GoBox

Degiacomi Simone

IIS “B. Castelli”

Classe 5^ AI

Anno scolastico 2015/2016

Premessa

La realizzazione di questo progetto nasce dalla mia passione per la programmazione e per le strutture informatiche la cui funzione è quella di permettere la comunicazione dei dispositivi in rete.

Sempre più spesso sentiamo parlare di “cloud”. Per cloud personale si intende una piattaforma in grado di rendere disponibili collezioni di risorse (principalmente documenti, immagini e file) condivisi da un utente a qualsiasi altro dispositivo autorizzato. Alcuni esempi sono Dropbox, Google Drive e Mega.

Grazie a internet è possibile trovare un’infinita lista di progetti e guide che consentono la realizzazione di un cloud personale attraverso strumenti domestici e d’uso comune. Sebbene molti utenti riescano a raggiungere il risultato desiderato seguendo tali procedure, è tuttavia richiesto l’utilizzo di diverse tecnologie e di software che necessitano di essere configurati in modo corretto.

Proprio in virtù di ciò è sorta l’idea di creare una piattaforma semplice da utilizzare e che, attraverso l’utilizzo di un singolo software, consenta la realizzazione un sistema di gestione dei file e dei documenti personali che possa essere alla portata di un utente comune.

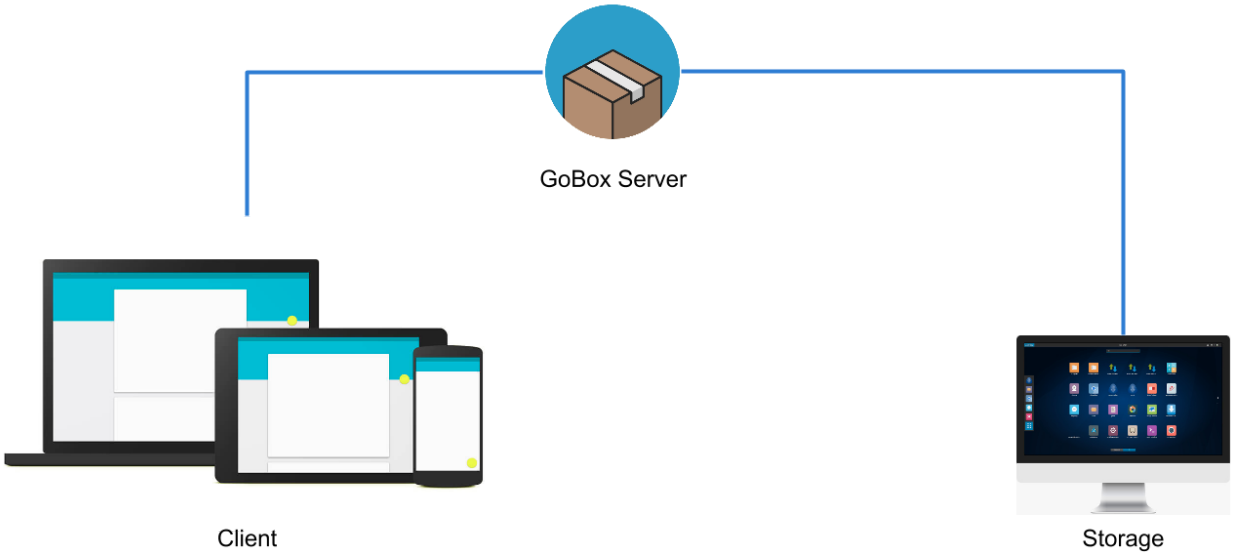
Indice

Descrizione

Questo documento descrive le funzioni della piattaforma Gobox e la loro relativa implementazione. Il testo è suddiviso in due parti: la prima consiste nella descrizione della piattaforma e delle funzionalità offerte, riassumendo in generale l'interazione tra i vari componenti. La seconda invece analizza l'implementazione e le tecnologie utilizzate per la sua realizzazione.

Introduzione

GoBox è una piattaforma che permette di realizzare in modo facile e veloce un cloud personale, utilizzando il proprio computer come spazio di archiviazione. Ogni utente in possesso delle proprie credenziali potrà accedere ai propri file da qualsiasi dispositivo attraverso l'apposita WebApp, oppure potrà sincronizzare i file direttamente su più computer. Inoltre, gli utenti avanzati saranno in grado di definire nuovi moduli da aggiungere al proprio cloud in modo da ampliarne le funzioni in relazione alle loro necessità.



Architettura generale

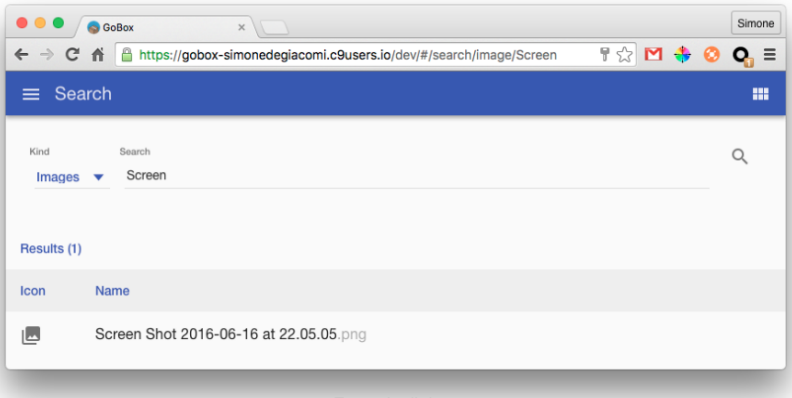
Funzionalità

Di seguito sono riportate le funzioni principali offerte dalla piattaforma. Queste funzioni sono accessibili da qualsiasi dispositivo, inclusi computer, smartphone e tablet, attraverso l'applicazione web utilizzabile per mezzo del browser.

Accesso ai propri file

La principale funzione di ogni sistema cloud è quella di garantire l'accesso ai propri file da qualsiasi dispositivo, indipendentemente dalla propria posizione e dal metodo di connessione. GoBox permette infatti di caricare, aggiornare e scaricare file e cartelle. In base al tipo di client utilizzato verranno proposte delle anteprime dei file, permettendo in tal modo di verificare se il file individuato corrisponda a quello desiderato, senza doverlo prima trasferire interamente.

Ricerca



Esempio di ricerca

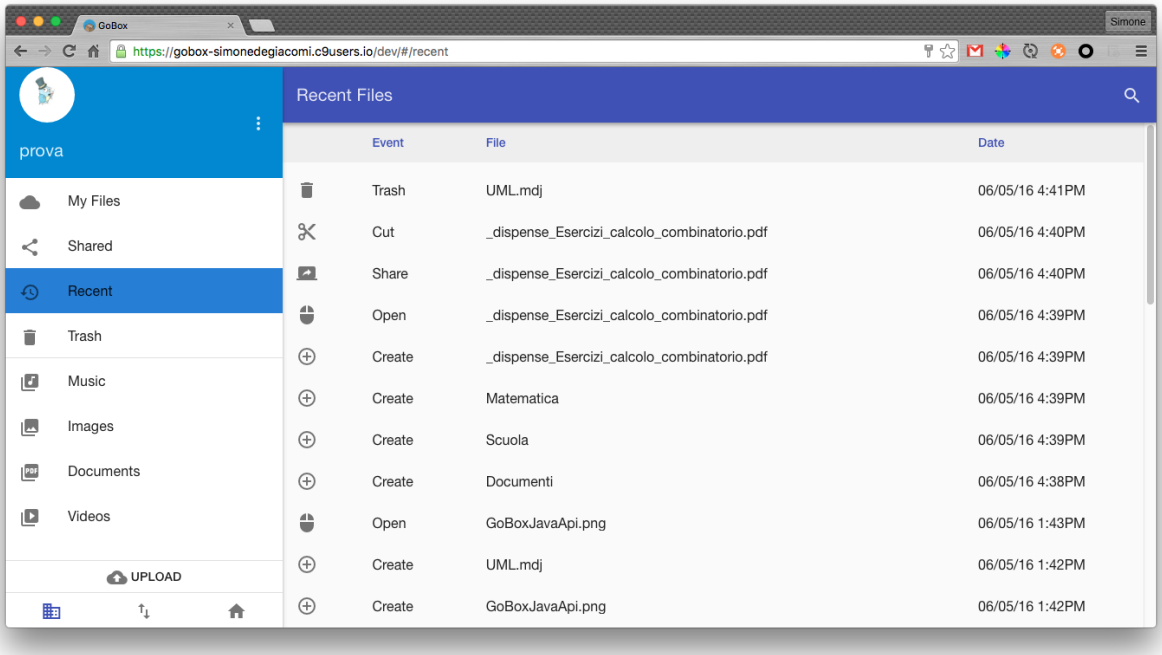
Per poter gestire un elevato numero di file, gli utenti si servono solitamente di uno strumento che sia in grado di filtrare e categorizzare i propri file. GoBox fornisce quindi, attraverso la WebApp, questo strumento, caratterizzato da un semplice filtro per parola chiave o per categoria (file audio, immagini, video e documenti). Per garantire una ricerca veloce, i file vengono indicizzati in un database al momento del loro caricamento.

Condivisione

Punto di forza di GoBox è anche la semplice condivisione dei file caricati sul proprio storage. L'utente avrà infatti la possibilità di condividere un qualsiasi file o cartella generando un link ad accesso pubblico. Il link porterà il visitatore in una pagina all'interno della WebApp, che risulterà accessibile anche agli utenti non registrati. In questa pagina sarà mostrata l'anteprima del file oppure sarà possibile scaricarlo. Eventuali modifiche del file condiviso effettuate dal possessore dello storage verranno automaticamente pubblicate. Il possessore dell'account potrà disattivare il link (impedendo dunque l'accesso al file) in qualsiasi momento.

Cronologia

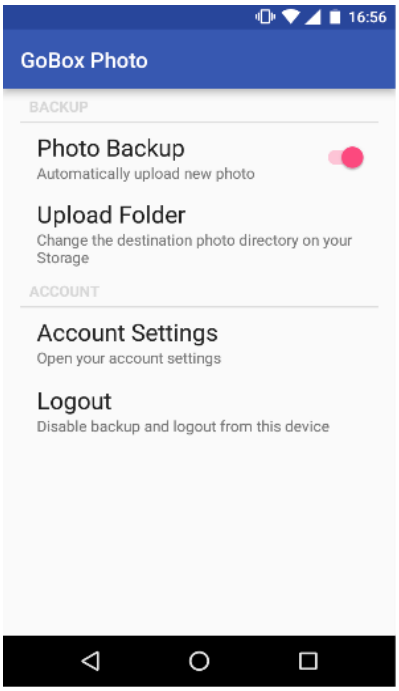
Attraverso la WebApp è inoltre possibile consultare una cronologia di tutte le operazioni effettuate sui propri file. La lista è accompagnata dal nome del file modificato, il tipo di modifica apportata ad esso (con annessa un'icona) e la data e l'ora della generazione dell'evento.



