel ciclo infinito del ServerV, si accetta una connessione, si identifica chi si è messo in collegamento (se un CentroVaccinale, un ServerG a causa di un ClientT o un ServerG a causa di un ClientS), si acquisisce il mutex sul file descriptor di connessione e lo si duplica in un altro file descriptor, in quanto successivamente quello originale di connessione verrà chiuso dal thread main e riutilizzato per altri thread, si rilascia il mutex e si individua la routine apposita da invocare a seconda del mittente della richiesta al ServerV. Dopodiché si chiude il socket file descriptor relativo a quella connessione nel thread main e il flusso principale del ServerV si rimette in attesa di una nuova connessione. Si noti anche in questo caso, come il codice successivo al ciclo infinito non venga mai eseguito. Una volta individuata quale routine invocare tramite il costrutto di “switch-case”, si crea un thread apposito mediante “pthread_create(...)” e passando come argomento il socket file descriptor precedentemente duplicato nella variabile “threadConnectionFileDescriptor”. Inoltre, i thread sono caratterizzati dall’essere “detached” ovvero non è possibile poi effettuare “join” con il flusso principale. Questo perché vengono visti come elementi a sè stanti e autonomi che sono creati come thread solo al fine di raggiungere gli obiettivi prima proposti. I thread creati si occuperanno di gestire le risorse eventualmente acquisite in maniera oculata e di liberarle prima di terminare con la “pthread_exit(...)”. Si noti come la variabile per l’identificazione del “tid” del thread è sempre la stessa. Solitamente si usa un array di tid al fine di effettuare la join di tutti quanti i thread successivamente, o di particolari gestioni individuali dei thread identificati mediante i singoli tid. Nel caso del ServerV questa cosa non è necessaria, tant’è che vengono creati come detached thread.

A questo punto si considera prima la procedura “threadAbort(...)” e poi le restanti routine che rappresentano l’esecuzione dei singoli thread generati di volta in volta. La procedura “threadAbort(...)” ha il compito di liberare tutte le risorse allocate mediante memoria dinamica che gli vengono passate tramite una lista di puntatori a “void” di lunghezza variabile. Si tratta infatti di una “variadic function” letteralmente. Inoltre, chiude il socket file descriptor di connessione usato dal thread per comunicare col ServerG o con il CentroVaccinale e invoca l’errore il cui numero e la cui stringa associata da stampare sono passate tramite parametro, così come i puntatori a blocchi dinamici di memoria e il socket file descriptor di connessione.

Si analizza ora il funzionamento delle tre funzioni invocate per soddisfare le richieste dei CentriVaccinali e dei ServerG. La struttura e il significato delle operazioni è simile tra le varie funzioni. In prima analisi si considera “centroVaccinaleRequestHandler(...)”.

La prima operazione che viene effettuata è l'acquisizione del mutex al fine di andare a dereferenziare in mutua esclusione il valore contenuto nel puntatore passato come argomento alla routine assegnata al thread. Si rilascia il mutex, si dichiarano alcune variabili iniziali che verranno prese in considerazione man mano durante l'analisi e si alloca lo spazio necessario in memoria per i pacchetti di livello applicazione necessari. A questo punto si attende un pacchetto "centroVaccinaleRequestToServerV", e si costruisce subito parte della risposta tramite il pacchetto "serverV_ReplyToCentroVaccinale". A questo punto si aprono (e se non esistono si creano anche) i file "serverV.dat" e "tempServerV.dat", il primo in sola lettura mentre il secondo in sola scrittura. A questo punto si effettua il controllo secondo il quale siano passati o meno cinque mesi dall'ultimo vaccino effettuato e cioè si controlla se il Vanilla Green Pass risulta essere scaduto o meno. Questa operazione è un po' inefficiente avendo implementato il tutto mediante "file" memorizzato su file system. Si passa quindi alla lettura sequenziale riga per riga del file fin quando non si trova la riga che contiene il codice di tessera sanitaria inviato dal CentroVaccinale. Nel caso peggiore la complessità è lineare in quanto si deve scorrere tutto il file.