Pagina dedicata alla cronologia

Backup Foto

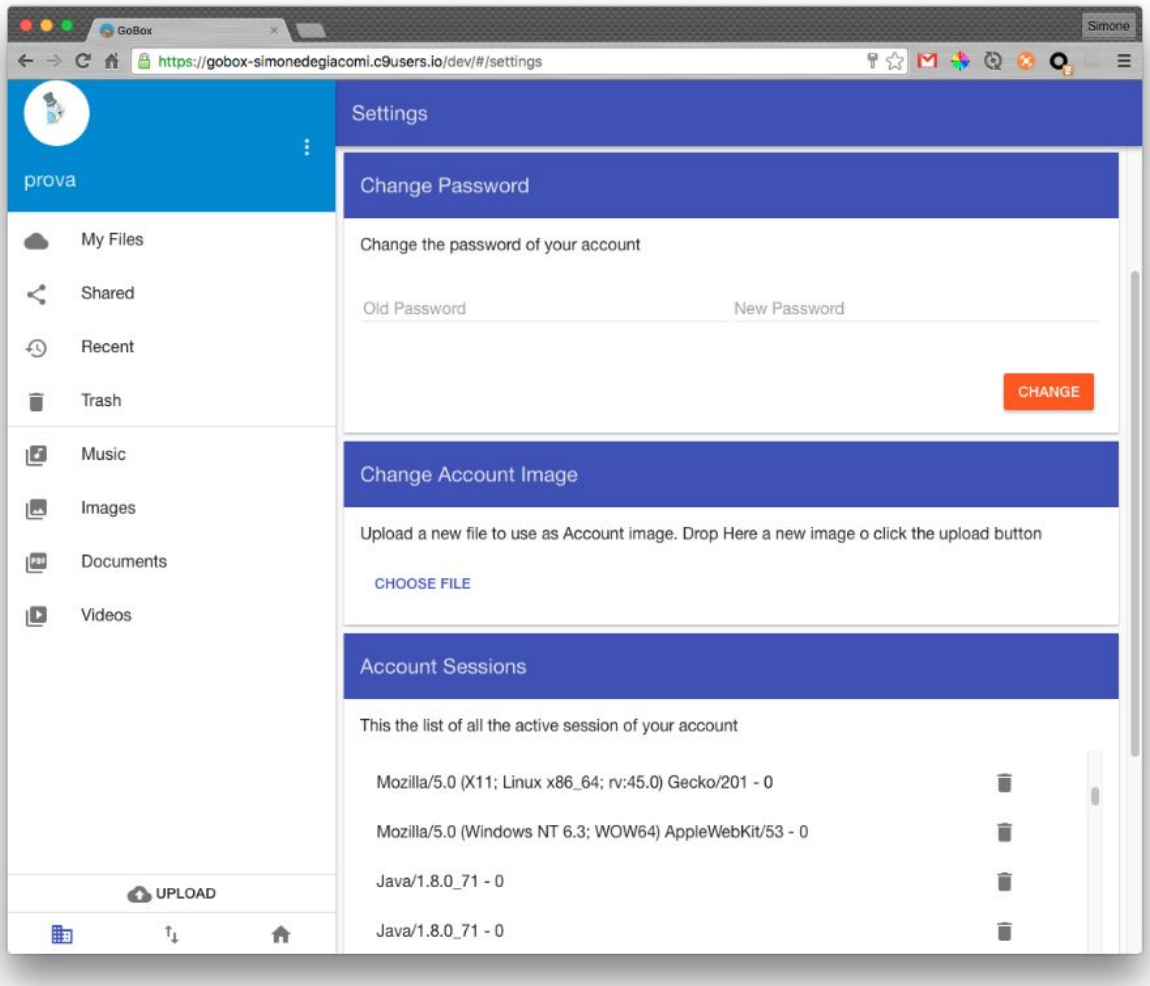
La piattaforma GoBox include anche un'applicazione nativa per Android che può essere utilizzata per effettuare il backup automatico delle fotografie acquisite tramite la fotocamera dello smartphone. Scaricando l'applicazione ed eseguendo il login sarà infatti possibile scegliere la cartella dello storage di destinazione delle fotografie caricate, che risulteranno quindi immediatamente accessibili attraverso il client per pc o la WebApp. Ogni volta che una nuova fotografia viene acquisita, l'utente riceverà una notifica e verrà informato dello stato attuale del caricamento della foto. Qualora il caricamento non dovesse essere portato a termine, una notifica segnalerà l'errore e, toccando la notifica, l'utente potrà accedere alle impostazioni dell'applicazione per risolvere il problema (per esempio nel caso in cui la cartella di destinazione venisse cancellata). Infine è possibile attivare o disattivare il caricamento automatico in qualsiasi momento, semplicemente toccando l'apposito *toggle* (pulsante acceso/spento) dell'applicazione.



Applicazione Android

Gestione delle sessioni

Nel caso in cui l'utente carichi sul proprio storage file personali a cui desidera accedere da qualsiasi luogo e con qualsiasi mezzo, anche utilizzando computer o smartphone non propri, è necessario che egli inserisca le proprie credenziali in tali dispositivi, con cui si potrà avere un accesso illimitato ai propri documenti. Per questo motivo è stato aggiunto un pulsante, sia nella WebApp sia nel client per computer, che permette di eseguire il logout alla fine dell'utilizzo su tale dispositivo, invalidando il token associato ad esso. Nel caso in cui questa procedura di sicurezza non venisse rispettata, è comunque possibile disabilitare la sessione di un qualsiasi dispositivo attraverso l'interfaccia web.



Schermata delle impostazioni

Istruzioni d’uso

Preparazione

Per creare il proprio cloud è necessario che l'utente si registri all’apposito sito web fornendo un nome utente, la propria mail ed una password. Una volta effettuata la registrazione basterà scaricare il programma da installare sul computer destinato alla memorizzazione dei file. Questo computer, all’interno di GoBox, viene chiamato *storage* e dovrà essere costantemente acceso e connesso a internet per gestire e rendere accessibili i propri file. Dato che il software è realizzato in Java e non richiede computer con elevata potenza di calcolo, è possibile utilizzare qualsiasi dispositivo economico e a basso consumo energetico.

Utilizzo

Una volta che lo storage è stato inizializzato e connesso ad internet, risulterà possibile accedere e modificare da subito i propri file in diversi modi:

- utilizzando la WebApp presente sul sito web, disponibile per computer, tablet e smartphone;
- installando il client su un computer diverso dallo storage, per ottenere la sincronizzazione automatica;
- scaricando l'applicazione Android per abilitare il backup automatico delle fotografie acquisite;

Componenti

Storage

Lo storage, una volta configurato, stabilirà una connessione diretta con i server di GoBox, rimanendo poi in costante attesa di ricevere richieste dai vari client (ovvero i propri dispositivi). I compiti di questo componente sono tre: la gestione delle richieste dei client, il monitoraggio del file system su cui è installato e la gestione di una cronologia di tutte le azioni effettuate sui file.

Richieste dei client

Lo storage si occupa di soddisfare tutte le richieste che vengono effettuate dai client. In base al tipo di richiesta e azione che essa comporta, lo storage trasferisce i file necessari ed esegue delle operazioni sul database. Inoltre, alcune operazioni possono generare un evento, che verrà registrato nel database e propagato a tutti i client attualmente connessi. Le azioni che comportano la creazione degli eventi sono per esempio il caricamento di un file, la modifica e l’apertura di esso, mentre azioni come la visualizzazione dell’anteprima di un file non comportano la generazione di alcun evento.

Monitoraggio dei file

Per monitorare i file presenti sul file system su cui è installato lo storage, viene avviato un servizio che si occuperà di "sorvegliare" i file e le cartelle per qualsiasi tipo di modifica. La parte di software che esegue questo compito utilizza, quando possibile, delle API offerte dal sistema operativo (per esempio *inotify* su Linux) per non dover ricadere su un *polling* (controllo continuo dello stato dei file) delle cartelle, in modo da ridurre l'impatto prestazionale.

Storico degli eventi

Lo storage salva all'interno del proprio database tutti gli eventi generati dalle operazioni dell'utente sui propri file, come l'apertura (in questo caso il controllo è limitato ai file aperti tramite la WebApp, poichè non è possibile catturare questo tipo di evento sul file system), il loro caricamento, modifica o cancellazione.
Inoltre, ogni qual volta che un'azione genera un nuovo evento, quest'ultimo viene inviato dallo storage a tutti i client, per poterli avvisare e fornirgli le indicazioni necessarie per completare la sincronizzazione.

Client

Il client per computer è un software che rimane in background per poter monitorare i cambiamenti apportati ai file. Ogni volta che il computer viene avviato oppure quando il client entra in funzione, viene eseguito un confronto con tutti i file presenti sul file system locale e sui file presenti nello storage, e vengono effettuate poi le necessarie azioni di sincronizzazione per aggiornare lo stato dei file del client. Una volta terminata la fase di sincronizzazione, il programma entra in uno stato di attesa per eventi dal file system locale o dallo storage. Per comunicare all'utente lo stato di eventuali trasferimenti, il programma crea un'icona nella *system tray*, che racchiude tutte le informazioni principali e la possibilità di disattivare o modificare l'esecuzione del client.

WebApp

La WebApp è il software pensato per la visualizzazione e la gestione dei file da smartphone, tablet e qualsiasi dispositivo dotato di un browser. Grazie alla configurazione grafica, la WebApp si adatterà al dispositivo su cui è utilizzata, offrendo controlli immediati all'utente.
Da questo client è possibile:

- caricare e scaricare file dal proprio storage;
- amministrare il proprio account cambiando l'immagine del profilo, sostituire la password o gestire le sessioni attive;
- eseguire una ricerca tra i propri file per nome e tipo;
- consultare la cronologia delle operazioni eseguite sui propri file;
- gestire la condivisione dei file;
- spostare nel cestino o ripristinare i file;

Applicazione Android

Come anticipato, la piattaforma è affiancata da un'applicazione Android nativa per il caricamento automatico delle fotografie acquisite con la fotocamera dello smartphone.
Le funzioni che questo software espone sono:

- la selezione della cartella di destinazione, in cui andranno caricate le immagini;
- un pulsante che permette di attivare o disattivare il servizio in qualsiasi momento, senza modificare le impostazioni e mantenendo le immagini precedentemente caricate;

Server Principale

Il server principale è il componente che si occupa della gestione e autenticazione degli utenti e mette in collegamento i client con lo storage. L'iterazione tra l'utente e questo componente è nascosta, poichè viene utilizzato direttamente solo dai componenti precedentemente elencati.

Architettura

Una delle componenti fondamentali per la piattaforma GoBox riguarda la comunicazione tra i vari client, lo storage dell'utente e il server principale.
Gli obbiettivi che la struttura deve soddisfare sono:

- **Bassa latenza**: Visualizzazione immediata della lista dei file e esecuzione veloce di tutte le operazioni che non richiedono l'effettivo trasferimento dei dati, come la copia di file in cartelle diverse, la rinomina, la ricerca e la condivisione.
- **Sicurezza**: Deve garantire un sufficiente livello di sicurezza per l'utente, indipendente dalla connessione o dal dispositivo utilizzato per accedere al proprio account.
- **Portabilità**: Deve essere facilmente implementabile su tutti i dispositivi e sulle varie piattaforme.

Per realizzare un'infrastruttura che soddisfasse questi requisiti, sono stati scelti 2 protocolli.

Protocolli Utilizzati

WebSocket

Il protocollo WebSocket (la cui sigla è WS) fornisce un canale di comunicazione di tipo *full-duplex* (bidirezionale). Uno dei suoi principali vantaggi è il fatto di essere utilizzabile all'interno dei browser web, estendendo le possibilità delle WebApp, soprattutto nell'ambito di propagazione di eventi in real-time. Questo protocollo è nato nel 2011 ed è standardizzato dall'IETF con il nome RFC6455.
Ogni connessione Websocket viene iniziata attraverso una procedura di *handshake* composta da una normale richiesta HTTP. In questa richiesta devono essere specificati dei particolari *Header* (Proprietà delle richieste HTTP) che identificano la versione del protocollo con cui il browser (o comunque il client) vuole comunicare. Quando il server rileva questi Header, risponde con un codice di risposta HTTP 101, il quale indica un cambio di protocollo. Dato che la procedura di handshake è basata su HTTP, è possibile specificare qualsiasi header e sono, di conseguenza, presenti anche i cookie (di questo verrà discusso nella sessione dedicata all'autenticazione). Inoltre è possibile utilizzare una richiesta di tipo HTTPS durante la procedura handshake: in questo caso si parla di protocollo Secure Websocket (WSS) che non è altro che il medesimo protocollo con l'aggiunta del TLS, per garantire autenticità e segretezza.