Supponiamo prima che il codice di tessera sanitaria sia presente nel file in una delle sue righe. A questo punto si preleva la data di scadenza del Vanilla Green Pass associato e la si confronta con la data attuale di sistema ottenuta mediante "getNowDate()" una volta che entrambe le date sono state convertite secondo la "struct tm". Il confronto tra le due date avviene tramite la funzione "difftime(...)" che restituisce il numero di secondi risultanti dall'operazione "data_1" - "data_2" dove "data_1" e "data_2" sono rispettivamente il primo e il secondo parametro passati alla funzione. Si divide il numero di secondi in modo da ottenere i mesi e si verifica se siano passati meno di cinque mesi e che sia passato un numero di mesi maggiori di zero (il valore di ritorno di "difftime(...)" è un double). Questo perché qualora i mesi passati siano troppi, il valore di ritorno può essere addirittura negativo; quindi, è necessario questo ulteriore controllo per correttezza. In ogni caso, qualora le due condizioni siano rispettate il flag di "isVaccineBlocked" viene attivato, la data estratta dal file viene posizionata nel pacchetto di risposta e non sarà possibile successivamente entrare nel "branch condizionale" che permette poi di confermare la possibilità di procedere con la vaccinazione al CentroVaccinale. Inoltre, se il codice di tessera sanitaria viene trovato nel file, allora si termina l'esecuzione del ciclo appena dopo l'esecuzione delle operazioni poc'anzi menzionate e si passa alle prossime istruzioni. Se invece il codice di tessera sanitaria non è presente nel file, il flag "isVaccineBlocked" è posta ancora a FALSE e quindi si entra nel blocco condizionale successivo.

Le prossime istruzioni sono le seguenti: se il vaccino non è stato bloccato dal controllo precedente, allora la data di scadenza calcolata dal CentroVaccinale va bene e viene posta nel pacchetto di risposta; si chiude e si riapre il file "serverV.dat" al fine di riposizionarsi all'inizio del file; si copia l'intero contenuto del file "serverV.dat" all'interno del file "tempServerV.dat" tranne che per la riga contenente il Vanilla Green Pass del cittadino il cui codice di tessera sanitaria è stato inviato dal CentroVaccinale e infine si inserisce all'interno del "tempServerV.dat" la nuova riga relativa al cittadino che ha fatto richiesta di vaccinarsi e la cui richiesta è stata accettata. Tale riga rispetta il formato dei dati all'interno del file "serverV.dat". A questo punto si chiudono entrambi i file, si cancella il file "serverV.dat", si rinomina il file "tempServerV.dat" con il nuovo nome "serverV.dat" e si crea un nuovo file "tempServerV.dat". Qualora invece il vaccino sia stato bloccato si chiudono soltanto i due file aperti.

A questo punto se il codice di tessera sanitaria è stato trovato all'interno del primo controllo di cui si è discusso prima all'interno del file "serverV.dat", allora si rilascia il mutex acquisito e si invia il pacchetto serverV_ReplyToCentroVaccinale al CentroVaccinale liberando due risorse in più (allocate precedentemente) rispetto all'altro ramo della condizione. Infine, si liberano le risorse allocate, il socket file descriptor di connessione e il thread termina.