Esempio di un URI con protocollo Websocket:
`ws://www.domain.com/`

Esempio di un URI con protocollo Secure Websocket
`wss://www.domain.com/`

Una volta stabilito un canale Websocket il client e il server possono scambiarsi dei messaggi (comprese stringhe e file binari) a vicenda.

Struttura dei messaggi

Come precedentemente anticipato la comunicazione attraverso i Websocket consiste nell'invio reciproco di messaggi. Nel caso di GoBox, i messaggi vengono codificati in JSON e viene dato ad ogni messaggio un nome.

In questo modo si ottiene un messaggio simile al seguente:

```
{
	"name": "nome dell'evento",
	"data": Object
}
```

Il valore del nome ("name") raggruppa quindi il tipo di messaggio in una categoria, mentre il valore "data" può essere a sua volta un altro oggetto, un vettore o un valore, che rappresenta il dato scambiato relativo a questo messaggio.

Questo risultato non è però in grado di soddisfare le necessità di comunicazione di GoBox, dato che buona parte delle singole operazioni che i client devono soddisfare richiedono più messaggi: una domanda rivolta verso lo storage e una relativa risposta.
Una semplice soluzione a questa necessità può essere l'invio di un messaggio dal client di tipo richiesta e la ricezione di una risposta inviata dallo storage, rispettando l'ordine di invio. Questa soluzione limita però le capacità di quest'ultimo, perchè durante le operazioni che richiedono tempo per l'elaborazione di una risposta, lo storage risulterebbe bloccato. Inoltre, bisogna ricordare che ogni storage gestisce più client contemporaneamente, quindi non solo un client risulterebbe inagibile, ma tutti quelli dello stesso utente (poichè lo storage dovrebbe rispettare appunto l'ordine).

Per risolvere questo problema viene introdotto un particolare messaggio, caratterizzato da un ID ("_queryId") che serve ad identificare la domanda.

La domanda si presenta quindi nel seguente modo:

```
{
	"name": "Nome della query",
	"_queryId": 123,
	"data": Object
}
```

La relativa risposta sarà un messaggio con il medesimo ID (associando la risposta alla precedente domanda) e che avrà come campo "name" una parola chiave che identifica la risposta ("queryResponse"). Anche nella risposta viene ovviamente mantenuto il valore "data".

Esempio di risposta:

```
{
        "name": "queryResponse",
        "_queryId": 123,
        "data": Object
}
```

La generazione dell'ID è compito del client, che può scegliere a suo piacere un qualsiasi numero: una buona scelta può essere l'utilizzo di un contatore o di un numero casuale.

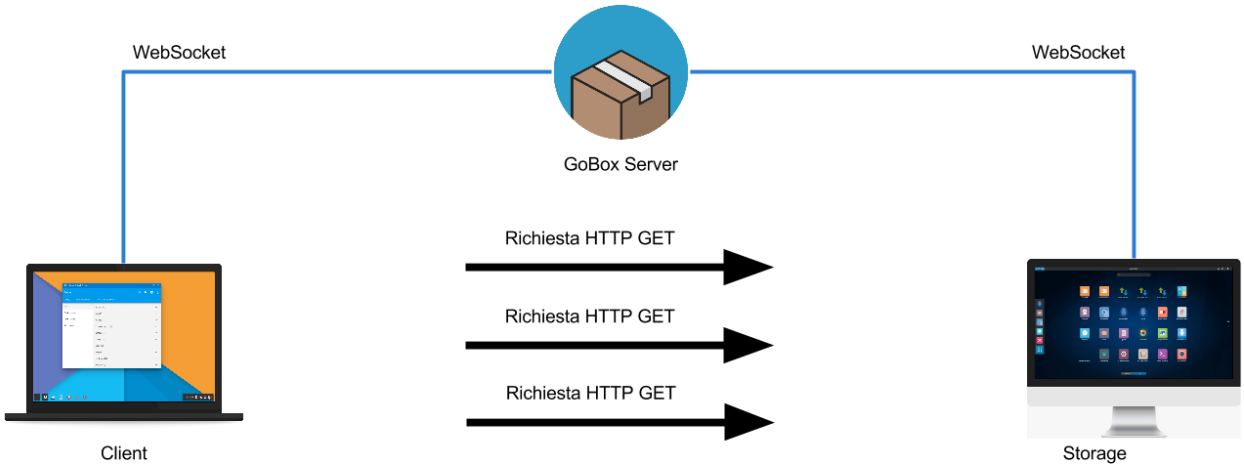
NB: Dato che l'ID è scelto da client, possono verificarsi problemi di ripetizione. Tenendo però conto che l'ID ha una "vita" lunga quanto il tempo passato dall'invio della richiesta alla ricezione della risposta, si è ritenuto un compromesso accettabile, rispetto alla generazione di essi dal lato storage (la quale avrebbe richiesto un ulteriore scambio di messaggi).

HTTPS

Il protocollo HTTPS è basato sul protocollo HTTP, con l'aggiunta della sicurezza fornita dal protocollo TLS.
Questo protocollo offre diversi vantaggi, tra cui il fatto di essere semplice da implementare con i principali linguaggi di programmazione e il fatto che permetta di trasferire qualsiasi tipo di contenuto.

Perché due protocolli?

Non sono stati utilizzati esclusivamente i WebSocket perchè i relativi canali sono basati su TCP, quindi durante il trasferimento di grandi quantità di dati (come ad esempio file multimediali) il canale sarebbe risultato ostruito. Questo problema è evitabile creando nuovi WebSocket all'occorrenza, ma dato che ogni WebSocket inizia con una richiesta HTTP, è sufficiente limitarsi ad essa, evitando i tempi di ulteriori handshake. In questo modo viene creata una nuova richiesta HTTP per ogni trasferimento, in modo da bloccare i singoli canali dedicati ai trasferimenti e lasciando libero il canale WebSocket.



Rappresentazione di richieste HTTP multiple

La piattaforma si affida quindi ad entrambi i protocolli per poter realizzare due tipi di comunicazioni:

- comunicazioni di servizio: Scambio di messaggi codificati in JSON, che permettono la gestione dei file, la ricerca, la gestione del cestino e delle condivisioni attraverso i WebSocket;
- trasferimento di file: L'effettivo scambio dei file tra il client e lo storage, utilizzando il protocollo HTTPS;

Difficoltà

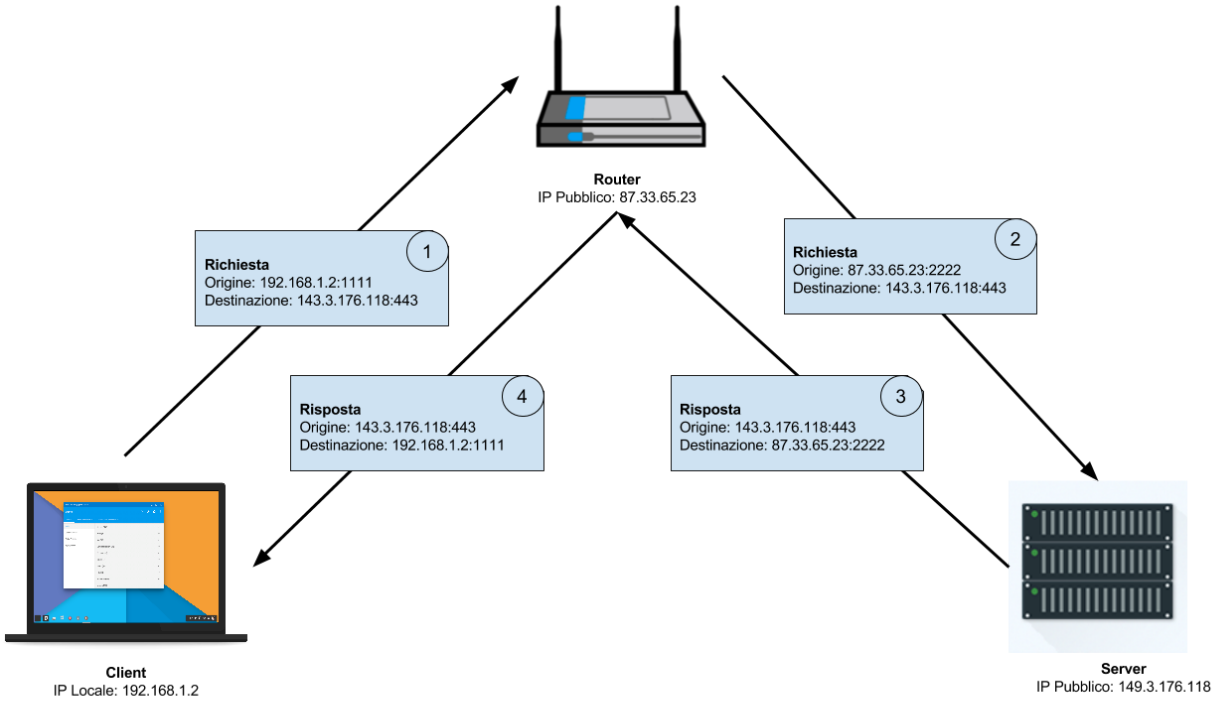
Volendo mantenere semplice l'utilizzo di GoBox, fornendo un servizio di tipo *zero-config* (ovvero senza la necessità di configurazioni da parte dell'utente), nascono dei problemi derivanti dai protocolli scelti: Per poter realizzare un server HTTPS (e di conseguenza anche WSS) è necessario disporre di un indirizzo IP pubblico e una corretta configurazione del *port-forwarding* del router, in modo tale che la porta in ascolto dal server sia accessibile da internet.

Port Forwarding

Questo problema nasce dal fatto che i dispositivi domestici che permettono di collegarsi ad internet utilizzano il NAT per poter far comunicare più dispositivi contemporaneamente su internet.

Il NAT è un metodo utilizzato per mappare più indirizzi IP, in questo caso i dispositivi connessi alla rete locale, ad un unico indirizzo IP (in questo caso quello pubblico). Quando viene utilizzato questo metodo, i pacchetti TCP inviati dai client della rete locale vengono modificati dal router, che si preoccuperà di memorizzare quale client ha inviato il pacchetto (per esempio attraverso l'utilizzo di una mappa chiave-valore che associa la porta all'indirizzo). Quando un pacchetto di risposta tornerà al router, quest'ultimo dovrà nuovamente modificare il pacchetto, sostituendo l'IP di destinazione con quello del client all'interno della rete locale.

Di seguito è rappresentata la modifica apportata ai pacchetti utilizzando il NAT:



Rappresentazione dell'applicazione del NAT

Per quanto riguarda le connessioni in ingresso, è necessario eseguire una configurazione di port-forwarding, comunicando al router l'indirizzo a cui dovrà delegare i pacchetti delle connessioni in arrivo dall'esterno. Questo procedimento è necessario perchè altrimenti un client all'esterno della rete locale che voglia comunicare con un preciso dispositivo (un server) dietro al NAT, pur conoscendo l'indirizzo IP pubblico del router, non sarebbe in grado di raggiungere il dispositivo desiderato, poichè il router non saprebbe a chi delegare la richiesta.

Questo tutto sommato rimane comunque un problema di poca importanza, dato che grazie a protocolli come l'UPnP questa configurazione può essere spesso automatizzata. Inoltre, nel caso in cui il protocollo UPnP non sia supportato dal router, o sia stato disabilitato, l'operazione richiesta all'utente è semplice.

IP Pubblico

Questo problema è invece più difficile da risolvere, perchè non tutti i provider di internet lo forniscono. La maggior parte di essi offrono il servizio utilizzando una *Carrier Grade NAT*, una configurazione molto simile a quella precedentemente illustrata, solamente che anziché essere eseguita a livello di rete locale (che comunque rimane, non viene sostituita), viene eseguita dall'operatore ad un livello geografico più ampio (a livello di MAN).

Esistono delle tecnologie, come l'*hole-punching*, che permettono, con una buona probabilità di successo, di contattare un dispositivo dietro ad una NAT, ma risultano difficili da implementare all'interno del browser.

Autenticazione e autorizzazione

Per gestire l'autenticazione e l'autorizzazione all'interno di GoBox viene utilizzato un sistema di autenticazione ad Account. Ogni account ha una password associata. Gli account vengono salvati nel database del server principale.

JSON Web Token

JSON Web Token (abbreviato JWT) è uno standard che definisce un metodo per il trasferimento di oggetti JSON sotto forma di stringhe compatte, e *self-contained*, ovvero possono essere inserite dove si preferisce, anche accodandole a URL senza doverle prima codificare.
Le informazioni contenute all'interno del token possono essere verificate poichè il token viene firmato attraverso algoritmi a chiave simmetrica (utilizzando HMAC) o asimmetrica (RSA).
Ogni token consiste in una stringa formata da tre parti, concatenate da un carattere di punto.

Queste tre parti sono:

- **Header**: Contiene le informazioni sulla creazione del token, sull'algoritmo con cui viene firmato, etc… .
- **Payload**: Contiene l'oggetto json criptato e convertito in base 64.
- **Signature**: Contiene tutte le componenti della firma del token.

Nel payload possiamo inserire tutte le informazioni necessarie che dobbiamo trasferire. Ovviamente, più dati inseriamo all'interno di questo oggetto, più il token risulterà lungo, una volta assemblato.
Attenzione però, il payload non viene criptato, ma solamente l'hash risultante, quindi non è consigliato inserire dati privati o password.
Nel caso di GoBox i token contengono l'ID dell'utente, un flag che indica se questo token è di un client o dello storage, e una stringa casuale che serve per poter identificare la sessione.

Modalità di connessione dei Client

Ogni client che si collegherà al proprio account potrà gestire i dati e il proprio account in tre modalità diverse.

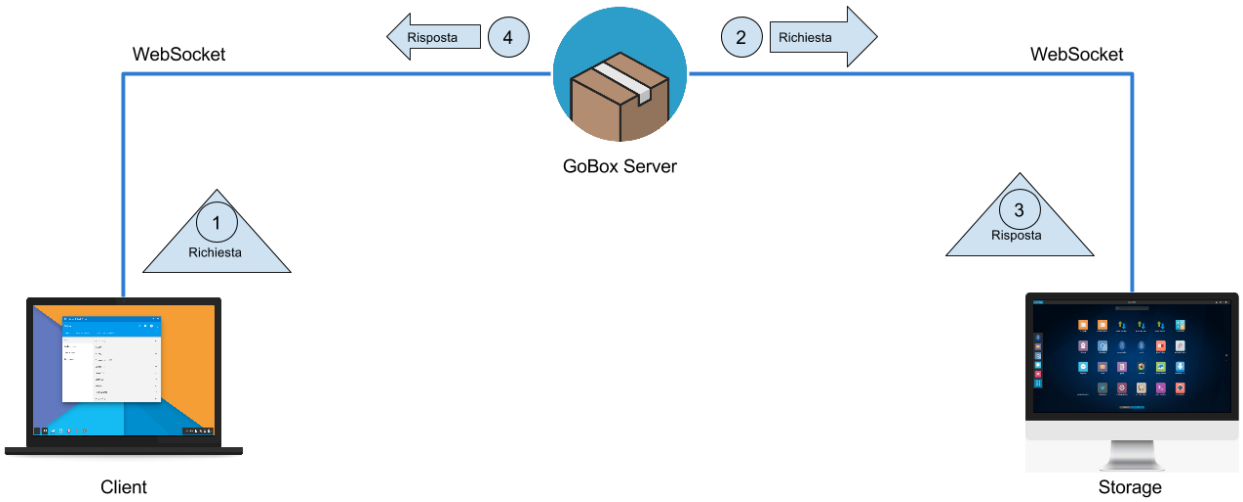
Modalità Bridge

La modalità Bridge è la modalità più stabile, ed è quella scelta da tutti i client all'avvio. Consiste in una connessione Websocket del client verso il server principale. In questo caso tutte le

richieste di servizio e i trasferimenti dei file verranno inviate al server principale, senza mai comunicare direttamente con lo storage, ma utilizzando il server principale come “ponte”.

Richiesta di servizio

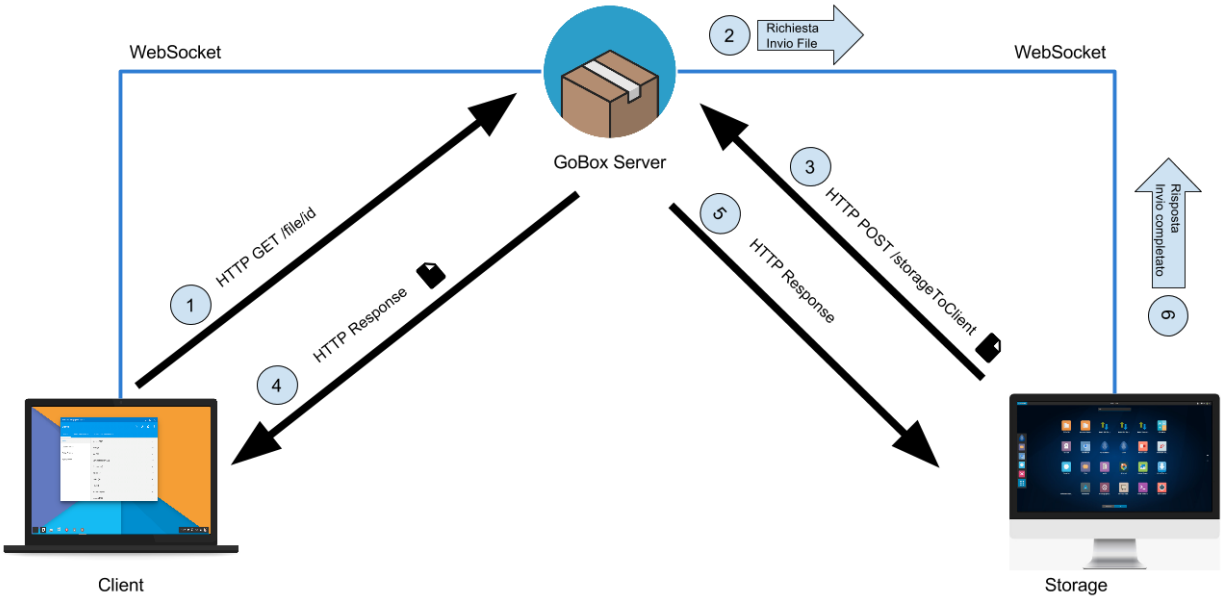
In questa modalità per poter effettuare le richieste di servizio il client invierà la richiesta al server. Il server registrerà quindi l’ID della richiesta (il campo “_queryId”) utilizzato e inoltrerà allo storage il messaggio, senza apportare alcuna modifica ad esso. Quando lo storage risponderà, il server leggerà e confronterà l’ID della richiesta per individuare il client che l’ha generata. Infine il server invierà la risposta al client e eliminerà il riferimento all’ID utilizzato.



Scambio di richieste e risposte tra client e storage attraverso il canale WebSocket con il server principale

Download di un file

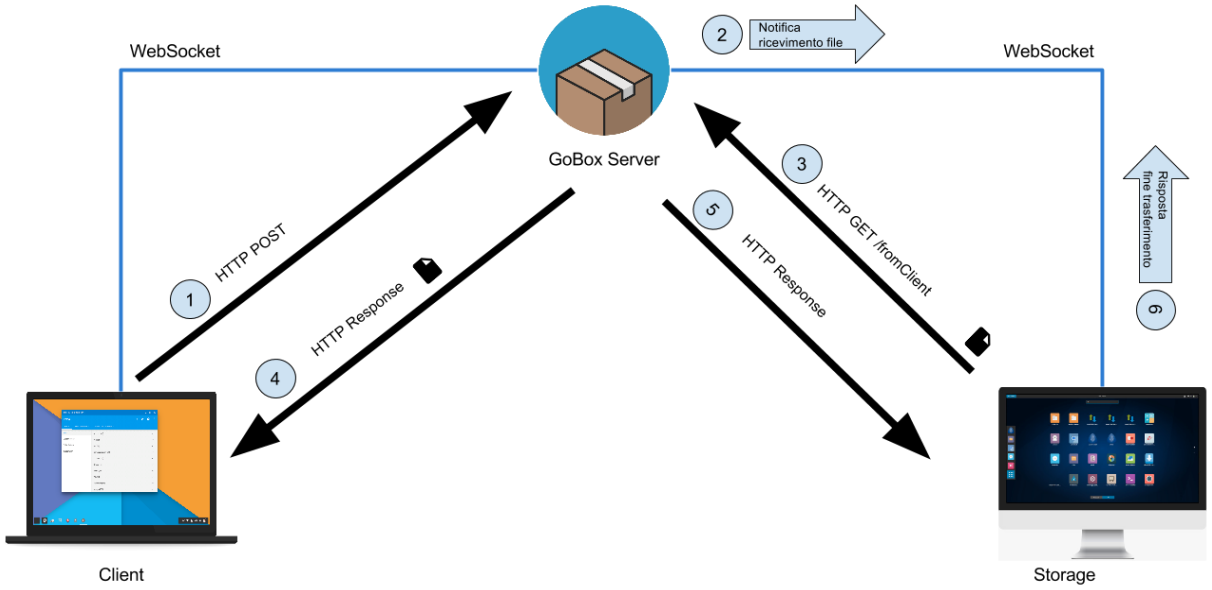
Nella modalità bridge, anche le richieste di trasferimento dei file (utilizzando il protocollo HTTP) devono essere fatte verso il server. In questo caso una richiesta di tipo GET viene fatta verso il server, specificando nei parametri del URL (query parameters) l’ID del file desiderato. Il server, ricevuta la richiesta HTTP, invierà una richiesta attraverso il canale Websocket allo storage. Lo storage che riceve la richiesta, dopo aver verificato che il file esiste, inizierà una connessione HTTP di tipo POST verso lo storage, inviando il file richiesto. Il server man man che riceve il file dallo storage lo invia al client, copiando il contenuto della POST del server nel contenuto della risposta alla GET del client. Quando il trasferimento del file sarà completato, lo storage concluderà la richiesta Websocket invita dal server, specificando eventuali errori durante il trasferimento.



Download di un file in modalità Bridge

Caricamento di un file

La procedura di caricamento di un file non differisce di molto rispetto a quella di download. Anche in questo caso il client invierà una richiesta HTTP (questa volta di tipo POST) al server, il quale comunicherà con lo storage avvisandolo dell'intenzione del client. Lo storage inizierà quindi una richiesta HTTP di tipo GET verso il server. Quindi, il contenuto della POST del client verrà copiato nella risposta della GET dello storage. A trasferimento completato, lo storage soddisferà la richiesta del server principale specificando la presenza di eventuali errori.