In seconda analisi si considera “clientS_viaServerG_RequestHandler (...)”. La struttura è molto simile a quella della precedente funzione presa in considerazione. Ragion per cui si sorvola la parte iniziale dichiarativa e si passa alle peculiarità di questa funzione. L’idea è quella di scorrere il file “serverV.dat” come nel caso precedente finché non si trova il codice di tessera sanitaria inviato dal ServerG. Qualora venisse trovato, si estrae la data di scadenza associata, la si confronta con la data di sistema attuale e si verifica se vi è una validità in termini di data del Vanilla Green Pass trovato. Poi si estrae lo stato di validità del Vanilla Green Pass nella stessa riga e si verifica se vi è anche una validità in termini di “stato di attivazion” del Vanilla Green Pass. In basi a questi due aspetti si impostano le variabili “isGreenPassExpired” e “isGreenPassValid” rispettivamente. Se invece il Vanilla Green Pass non è stato trovato, automaticamente la risposta alla richiesta di verifica di validità darà esito negativo. Successivamente se il Vanilla Green Pass è stato trovato nel file “serverV.dat”, se il Vanilla Green Pass non è scaduto ed è attivo, allora si imposterà il campo di esito della richiesta del ServerG a TRUE. Dopodiché in entrambi i rami della condizione si rilascia il mutex acquisito per l’accesso ai file su file system e si invia il pacchetto al ServerG con una “fullWrite(...)”. Infine, si liberano le risorse acquisite.

In ultima analisi si considera “clientT_viaServerG_RequestHandler (...)”. La struttura è molto simile a quelle delle precedenti funzioni prese in considerazione. Ragion per cui si sorvola la parte iniziale dichiarativa e si passa alle peculiarità di questa funzione. L’idea è quella di scorrere il file “serverV.dat” come nel caso precedente finché non si trova il codice di tessera sanitaria inviato dal ServerG. Qualora venisse trovato, si estrae la data di scadenza associata del Vanilla Green Pass trovato e la si salva in una variabile locale. Se invece, il Vanilla Green Pass non viene trovato allora il flag “healthCardNumberWasFound” resterà a FALSE, così come nelle funzioni precedenti. A questo punto si chiude il file “serverV.dat”, se è stato trovato il Vanilla Green Pass nel file “serverV.dat” allora si riapre il file “serverV.dat” e si apre il file “tempServerV.dat”, si copia tutto il contenuto del primo dei due nel secondo dei due, tranne che per la riga che contiene le informazioni relative al codice di tessera sanitaria inviato dal ServerG e si inserisce nel file “tempServerV.dat” un ulteriore riga con codice di tessera sanitaria pari a quello inviato dal ServerG, con data di scadenza quella ricavata dall’estrazione nella ricerca precedente e come stato di validità, quello che è stato richiesto dal ServerG. A questo punto si elimina il file “serverV.dat”, si rinomina “tempServerV.dat” in “serverV.dat” e si crea un nuovo “tempServerV.dat”, così come avveniva in “centroVaccinaleRequestHandler(...)”. Solo se si entra in quest’ultimo branch condizionale, l’esito dell’aggiornamento richiesto in termini di scrittura all’interno del file viene considerato e impostato come “andato a buon fine” nel pacchetto di risposta che verrà inviato. Infine, si invia il pacchetto serverV_ReplyToServerG_clientT, si rilascia il mutex acquisito per l’accesso al file system e si liberano le risorse di memoria dinamica acquisite per gestire la richiesta, dopodiché il thread termina.