Caricamento di un file in modalità Bridge

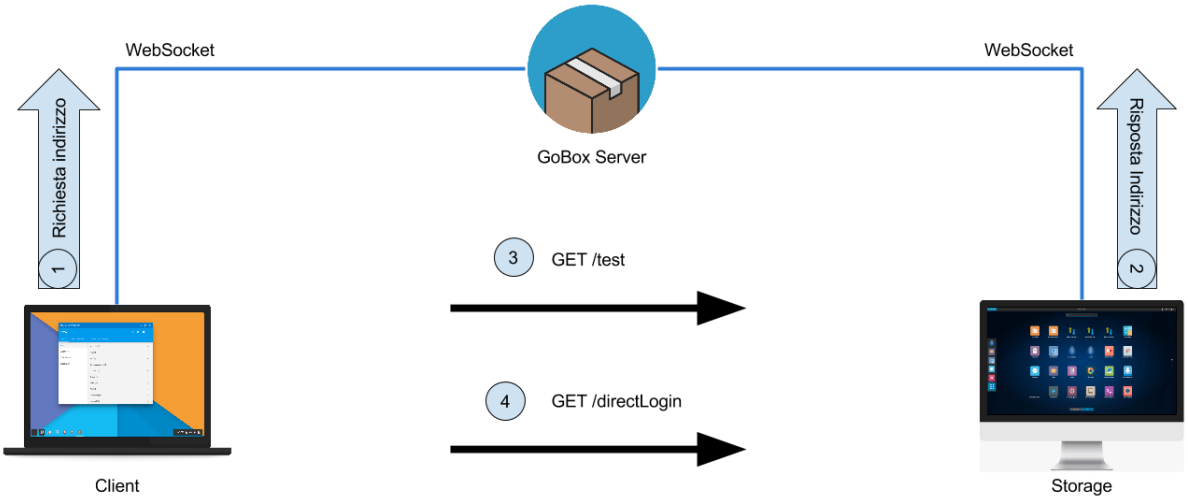
Modalità Direct

In questa modalità le richieste di servizio (lista dei file, gestione e modifica delle condivisione e gestione del cestino) devono essere inviate come richiesta Websocket al server, mentre le richieste di trasferimento (caricamento/scaricamento) dei file vengono effettuate direttamente allo storage. Questo garantisce minor carico sul server principale e una velocità di trasferimento maggiore, soprattutto per quanto ne riguarda la latenza, dato che il trasferimento non richiede la richiesta WebSocket intermedia.
Per utilizzare questa modalità il client deve eseguire una semplice procedura di handshake, iniziando necessariamente con una connessione completa in modalità Bridge. Una volta connesso, il client invierà una particolare richiesta WebSocket, alla quale lo storage risponderà con il proprio indirizzo IP pubblico, locale e con il numero di porta su cui è in ascolto. Infine, nella risposta è presente un stringa casuale, chiamata token temporaneo.
Il client può ora verificare se è in grado di collegarsi direttamente allo storage effettuando una richiesta HTTP di tipo GET all'indirizzo dello storage (viene effettuato prima un tentativo con l'indirizzo locale e poi con quello pubblico) ad un indirizzo di prova. Se la richiesta viene completata, il client può autenticarsi con lo storage, effettuando una nuova richiesta HTTP in cui specificherà il token temporaneo precedentemente ricevuto. A questo punto il token temporaneo non è più necessario, perchè un nuovo JWT viene generato dallo storage e inviato al client.

La risposta inviata dallo storage è simile alla seguente riportata:

```
{
        "temporaryToken": "ajKdkt451cE",
        "publicIP": "87.31.25.12",
        "localIP": "192.168.2.101",
        "certificate": [byte]
}
```

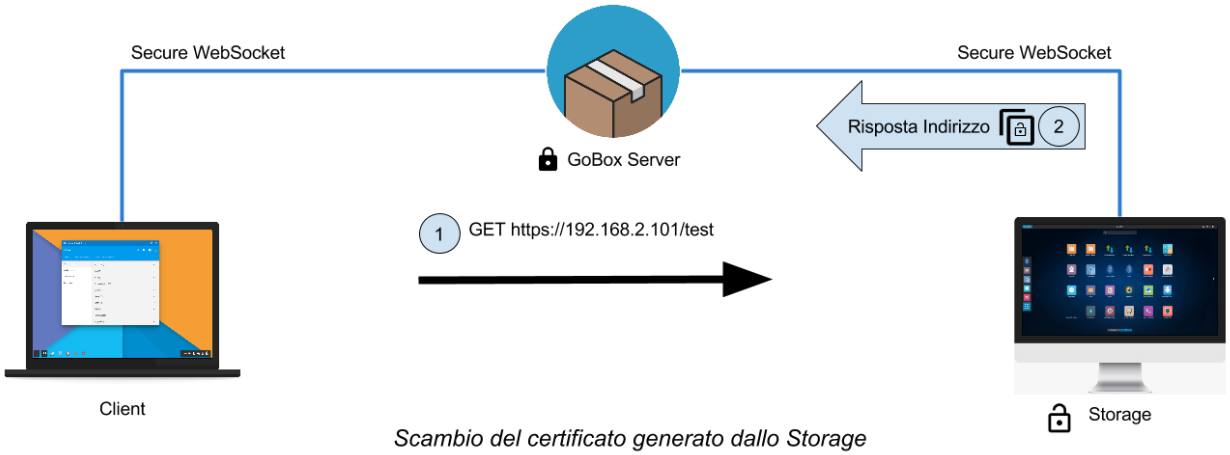
La procedura di handshake può essere raffigurata con il seguente schema:



Rappresentazione handshake modalità Direct

Certificato HTTPS

Durante la procedura di *handshake*, precisamente nel momento in cui il client riceve la risposta alla richiesta effettuata al server tramite il canale WebSocket, è possibile ottenere il certificato pubblico dello storage. In questo modo il client, quando esegue la connessione di test, può confrontare il certificato precedentemente ottenuto con quello della richiesta, verificando l'effettiva identità dello storage.

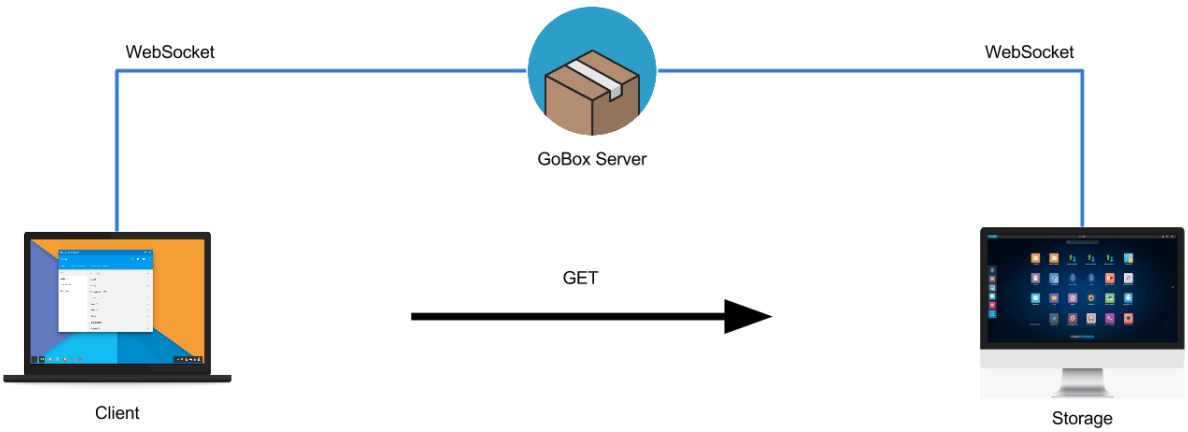


Scambio del certificato generato dallo Storage

Questo procedimento è necessario poichè lo storage dispone di un certificato auto generato, quindi non firmato e non verificabile. Dato che lo storage e il client sono entrambi connessi al server principale, attraverso il canale WebSocket sicuro (WSS) e tenendo conto che il server dispone di un certificato firmato, è possibile affermare con sicurezza l'identità dello storage.

Trasferimento di un file

La richiesta del trasferimento di un file del client connesso in modalità diretta verrà quindi inviata subito allo storage, senza passare per il server principale, evitando appunto la richiesta intermedia generata dal server.



Download di un file in modalità Direct

Nella figura è rappresentato il download di un file. Nel caso di un upload, l'unica differenza riguarderebbe il tipo di richiesta HTTP, precisamente di tipo POST.

Modalità Direct Local

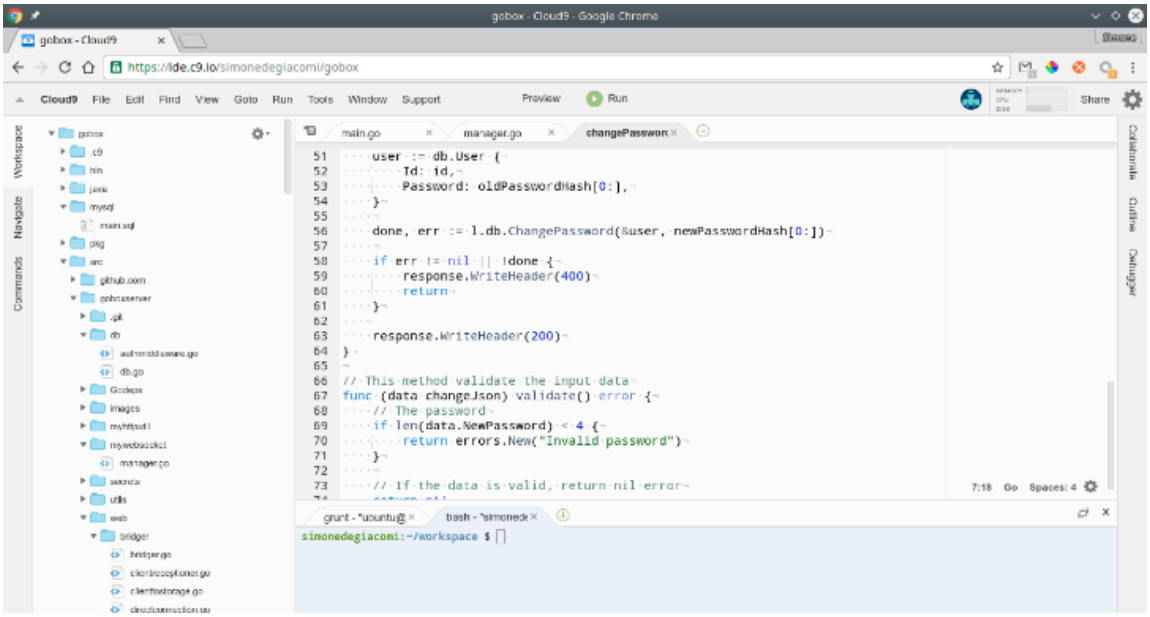
Questa modalità è quella che garantisce maggiori prestazioni se confrontata con le precedenti. Condivide la stessa base e handshake della modalità Direct, ma in questo caso tutte le richieste HTTP rivolte verso lo storage avvengono nella stessa rete locale, senza dover attraversare internet. Se la realizzazione di una connessione in modalità Direct soffre di problemi dovuti alla configurazione della rete, questa dovrebbe essere esente, salvo che in configurazioni di rete complesse.

Fase di sviluppo

Ambiente di sviluppo

Logo Cloud9

Per lo sviluppo del server e della web app è stata utilizzata la piattaforma Cloud9 che fornisce una macchina virtuale nel cloud e un relativo ambiente di sviluppo. Inoltre, viene associato un URL alla macchina virtuale, che è quindi possibile sfruttare come hosting per lo sviluppo.



Interfaccia di Cloud9

Macchina virtuale

La macchina virtuale fornita da Cloud9 ha come sistema operativo Ubuntu ed è possibile amministrarla attraverso un terminale. Nel caso dello sviluppo di GoBox sono stati installati i seguenti software:

- Nginx: un server HTTP con possibilità di *reverse-proxy*, utilizzato per poter separare le richieste API verso il server realizzato in Go da quelle relative ai file statici della WebApp, che sono vengono delegate al server istanziato da Grunt secondo la configurazione creata da Yeoman (Ulteriori informazioni nella sezione WebApp);
- MariaDB: un DBMS (Database Management System) relazionale open source che sopporta il linguaggio SQL;
- Go: il compilatore di Go e tutti gli strumenti inclusi;

- NodeJS: utilizzato per l'esecuzione di Grunt, Bower e Yeoman (Ulteriori informazioni nella sezione WebApp);

L'editor supporta il *syntax-highlighting* (caratteristica che modifica il colore delle parole chiave semplificandone la lettura del codice) di molti linguaggi, tra cui Go. Inoltre Cloud9 può anche essere utilizzato per l'hosting di sviluppo, dato che alla macchina virtuale è associato un URL pubblico.