Si allega uno snippet di codice del main di ServerV per maggiore chiarezza. Per le routine si consiglia fortemente la visione del codice.

```
if (pthread_mutex_init(& fileSystemAccessMutex, (const pthread_mutexattr_t *) NULL) != 0)
raiseError(PTHREAD_MUTEX_INIT_SCOPE, PTHREAD_MUTEX_INIT_ERROR);
if (pthread_mutex_init(& connectionFileDescriptorMutex, (const pthread_mutexattr_t *) NULL) != 0)
raiseError(PTHREAD_MUTEX_INIT_SCOPE, PTHREAD_MUTEX_INIT_ERROR);
if (pthread_attr_init(& attr) != 0) raiseError(PTHREAD_MUTEX_ATTR_INIT_SCOPE,
PTHREAD_MUTEX_ATTR_INIT_ERROR);
if (pthread_attr_setdetachstate(& attr, PTHREAD_CREATE_DETACHED) != 0)
raiseError(PTHREAD_ATTR_DETACH_STATE_SCOPE, PTHREAD_ATTR_DETACH_STATE_ERROR);

while (TRUE) {
    ssize_t fullReadReturnValue;
    socklen_t clientAddressLength = (socklen_t) sizeof(client);
    connectionFileDescriptor = waccept(listenFileDescriptor, (struct sockaddr *) & client, (socklen_t *) &
        clientAddressLength);
    if ((fullReadReturnValue = fullRead(connectionFileDescriptor, (void *) & requestIdentifier,
        sizeof(requestIdentifier))) != 0) raiseError(FULL_READ_SCOPE, (int) fullReadReturnValue);

    if (pthread_mutex_lock(& connectionFileDescriptorMutex) != 0) raiseError(PTHREAD_MUTEX_LOCK_SCOPE,
        PTHREAD_MUTEX_LOCK_ERROR);
    int * threadConnectionFileDescriptor = (int *) calloc(1, sizeof(* threadConnectionFileDescriptor));
    if (!threadConnectionFileDescriptor) raiseError(CALLOC_SCOPE, CALLOC_ERROR);
    if ((* threadConnectionFileDescriptor = dup(connectionFileDescriptor)) < 0) raiseError(DUP_SCOPE,
        DUP_ERROR);
    if (pthread_mutex_unlock(& connectionFileDescriptorMutex) != 0) raiseError(PTHREAD_MUTEX_UNLOCK_SCOPE,
        PTHREAD_MUTEX_UNLOCK_ERROR);

    switch (requestIdentifier) {
        case centroVaccinaleSender:
            // passo sempre lo stesso TID. Non mi interessa fare join. I thread sono detached.
            if ((threadCreationReturnValue = pthread_create(& singleTID, & attr, &
                centroVaccinaleRequestHandler, threadConnectionFileDescriptor)) != 0)
                raiseError(PTHREAD_CREATE_SCOPE, PTHREAD_CREATE_ERROR);
            break;
        case clientS_viaServerG_Sender:
            if ((threadCreationReturnValue = pthread_create(& singleTID, & attr, &
                clientS_viaServerG_RequestHandler, threadConnectionFileDescriptor)) != 0)
                raiseError(PTHREAD_CREATE_SCOPE, PTHREAD_CREATE_ERROR);
            break;
        case clientT_viaServerG_Sender:
            if ((threadCreationReturnValue = pthread_create(& singleTID, & attr, &
                clientT_viaServerG_RequestHandler, threadConnectionFileDescriptor)) != 0)
                raiseError(PTHREAD_CREATE_SCOPE, PTHREAD_CREATE_ERROR);
            break;
        default:
            raiseError(INVALID_SENDER_ID_SCOPE, INVALID_SENDER_ID_ERROR);
            break;
    }
    wclose(connectionFileDescriptor);
}
```


Manuale utente

7 - Manuale utente

7.1 – Istruzioni per la compilazione

Per quanto riguarda la fase di compilazione, è reso disponibile un makefile. Un makefile consiste in linee di testo che definiscono un file (o un gruppo di file) oppure il nome di una regola dipendente dal gruppo di file. I file generati sono contrassegnati come i loro file sorgenti, mentre i file sorgenti sono contrassegnati a seconda dei file inclusi internamente. Dopo che ogni dipendenza è dichiarata, può seguire una serie di linee indentate (da tabulazioni) che definiscono come trasformare i file di ingresso nei file d'uscita, se il primo è stato modificato più di recente rispetto al secondo. Nel caso in cui tali definizioni sono presenti, queste sono riferite a dei script di compilazione e sono passate alla shell per generare i file target. Le varie entità software sono state testate e validate al fine di ottenere un corretto funzionamento e un'elevata efficienza in termini di memoria grazie all'ausilio di Valgrind. Si veda la sitografia per maggiori informazioni al riguardo.