Gestione del Codice con Git

Il codice di tutti i componenti è stato costantemente versionato attraverso lo strumento Git.
Git espone moltissime funzioni, di cui le principali utilizzate per questo progetto sono il versionamento e la divisione di *features* (funzioni) in "branch".

GitHub

Logo GitHub

Inoltre, è stato associato ad ogni componente della piattaforma un repository pubblico su Github.com, un servizio che offre la possibilità di *hostare* repository online.

I repository creati sono:

- Repository WebApp: https://github.com/simonedegiacomi/goboxwebapp
- Repository Server: https://github.com/simonedegiacomi/goboxserver
- Repository GoBoxJavaApi: https://github.com/simonedegiacomi/goboxwebjavaapi
- Repository Client: https://github.com/simonedegiacomi/goboxclient
- Repository GoBoxPhoto: https://github.com/simonedegiacomi/goboxphoto

Server

Scelte di progettazione

Durante la scelta del linguaggio da utilizzare per l'implementazione del server sono stati prese in considerazione le seguenti necessità:

- capacità di gestire un alto numero di connessioni, poichè ogni singolo client e storage richiede più connessioni contemporanee;
- il server deve essere sempre in esecuzione, per poter far comunicare i diversi client tra di loro;

Il linguaggio scelto è stato Go.

Linguaggio di programmazione Go

Go (conosciuto anche come GoLang) è un linguaggio di programmazione open source creato da Google nel 2007. E' compilato e fornisce strutture per facilitare la gestione del codice concorrente, oltre a ad essere fornito con una libreria standard completa e ben documentata.

Gopher, la mascotte di Go

Sintassi

La sintassi di Go punta sulla semplicità, utilizzando un numero ristretto di parole chiave. E' staticamente tipizzato, per garantire concretezza e sicurezza ma riprende anche pattern di programmazione solitamente presenti nei linguaggi dinamici, prendendo le caratteristiche migliori di entrambi i mondi. Per esempio, non è necessario specificare il tipo di una variabile quando quest'ultimo è esplicito, ma viene segnalato l'errore se ne viene successivamente fatto un utilizzo errato.

Codice di esempio:

```
import ("strings") // Importazione di strings, dalla libreria standard

func getN () int { // Dichiarazione di una funzione che ritorna un intero
	return 5
}

func main () { // Dichiarazione della funzione main
	numero := getN() // Esempio di assegnamento implicito
	somma := numero + 2 // Anche la somma è un assegnamento implicito
	strings.ToLower(numero) // Errore durante la compilazione
}
```

Come è possibile notare dal semplice esempio, Go non utilizza il carattere punto e virgola (';') per indicare la fine di una riga.
Inoltre, le definizioni delle variabili (quando il tipo non è esplicito) e degli argomenti delle funzioni, per risultare di più facile lettura, sono in ordine opposto rispetto ai linguaggi più comuni: troveremo quindi il nome della variabile o della funzione precedere il tipo.
Un'altra differenza rispetto alla maggior parte degli altri linguaggi, riguarda il fatto che le funzioni possono ritornare più di un valore.
Infine in Go, l'importazione di librerie che non vengono poi utilizzate nel codice portano come risultato ad un errore di compilazione.

Gestione degli errori

Go non ha il concetto di eccezione, ma sfrutta la sua caratteristica che permette alle funzioni di poter ritornare più di un valore. Infatti, i metodi o le funzioni che possono dare errori durante l'esecuzione, ritornano una variabile (solitamente come ultimo valore) di tipo Error.

Programmazione a oggetti

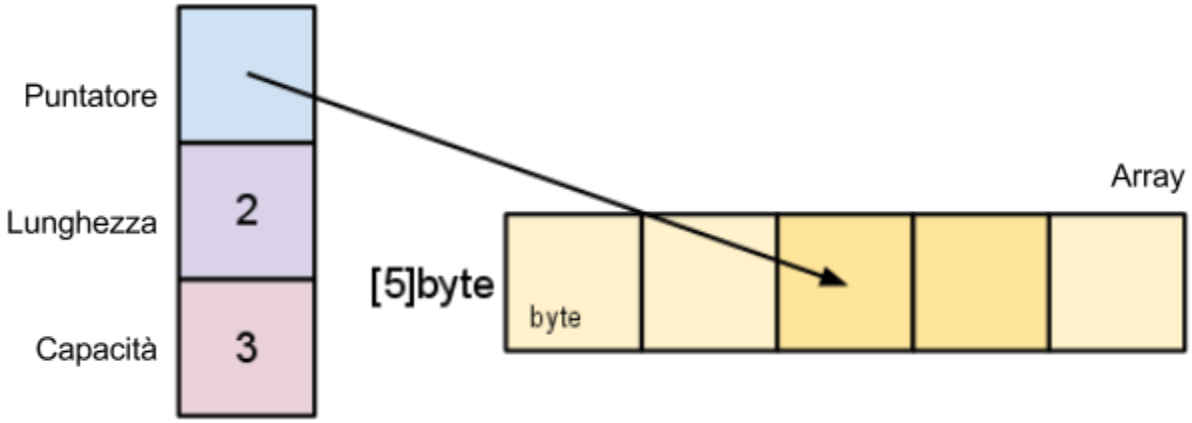
A differenza di quasi tutti i linguaggi di programmazione, non è semplice affermare se Go è un linguaggio orientato alla programmazione ad oggetti: infatti, come specificato sul sito ufficiale di Go, è possibile dichiarare delle strutture, a cui si possono "attaccare" dei metodi. Per associare un metodo ad una struttura, è sufficiente specificare il nome della struttura prima della definizione del metodo
Sono presenti anche delle interfacce, seppur non con lo stesso significato delle interfacce di Java: in Go, per implementare un'interfaccia basta definire un metodo che abbia certi parametri e che ritorni specifici valori. È quindi possibile implementare delle interfacce involontariamente, risultato non sempre desiderato.
Infine, è possibile ottenere un risultato simile agli indicatori di visibilità di Java, seguendo una regola che consiste nel chiamare i metodi privati (ovvero utilizzabili solamente all'interno del package in cui sono dichiarati) con la prima lettera minuscola, ponendo la prima lettera maiuscola per i metodi pubblici.

Vettori strutture dati

In Go, quando parliamo di vettori possiamo riferirci a due strutture dati: gli array e gli slice.

Gli array sono dei vettori statici, la cui lunghezza ne definisce il tipo: per esempio, un array di 4 interi è di tipo int[4], mentre un array di 5 elementi è di tipo int[5]. A differenza di Java, come in C, gli array non necessitano di alcun tipo di inizializzazione, ma è sufficiente la loro definizione. Tutti i valori degli array avranno valore *zeroed*. Infine, in Go una variabile di tipo array non è un puntatore al primo elemento dell'array, ma bensì l'intero array, causandone l'intera copia di esso nel caso di un assegnamento o passaggio come parametro di una funzione (pur essendo comunque possibile passare un puntatore all'intero array).

Difficilmente gli array vengono utilizzati nella programmazione con Go, spesso al loro posto troveremo gli slice. Il tipo di uno slice è definito solamente dal tipo di dato contenuto, non dal numero di valori contenuti. Internamente uno slice è una struttura che descrive un segmento di un'array. Questa struttura consiste in un puntatore all'array che contiene gli effettivi valori, il numero di elementi (lunghezza) e la capacità, ovvero il numero di valori inseribili all'interno del segmento di array. In questo caso è necessario inizializzarlo, utilizzando la funzione make. Questa funzione richiede tre parametri: il tipo, la lunghezza e, facoltivamente, la capacità.



"*Do not communicate by sharing memory; instead, share memory by communicating."*
The Go Blog

Go è fornito con delle ottime strutture per la gestione del codice concorrente che la rendono semplice e prestante. Iniziamo parlando di come è possibile realizzare un thread, chiamato in Go "goroutine": qualsiasi funzione può essere eseguita come goroutine, basta chiamarla utilizzando la parola chiave "go". In questo modo è possibile eseguire la stessa funzione, in diverse parti del codice, a volta nella stessa goruotine da cui viene chiamata e a volte in una goroutine separata.

Per poter far comunicare e sincronizzare più goroutine è possibile utilizzare i canali ("Channels"): strutture paragonabili a dei "tubi", con cui è possibile inviare valori da una parte all'altra. Esistono due tipi di canali, quelli *bufferizzati* (che hanno un buffer), di cui è possibile indicarne la dimensione e quelli senza buffer. La differenza sta nel fatto che quelli senza buffer causano un blocco della goroutine durante l'invio del valore, fino a quando il valore non viene ricevuto (e viceversa). Quando il buffer di un canale bufferizzato viene terminato, il canale ha lo stesso comportamento di quello senza buffer.

L'utilizzo dei canali per la sincronizzazione riprende la citazione sopra riportata, la quale vuole sottolineare la differenza rispetto all' implementazione della sincronizzazione in altri linguaggi (come C o java, in cui vengono condivise strutture di memoria).

Altri strumenti per la gestione del codice concorrente possono essere trovati all'interno del package *sync* della libreria standard.

Strumenti

Quando viene installato il compilatore vengono inclusi diversi strumenti che semplificano lo sviluppo di test e il debugging. I tools sono utilizzabili attraverso la riga di comando, in particolare utilizzando un programma chiamato go, i cui principali argomenti sono:

- go run [file]: Compila ed esegue il file specificato;
- go build: Compila e genera un file eseguibile;
- go test: Esegue la suite dei test;

Database

Il server utilizza un database di tipo SQL, in particolare l'implementazione MariaDB. In questo database vengono salvate le credenziali, le sessioni e i token degli utenti.

Strumenti utilizzati

GoDoc

GoDoc è un sito web che raggruppa e semplifica la consultazione delle librerie per Go. Potremo infatti visualizzare la documentazione della libreria standard, librerie hostate su Github, Bitbucket (simile a Github) e altri servizi. Una caratteristica molto interessante di questo sito è la possibilità di consultare il codice sorgente delle librerie, semplicemente cliccando sul nome della funzione o struttura dati cercata.

SQLx

SQLx è un a libreria Open Source per Go che estende le interfacce della libreria standard per l'utilizzo di database di tipo SQL. Fornisce delle funzioni di utilizzo comune, come per esempio la query di una singola riga.

Negroni

Negroni è una libreria che permette la definizione di middleware per gli handler HTTP (le funzioni che gestiscono le richieste HTTP). Le middleware sono utilizzate per raggruppare le funzioni di cui ogni handler necessità: per esempio, ogni handler, escluso quello per la registrazione, necessità di filtrare le richieste, rifiutando quelle dei client non autorizzati.
Utilizzando le middleware viene introdotto inoltre il concetto del "request lifetime", ovvero il ciclo di vita delle richieste, che inizia con la prima middleware e termina con l'ultima middleware, oppure quando una di esse conclude la richiesta.

Gorilla Toolkit: Mux, Context e WebSocket

Gorilla è un toolkit, un insieme di librerie create per l'implementazione di servizi web in Go. Gli strumenti utilizzate per l'implementazione di GoBox sono Gorilla Mux, Gorilla Context e Gorilla WebSocket.

Gorilla Mux è un *multiplexer* di richieste, che può essere utilizzato per dividere e raggruppare le richieste in base agli URL o ai parametri legati in essi.

La seconda libreria è associabile all'utilizzo di Negroni, poichè fornisce uno strumento (il contesto) con cui è possibile far comunicare più middleware durante il request lifetime. Utilizzando questa libreria la middleware dell'autenticazione potrà salvare nel ciclo di vita della richiesta l'ID o altre informazioni associate all'utente.

Gorilla WebSocket invece è un'implementazione in Go del protocollo WebSocket, parte fondamentale del server.

Implementazione

Il server in Go è stato realizzato cercando di riprendere gli aspetti della programmazione ad oggetti. Sono state quindi create delle strutture per poter dividere il codice, associando metodi ad ognuna di esse.

Gestione degli utenti - Package database

Il package database definisce i modelli dell'utente e delle sessioni e definisce i metodi che semplificano la gestione del database, raggruppando tutte le istruzioni SQL. In questo package

è anche definita la middleware che si occupa di verificare la validità di un token, bloccando le richieste non autorizzate.

Gestione della comunicazione - Package mywebsocket

Il package mywebsocket fornisce le strutture necessarie che implementano la struttura dei messaggi precedentemente illustrata nella sezione relativa alla struttura di comunicazione tra i vari componenti. Nel file manager.go è definita la struttura Manager, che contiene l'Upgrader, il Receiver e una mappa di *listener* (funzioni che vengono eseguite al verificarsi di un evento).

L'Upgrader è una struttura fornita dalla libreria Gorilla Websocket, e non è altro che il componente che trasforma un'iniziale connessione HTTP in una connessione WebSocket (*aggiorna*). La struttura Manager implementa l'interfaccia dell'handler HTTP, e quindi l'autenticazione è eseguita dalla middleware del package db. Una volta *aggiornata* la richiesta, viene avviata una nuova goruotine che si occuperà della lettura dei messaggi ricevuti.

Anche il Receiver è una struttura e possiede un metodo che viene chiamato prima di avviare la goroutine dedicata alla ricezione dei messaggi. La verifica del Receiver cambia in base a chi sta cercando di connettersi: nel caso sia uno storage verifica che non ci siano altri storage autenticati con lo stesso account, mentre nel caso sia un client verifica che lo storage sia connesso.

Comunicazione tra client e storage - Package bridger

In questo package sono contenute le strutture che gestiscono la comunicazione tra i vari client e gli storage. Vengono definite tre strutture all'interno del package: Bridger, Storage e Client.

La struttura Bridger viene creata da una funzione che simula un costruttore (chiamata appunto NewBridger) che richiede come parametri il riferimento al database e il router (della libreria Mux). Esso aggiunge al router gli handler necessari che riceveranno le richieste dei client e degli storage e crea una mappa che assocerà gli ID degli utenti agli storage connessi.

Quando uno storage inizia una connessione WebSocket, viene creata un nuova istanza della struttura Storage, che contiene una lista di Client connessi.

Infine, quando un Client si collega, viene verificato che il relativo storage sia connesso, e in caso positivo il Client viene aggiunto alla relativa lista della struttura Storage.

Gestione dei componenti - Package Server

È stato creato un package che contiene tutte le strutture per la gestione del database, ragruppando i metodi per la gestione degli utenti e delle loro sessioni.
La struttura principale è Server, la quale è formata da un puntatore alla struttura database, al router delle richieste HTTP, alla struttura Bridger e allo slice di byte che contiene la chiave per

la creazione dei JSON Web Tokens. E' presente un metodo NewServer che prepara il server componendo le middleware, istanziando gli handler e aggiungendole al router.

Client

Il client è il software che include sia lo storage sia il programma da installare sui computer secondari dell'utente per poter mantenere sincronizzati i file sul file system in modo automatico. Il programma è pensato per garantire la massima portabilità, ossia poter essere eseguito sul maggior numero di dispositivi. Lavorando costantemente in background, non dovrà intralciare il normale utilizzo del computer, preoccupandosi di non intaccare il sistema con cali prestazionali. Inoltre deve offrire all'utente un'interfaccia semplice e con poche configurazioni, ma comunque offrire molte funzionalità per gli utenti più esperti.

Scelte di progettazione

Per soddisfare i requisiti richiesti dal client è stato scelto di utilizzare il linguaggio di programmazione Java, che permette l'esecuzione sulle principali piattaforme con prestazioni più che accettabili. Per semplificare lo sviluppo, sono stati anche utilizzati vari strumenti e librerie, descritti nelle sezioni successive.

Java

Java è un linguaggio di programmazione ad oggetti creato nel 1995 dalla Sun Microsystem, acquistato successivamente dalla Oracle. Il suo motto è "Write once, run anywhere" (Scrivere una volta per poterlo eseguire ovunque) e punta sul cross-platform. È ad oggi uno dei linguaggi più diffusi tra gli sviluppatori e viene utilizzato per la creazione di programmi sia client che server.

Logo Java

Sintassi

La sua sintassi riprende molti aspetti da C e da C++ e fornisce strutture e parole chiave per la dichiarazione di classi e interfacce per facilitare la programmazione ad oggetti. Supporta l'ereditarietà e l'overriding dei metodi degli oggetti e nelle ultime versioni (attualmente Java 8) sono stati introdotti particolari costrutti che diminuiscono la quantità di codice (come le *lambda expression*). Altro componente molto potente offerto dal linguaggio sono le notazioni, piccole parole aventi una '@' come prefisso, definibili dall'utente, che vengono interpretate da Java per permettere di definire ulteriori proprietà e caratteristiche ai metodi e alle classi.

Compilazione e Java Virtual Machine

Il codice sorgente di un programma realizzato in Java deve essere compilato in un *bytecode*, un codice intermedio tra il sorgente e il codice binario. Questo bytecode non è direttamente eseguibile su un computer, ma deve essere inserito all'interno di una Java Virtual Machine, che interpreta il bytecode. Questo permette al codice Java di essere scritto un'unica volta per poi

essere eseguito ovunque (da qui deriva appunto il suo motto) sia presente la Java Virtual Machine, spostando la realizzazione di codice dipendente dalla piattaforma al momento in cui deve essere eseguito su un'architettura specifica. Questa caratteristica di essere multi-piattaforma si chiama *cross-platform*.

Package

I package in Java vengono utilizzati per raggruppare le varie classi, semplificando l'organizzazione del codice. Questo sistema di gestione si traduce nell'utilizzo di cartelle, nelle quali vengono inseriti i file Java ed ogni sotto cartella definisce un nuovo package. Inoltre i package permettono di evitare le collisioni tra classi aventi lo stesso nome.

Riflessione

La riflessione è una tecnica offerta da Java che permette di analizzare e modificare la struttura del programma in *runtime* (ovvero durante l'esecuzione del programma). Questo permette quindi di caricare classi e librerie non solo all'avvio del programma, ma proprio durante la sua esecuzione. Nonostante esistano molte librerie che permettono un utilizzo semplificato della riflessione, gli strumenti già inclusi risultano sufficienti per le normali operazioni, quali la creazione di istanze di classi caricate attraverso il loro nome (completo di package), l'identificazione di metodi attraverso il loro nome e i tipi di parametri richiesti e la loro esecuzione. Le notazioni vengono anche utilizzate per implementare questa tecnica, in modo da semplificare la ricerca delle classi che le utilizzando attraverso il loro nome.

Strumenti e tecnologie utilizzate

Maven

Logo Maven

Per semplificare lo sviluppo del software è stato utilizzato Maven, un software prodotto da Apache che provvede degli strumenti per i progetti realizzati in Java. Maven introduce il concetto di "fasi" di costruzione (*build life-cycle*, ciclo di vita di costruzione), che vengono eseguite in un preciso ordine. Attraverso la definizione di un file XML, chiamato *pom* (Project Object Model) è possibile personalizzare queste fasi specificando le operazioni necessarie per completare la compilazione attraverso la configurazione e l'inclusione di plug-in. Inoltre è possibile specificare tutte le dipendenze del programma, che verranno automaticamente scaricate da internet.
Maven esegue anche i vari test prima di completare il ciclo di build, che verrà interrotto nell'eventuale fallimento di uno o più test.

JitPack

Come precedentemente introdotto, Maven permette di gestire le dipendenze di un progetto specificando unicamente un identificativo, il quale però deve essere pubblicato su un repository di Maven. Nel caso di gobox, il codice è stato caricato su GitHub, non direttamente utilizzabile

con Maven. Qui entra in gioco JitPack, uno servizio web che permette di trasformare un repository GitHub in un piccolo repository Maven con il proprio progetto.

Google Guava e Gson

Per poter mettere in comunicazione tutti i componenti di GoBox vengono scambiati dei messaggi, codificati in formato JSON. Per poter gestire questo formato all'interno di Java è stata utilizzata una libreria di Google chiamata Gson. Questa libreria, oltre a permettere la creazione e la lettura di oggetti JSON come una semplice mappa chiave-valore, permette la serializzazione (e deserializzazione) di oggetti Java in modo automatico. In questo modo tutte le informazioni scambiate possono essere rappresentate con la definizione di classi. Gson utilizza le notazioni per permettere di specificare il modo in cui ogni attributo deve essere serializzato, oppure per escludere particolari attributi.
Inoltre è stato utilizzato anche l'insieme di libreria Guava, un pacchetto di software open source realizzato da Google.

Log4j

Log4j è una libreria Java sviluppata da Apache per gestire il logging all'interno di programmi. È configurabile attraverso la definizione di un file in formato properties di Java ed è possibile personalizzarlo per mostrare i log più adatti per il tipo di programma. E' possibile visualizzare i messaggi stampandoli sullo standard output, ovvero all'interno della console da cui il programma viene eseguito oppure concatenandoli in un file. Ogni messaggio di log è correlato ad un livello di log (livello di informazione, debug o errore critico), alla data di emissione e all'esatta riga del codice che l'ha generato.

Tika e mp3agic

Tika è sviluppata anch'essa da Apache e permette l'estrazione di *metadata* (informazioni) dai file. Permette inoltre di determinarne il *mime* (una sigla che identifica il tipo e il formato di una file).
La libreria mp3agic, è invece utilizzata per l'estrazione dell'*album art* (l'immagine dall'album musicale) dai file audio di tipo mp3, per la generazione della loro anteprima.

H2

Durante la progettazione dello storage è stata scelto l'utilizzo di un database per l'indicizzazione dei file, ottenendo prestazioni di ricerca e gestione delle informazioni sui file più veloce rispetto alla lettura degli attributi di quest'ultimi. L'utilizzo di un database semplifica inoltre anche le operazioni di condivisione e la gestione del cestino, dato che permette di associare ogni singolo file ad un ID univoco, di semplice generazione. L'idea è ricaduta su un database di tipo relazionale che supportasse il linguaggio SQL e che includesse la possibilità di essere utilizzato al di fuori di un server, ovvero in modalità *embedded*, risultando in un unico file. Inizialmente è

stato preso in considerazione l'utilizzo di SQLite, sostituito poi da H2, che rispetto al suo concorrente ha un supporto maggiore alla sintassi SQL.
H2 è inoltre accompagnato da uno strumento che permette la gestione del database attraverso una console raggiungibile dal browser (ottenendo un risultato simile all'interfaccia di gestione PHPMyAdmin), realizzata creando un piccolo server HTTP locale, utile durante il debug del programma.

ORMLite

ORMLite è una libreria Java, disponibile anche per Android, che permette di utilizzare un database relazionale SQL mappando le istanze di un'entità (in poche parole, una riga di una tabella) in oggetti Java, seguendo l'idea ORM (Object Relational Mapping). Questo sostituisce la scrittura di query con la creazione e configurazione di oggetti (chiamati QueryBuilder) che seguono il pattern Builder. Per poter mappare una "riga" di una tabella SQL, ORMLite si serve delle notazioni di Java e quindi le classi che necessitano di essere serializzate devono essere modificate con la loro aggiunta, per descrivere il modo in cui i singoli attributi dovranno essere serializzati (specificando il nome della colonna, il tipo di colonna, indicandone il valore predefinito, etc..).