I singoli eseguibili delle varie entità analizzate nelle sezioni precedenti sono già disponibili. Si precisa che sono stati generati su una macchina con processore Intel e sistema operativo macOS Monterey 12.1. Alternativamente è possibile ricompilare tutti i sorgenti ed effettuare il linking delle relative librerie in maniera facile e veloce tramite il makefile realizzato. Per fare ciò, basta recarsi da shell (o terminale) in “../VanillaGreenPass/src/”, digitare il comando “make” e premere il tasto “Invio” (o Enter). A questo punto verrà avviata l'utility make che provvederà ad effettuare le operazioni dette poc'anzi in base alla struttura del makefile realizzato. Il makefile è consultabile in “../VanillaGreenPass/src/Makefile”. Qualora per qualche motivo il makefile non dovesse funzionare, si consiglia di utilizzare i consueti comandi per la compilazione e il linking da terminale per ogni entità individuata precedentemente.

Infine, non sono richieste particolari librerie esterne. In ogni caso si allegano di seguito le librerie utilizzate:

- stdio.h
- stdlib.h
- unistd.h
- errno.h
- string.h
- sys/types.h
- sys/socket.h
- arpa/inet.h
- time.h
- ctype.h
- pthread.h
- stdarg.h

7.2 – Istruzioni per l’esecuzione

Si prendono ora in considerazione le istruzioni per l’avvio e l’esecuzione di ognuna delle entità del sistema software realizzato. Ogni entità del sistema (tranne che per il ServerV) dispone di un file di configurazione all’interno del quale sono riportate due informazioni: IP e porta. Tali informazioni sono l’IP e la porta dell’entità di rete alla quale è necessario collegarsi per adempiere le proprie funzioni. Di conseguenza ogni entità che dispone di tale file “.conf” vuol dire che ha bisogno di mettersi in collegamento, comunicare con un’altra entità di rete al fine di un corretto funzionamento. Il file di configurazione prende il nome dall’entità che ha la necessità di avere tale file. La struttura generica di un file di configurazione è la seguente:

```
<IP dell’entità di rete alla quale collegarsi>  
<Porta dell’entità di rete alla quale collegarsi>
```

Il file ha anche una terza riga vuota di default che rappresenta l’EOF (end-of-file). Le informazioni sono separate su righe differenti grazie al carattere “new-line”.

Per esempio, il “clientCitizen” avrà nel proprio file “clientCitizen.conf” sulla prima riga l’IP del centro vaccinale al quale collegarsi e sulla seconda riga la porta del centro vaccinale al quale collegarsi (o alternativamente sulla quale il centro vaccinale è in ascolto per un clientCitizen). Quindi fattore essenziale per il suo avvio è impostare tali parametri. Essendovi la possibilità di avere più di un centro vaccinale, si sceglie uno di essi e si impostano i suoi dati in base a quanto detto prima all’interno del file di configurazione. In maniera del tutto analoga all’interno del file di configurazione del centro vaccinale (“centroVaccinale.conf”) e del ServerG (“serverG.conf”), si ritroveranno IP e porta del ServerV al quale collegarsi; mentre all’interno del file di configurazione del ClientT (“clientT.conf”) e del ClientS (“clientS.conf”), si ritroveranno IP e porta del ServerG al quale collegarsi. Per cambiare tali parametri che attualmente sono impostati nella configurazione

```
<IP loopback>  
<porta progressiva>
```

basta andare ad aprire tali file con un qualunque editor di testo e apportare le modifiche opportune. Nella configurazione rilascio il ServerV si suppone sulla porta 20000, il ServerG sulla porta 30000, il CentroVaccinale sulla porta 10000. In base a queste ultime informazioni i file di configurazione .conf sono stati compilati di conseguenza.

Si prendono ora in considerazione gli argomenti di input passati da terminale per ogni entità realizzata seguenti la stringa “./entitàSoftwareDaAvviare”. Si precisa che le virgolette non sono necessarie, denotano soltanto il testo da inserire via terminale prima degli argomenti successivi.


- ClientCitizen e ClientS prevedono un argomento di input da linea di comando che consiste nel codice della tessera sanitaria. Tale codice deve rispettare il formato predefinito ipotizzato, ovvero formato da venti caratteri (lettere, numeri, simboli). In particolare, per ClientCitizen, tale argomento di input rappresenterà il codice della tessera sanitaria del cittadino che sta richiedendo una nuova dose di vaccino; mentre per ClientS rappresenterà il codice della tessera sanitaria associato a un Vanilla Green Pass che un ipotetico esercente (o altra entità del mondo reale) sta richiedendo di verificare.

- ClientT prevede due argomenti di input da linea di comando che corrispondono rispettivamente al codice della tessera sanitaria e al nuovo stato da associare al Vanilla Green Pass collegato a quel codice di tessera sanitaria (se esistente). Anche in questo caso il codice di tessera sanitaria deve rispettare il formato predefinito ipotizzato, ovvero formato da venti caratteri (lettere, numeri, simboli). Invece lo stato deve essere pari a 0 oppure a 1, ad indicare rispettivamente la volontà di invalidare il Vanilla Green Pass associato al codice di tessera sanitaria passato come primo argomento oppure la volontà di riattivare il Vanilla Green Pass associato al codice di tessera sanitaria passato come primo argomento. Qualora un Vanilla Green Pass fosse già attivo e si richiede di riattivarlo, quest'operazione non produce stati d'errore così come l'operazione speculare riguardante l'invalidazione.
- CentroVaccinale, ServerG e ServerV prevedono un solo argomento di input, ovvero la porta sulla quale viene attivato il rispettivo servizio del centro vaccinale, del ServerG, del ServerV. Chiaramente i numeri di porta devono essere consistenti con i requisiti di un numero di porta per il sistema operativo. Si suppone inoltre, che si faccia uso solo di numeri di porta registrati o effimeri. Infine, per un corretto funzionamento i numeri di porta dati in input a queste entità software devono essere coerenti con quelli presenti all'interno dei file di configurazione precedentemente analizzati.

Infine, un'ultima istruzione relativa all'esecuzione è l'ordine con il quale vanno eseguite le varie entità software realizzate. Si raccomanda per prima l'avvio del ServerV, poi dei CentriVaccinali e dei ServerG e infine l'avvio dei vari ClientCitizen, ClientS e ClientT. Qualora si dovesse seguire l'ordine inverso, ovvero avviando prima i client, verrà segnalato un errore e i programmi client terminano. Mentre qualora si dovessero avviare prima le entità software di secondo livello all'interno dell'architettura, ovvero ServerG e CentroVaccinale, senza aver prima avviato il ServerV, non si genererà un errore almeno fino a quando non vi sarà una richiesta di un client. Questo perché il collegamento col ServerV è realizzato soltanto in quel momento.

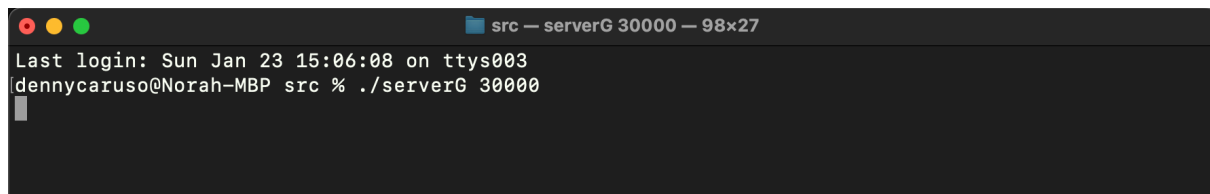
Si precisa che i test di esecuzione sono stati condotti oltre che su macOS Monterey 12.1 anche su una macchina virtuale Linux Mint 19.1 Cinnamon 4.0.10 con versione del kernel Linux 4.15.0-52-generic.

Si allegano qui di seguito alcune catture dello schermo relative all'esecuzione delle varie entità realizzate al fine di poter mostrare come avviarle correttamente e cosa si visualizza durante l'esecuzione. In ordine è possibile visionare l'esecuzione di un ServerV, di un ServerG, di un CentroVaccinale, di un ClientCitizen, di un ClientS, di un ClientT. Come si può notare ServerV, ServerG e CentroVaccinale non presentano output, a differenza di ClientCitizen, ClientS e ClientT. Inoltre, si rende evidente come l'interazione dell'utente è molto banale e facilitata in quanto consiste solo dell'avvio dell'entità di turno considerata.

A terminal window with a dark background. The title bar at the top shows a folder icon, the text "src — serverV 20000 — 98x27", and three colored window control buttons (red, yellow, green). The terminal text shows the user "dennycaruso@Norah-MacBook-Pro" in the "src" directory executing the command "./serverV 20000". A cursor is visible on the line following the command.

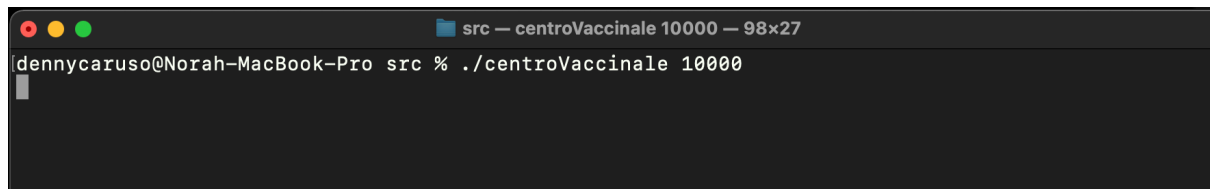
```
src — serverV 20000 — 98x27
dennycaruso@Norah-MacBook-Pro src % ./serverV 20000
```

Figura 7.1: Esecuzione di ServerV

A terminal window with a dark background. The title bar at the top shows a folder icon, the text "src — serverG 30000 — 98x27", and three colored window control buttons (red, yellow, green). The terminal text shows the user "dennycaruso@Norah-MBP" in the "src" directory executing the command "./serverG 30000". A cursor is visible on the line following the command. A login message "Last login: Sun Jan 23 15:06:08 on ttys003" is also visible.

```
src — serverG 30000 — 98x27
Last login: Sun Jan 23 15:06:08 on ttys003
dennycaruso@Norah-MBP src % ./serverG 30000
```

Figura 7.2: Esecuzione di ServerG

A terminal window with a dark background. The title bar at the top shows a folder icon, the text "src — centroVaccinale 10000 — 98x27", and three colored window control buttons (red, yellow, green). The terminal text shows the user "dennycaruso@Norah-MacBook-Pro" in the "src" directory executing the command "./centroVaccinale 10000". A cursor is visible on the line following the command.

```
src — centroVaccinale 10000 — 98x27
dennycaruso@Norah-MacBook-Pro src % ./centroVaccinale 10000
```

Figura 7.3: Esecuzione di CentroVaccinale

```

dennycaruso@Norah-MacBook-Pro src % ./clientCitizen 000000000000000000
Benvenuti al Centro Vaccinale
Numero tessera sanitaria: 000000000000000000

... A breve ti verra' inoculato il vaccino...

... Vaccinazione in corso ...

La vaccinazione e' andata a buon fine.
Data a partire dalla quale e' possibile effettuare un'altra dose di vaccino: 01-06-2022
Arrivederci.
dennycaruso@Norah-MacBook-Pro src % ./clientCitizen 000000000000000000

Benvenuti al Centro Vaccinale
Numero tessera sanitaria: 000000000000000000

... A breve ti verra' inoculato il vaccino...

Non e' possibile effettuare un'altra dose di vaccino. Devono passare almeno 5 mesi dall'ultima inoculazione.
Data a partire dalla quale e' possibile effettuare un'altra dose di vaccino: 01-06-2022
Arrivederci.
dennycaruso@Norah-MacBook-Pro src % ./clientCitizen 000000000000000001

Benvenuti al Centro Vaccinale
Numero tessera sanitaria: 000000000000000001

... A breve ti verra' inoculato il vaccino...

... Vaccinazione in corso ...

La vaccinazione e' andata a buon fine.
Data a partire dalla quale e' possibile effettuare un'altra dose di vaccino: 01-06-2022
Arrivederci.
dennycaruso@Norah-MacBook-Pro src % █

```

Figura 7.4: Esecuzione di ClientCitizen

```

dennycaruso@Norah-MBP src % ./clientS 00000000000000000000

Verifica GreenPass
Numero tessera sanitaria: 00000000000000000000

... A breve verra' mostrato se il GreenPass inserito risulta essere valido...

... Verifica in corso ...

La tessera sanitaria immessa risulta essere associata a un GreenPass attualmente valido.
Arrivederci.
dennycaruso@Norah-MBP src % ./clientT 00000000000000000000 0

Aggiornamento Validita' GreenPass
Numero tessera sanitaria: 00000000000000000000

... A breve verra' inviato il nuovo stato di validita' del Green Pass...

... Aggiornamento in corso ...

L'aggiornamento del Green Pass associato alla tessera sanitaria 00000000000000000000, e' andato a
buon fine.
Arrivederci.
dennycaruso@Norah-MBP src % ./clientS 00000000000000000000

Verifica GreenPass
Numero tessera sanitaria: 00000000000000000000

... A breve verra' mostrato se il GreenPass inserito risulta essere valido...

... Verifica in corso ...

La tessera sanitaria immessa non risulta essere associata a un GreenPass attualmente valido.
Arrivederci.
dennycaruso@Norah-MBP src % █

```

Figura 7.5: Esecuzione di ClientS

```

src --zsh -- 98x27
dennycaruso@Norah-MBP src % ./clientT 00000000000000000000 0

Aggiornamento Validita' GreenPass
Numero tessera sanitaria: 00000000000000000000

... A breve verra' inviato il nuovo stato di validita' del Green Pass...

... Aggiornamento in corso ...

L'aggiornamento del Green Pass associato alla tessera sanitaria 00000000000000000000, e' andato a
buon fine.
Arrivederci.
dennycaruso@Norah-MBP src % ./clientT 00000000000000000000 1

Aggiornamento Validita' GreenPass
Numero tessera sanitaria: 00000000000000000000

... A breve verra' inviato il nuovo stato di validita' del Green Pass...

... Aggiornamento in corso ...

L'aggiornamento del Green Pass associato alla tessera sanitaria 00000000000000000000, e' andato a
buon fine.
Arrivederci.
dennycaruso@Norah-MBP src % █

```

Figura 7.6: Esecuzione di ClientT

Sviluppi futuri

8 - Sviluppi futuri

È chiaro che aggiungere peculiarità e dettagli in linguaggio naturale risulta essere molto semplice, un po' meno l'effettiva implementazione di alcuni di essi. Si riportano di seguito alcuni ulteriori aggiornamenti che sarebbe desiderabile implementare nel sistema software realizzato.

- Interfacciamento con l'utenza manageriale al fine di cambiare periodo di validità di un singolo Vanilla Green Pass, per modificare il formato di memorizzazione delle informazioni relative a un Vanilla Green Pass, per effettuare operazioni di ricerca e gestionali all'interno dell'insieme dei Vanilla Green Pass archiviati, gestione scorte vaccini per il centro vaccinale, etc.
- Sostituire il salvataggio dei dati da file memorizzato su file system a un database vero e proprio (relazionale, ad oggetti, etc). Ciò comporta una revisione delle interazioni finora previste, l'interazione e l'inclusione con librerie/tool esterni come, per esempio, SQLite.
- Rendere il ServerV maggiormente decentralizzato con l'esecuzione di più istanze del ServerV, eliminando così la problematica del single point of failure (SPOF).
- Risoluzione dei nomi a dominio mediante DNS.
- Risoluzione degli indirizzi IPv6 e/o doppia gestione di indirizzi sia IPv4 che IPv6.
- Realizzazione di un'interfaccia grafica appropriata che sostituisca quella a linea di comando.
- Crittografia dei dati trasmessi da un'entità della rete all'altra e serializzazione secondo un formato proprietario specifico dei dati salvati in maniera persistente su supporto di memorizzazione di massa (file di configurazione, eventuali file temporanei, etc.).