NV Websocket Client

NV Websocket Client è una libreria open source che fornisce un'implementazione di un client Websocket in Java, compatibile anche con Android. E' semplice da utilizzare ed è provvista di una buona documentazione, oltre ad essere accompagnata da validi esempi di utilizzo.

JUnit

JUnit è un framework per la definizione di test. I test sono particolari programmi che servono a *testare* specifiche funzioni del programma. Esistono principalmente due tipi di test:

- Unit test: Test di unità che vanno a verificare il corretto funzionamento di componenti isolati, come una classe, che non dipendono da risorse esterne. Questi test sono di facile scrittura e esecuzione;
- Integration test: Test di integrazione che vanno a verificare il corretto risultato portato dall'unione di più classi e dall'utilizzo di risorse esterne come file e database. Sia la scrittura che l'esecuzione di questi test risulta più complessa, soprattutto durante la preparazione dell'*environment* (ambiente) per la loro esecuzione.

Implementazione

Dopo la prima fase di progettazione caratterizzata dallo studio delle necessità del programma si è scelto di dividere il client in due progetti: il primo dedicato alla definizione delle classi e delle interfacce, utilizzabile come client API, il secondo per definire l'implementazione effettiva del client. Ogni progetto ha un file *Project Object Model* di Maven e un repository Git dedicato.

Progetto GoBoxJavaAPI - Definizione delle strutture

Come anticipato, il primo progetto contiene le definizioni delle interfacce e delle classi utilizzate all'interno di GoBox. Inoltre, è inclusa anche un'implementazione di un client che offre le funzioni base per poter gestire uno storage. Questo progetto Maven è pensato per poter essere importato all'interno di altri progetti, come GoBoxClient e l'applicazione nativa Android, evitando una ridondanza del codice.

Implementazione dei messaggi - Classe MyWSClient

Come già visto nella sezione dedicata ai protocolli utilizzati e alla relativa implementazione utilizzata all'interno di questa piattaforma, è stato introdotto il concetto di messaggio e di richiesta inviati tramite WebSocket. Le classi che si occupano di realizzare questa funzione in Java sono contenute all'interno del package myws, e la principale è MyWSClient.
Questa classe è provvista di un costruttore che riceve come parametri l'indirizzo del server WebSocket a cui connettersi e un'istanza di un'implementazione dell'interfaccia AbstractExecutorService, che verrà utilizzata per la schedulazione dell'esecuzione dei listener al momento della ricezione dei messaggi o delle richieste. Una volta creato l'oggetto, è possibile aggiungere i listener, sia per gli eventi sia per le richieste. I listener possono essere aggiunti utilizzando uno specifico metodo e specificando il nome del messaggio e il listener, oppure solamente il listener, che dovrà essere definito utilizzando una delle apposite annotazioni: WSQuery o WSEvent. Quando un listener viene aggiunto utilizzando le annotazioni, viene utilizzata la riflessione di Java per poter ricavare il nome del messaggio. Una volta aggiunti tutti i listener necessari, si effettua la connessione utilizzando il metodo *connect*. Per poter utilizzare questa classe su Android è sufficiente chiamare i metodo connect e shutdown nel momento opportuno, seguendo il life-cycle delle activity o dei service.

Un'ultima nota su questa classe riguarda ii due modi con cui si possono inviare i messaggi di tipo richiesta:

- modalità sincrona: utilizzando il metodo che richiede come parametri il nome del messaggio, l'oggetto JSON da inserire nella proprietà "data" e la callback che verrà eseguita una volta ricevuta la risposta;
- modalità asincrona: utilizzando il metodo che non richiede la callback come parametro, ma che ritorna un oggetto di tipo FutureTask. In questo modo è possibile effettuare la query, ottenere il FutureTask come risultato e chiamare il metodo *get* che bloccherà il thread finché la risposta non sarà ricevuta;

Rappresentazione dei file - Classe GBFile

La classe GBFile è la classe utilizzata per rappresentare un file all'interno di gobox. La classe contiene diversi attributi tra cui:

- il nome del file;
- una flag che definisce se il file è una cartella;
- la dimensione espressa in byte, se non è una cartella;

- la data di creazione e di ultima modifica (numero di secondi dal 1 Gennaio 1970 UTC);
- lista di “figli” se è una cartella, ovvero i file contenuti in essa;
- lista di altri GBFile che descrivono il path (percorso);
- un ID per identificare il file e il rispettivo ID del padre (la root ha un ID speciale con valore pari a 1 e 0 come ID del padre);

Questa classe è definita anche attraverso l’utilizzo di notazioni di ORMLite e Gson, per la rispettiva serializzazione all’interno del database e della rappresentazione JSON.

Una volta che l’oggetto viene serializzato in JSON risulta simile al seguente:

```
{
	"ID": 2,
	"fatherID": 1,
	"name": "Documenti",
	"children": [
		{
			"ID": 3,
			"fatherID": 2,
			"name": "relazione.pdf",
			"mime": "application/pdf",
			"size": 2048,
			"path": [
				{
					"ID": 2
				}
			]
			"isDirectory": false
		}
	],
	"path": [],
	"isDirectory": true
}
```

NB: Per evitare problemi derivati dalla presenza di una dipendenza circolare nella rappresentazione JSON, precisamente per quanto riguarda la lista di figli e la path, queste collezioni contengono solamente l’ID del file.

Gestione degli eventi e della cronologia - Classe SyncEvent

Ogni operazione eseguita dai client su un file causa la generazione, da parte dello storage, di una nuova notifica di sincronizzazione. Quest’ultima è rappresentata dalla classe SyncEvent, la quale contiene:

- Il tipo di evento, come creazione, modifica, eliminazione, copia, condivisione etc… ;
- Il file associato all’evento;
- Il file allo stato precedente dell’evento, se presente (come per esempio nel caso del movimento di un file);
- data dell’evento espressa in millisecondi;

- ID identificativo dell'evento;

Esempio della serializzazione in JSON di un evento:

```
{
	"ID": 5,
	"date": 45564564572,
	"Kind": "FILE_MOVED"
	"before": {
		"ID": 3,
		"fatherID": 1,
		"name": "documento.pdf",
		"lastUpdate": 45564564572,
		"mime": "application/pdf",
		"size": 1024
		"path": []
	},
	"file": {
		"ID": 3,
		"fatherID": 2,
		"name": "documento.pdf",
		"lastUpdate": 45564564572,
		"mime": "application/pdf",
		"size": 1024,
		"children": [
			{
				"ID": 3
			}
		]
		"path": [
			{
				"ID": 2,
			}
		]
	}
}
```

Ogni evento generato viene infine inserito in un'apposita tabella del database, per poter fornire il servizio della cronologia.

Gestione degli indirizzi - Singleton URLBuilder

URLBuilder è una classe che segue il pattern singleton, che consiste nel mantenere un'unica istanza comune durante tutta l'esecuzione del programma. In Java è possibile realizzare un Singleton ponendo il costruttore privato ed esponendo un metodo statico che permette di ottenere la singola istanza. Questa classe è utilizzata per la gestione degli URL, in modo da evitare l'inserimento di indirizzi *hard-coded* nel codice.
Quando l'oggetto viene creato non contiene nessun URL, ed è possibile caricare una o più liste di URL chiamando il metodo load. Se il metodo per ottenere un indirizzo viene chiamato prima

ancora di caricare una lista di indirizzi, l'oggetto prova a caricare la lista da un file predefinito, contenuto nella cartella dedicata alle risorse (creata e gestita da Maven).

Autenticazione - Classe GBAuth

Come si intuisce dal nome, questa classe contiene le informazioni dell'utente relative alle credenziali e contiene il token di autenticazione. Fornisce inoltre i metodi necessari per il controllo della validità del token e per l'esecuzione della procedura di login e logout. Questi metodi utilizzano URLBuilder per ottenere gli indirizzi a cui fare le richieste, le quali verranno eseguite nello stesso thread da cui sono invocate, rendendo i metodi bloccanti.
Poichè il token di autenticazione cambia durante l'esecuzione del programma, è possibile aggiungere all'oggetto un listener, che verrà chiamato ad ogni cambiamento del token, permettendo di salvarlo.

Utilizzo dello storage - L'interfaccia GBClient e la classe StandardGBClient

L'interfaccia del client definisce i metodi fondamentali che ogni client deve implementare per avere una gestione basica dei file all'interno del proprio storage. All'interno dello stesso package viene inserita un'implementazione, chiamata StandardGBClient. Il costruttore di StandardGBClient richiede un oggetto GBAuth, che utilizzerà per autenticarsi con il server. Questo client crea un canale WebSocket con il server principale e rimane in ascolto di SyncEvent. E' possibile aggiungere uno o più listener che verranno chiamati alla ricezione di un nuovo SyncEvent. Dato che ogni volta che viene eseguita un'operazione lo storage invia la notifica anche al client che l'ha causata, è possibile abilitare un filtro, che elimina l'effetto echo.

Gestione delle modalità di connessione - Classe TransferProfile

La classe TranferProfile è utilizzata dallo StandardGBClient, per gestire le diverse modalità di connessione. Il client utilizza un oggetto di questo tipo per ottenere gli URL a cui eseguire le richieste per il trasferimento dei file. Per funzionare correttamente, il client deve chiamare un suo specifico metodo quando completa il cambio di modalità, specificando il nuovo indirizzo di destinazione, per poter generare i nuovi indirizzi.

Progetto GoBoxClient - Implementazione del client per pc

Questo progetto Maven dipende dal progetto GoBoxJavaAPI, importato utilizzando JitPack. GoBoxClient è il progetto che implementa effettivamente il programma che viene eseguito sui computer, sia come client che come storage. Per questo motivo deve poter essere adattato al computer su cui viene eseguito e deve adeguarsi alle richieste dell'utente, requisito soddisfatto dalla presenza di un file di configurazione.

Configurazione dell'esecuzione - Classe Config

La classe Config è un singleton che include la configurazione e le preferenze che il programma deve utilizzare durante la sua esecuzione. Alla creazione dell'oggetto non contiene nessuna proprietà, che possono però essere aggiunte caricandole da un file oppure chiamando lo

specifico metodo. Il metodo utilizzato per ottenere una proprietà permette anche di specificare un valore di default, che verrà salvato e ritornato nel caso in cui non sia incluso nella collezione delle proprietà. Il file contenente le proprietà della configurazione deve essere salvato nel formato properties di Java.

Attributi configurabili

Nel file di configurazione possono essere specificate varie proprietà, tra cui:

- path: Il percorso della cartella che conterrà tutti i file, Il suo valore predefinito è un percorso relativo “files/”;
- dbPath: Indirizzo della cartella in cui verranno creati i file del database, utilizzato solo nello storage. Valore di default “db/”;
- cache: Il percorso della cartella dedicata alle *thumbnail*, le anteprime dei file. Questa proprietà viene utilizzata solamente nello storage, ed il suo valore predefinito è “caches/”;
- trash: Il percorso della cartella dedicata ai file cestinati. Anche questa proprietà viene ignorata sui client, e il suo valore predefinito è “trash/”;
- useProxy: Flag che indica se utilizzare un server proxy;
- proxyIP: Indirizzo del server proxy HTTP da utilizzare;
- proxyPort: Porta del server proxy HTTP da utilizzare
- directConnection: Indirizzo su cui lo storage avvierà il server HTTPS per la direct connection, Il valore predefinito è “0.0.0.0”;
- directConnectionPort: Porta ascoltata dal server HTTPS per la connessione diretta;

Gestione della sincronizzazione - Classe Sync

La classe Sync fornisce il servizio di sincronizzazione dei file del client con quelli dello storage. Il suo costruttore richiede come parametri il client da utilizzare (ovvero un’istanza di GBClient) e un’istanza dell’interfaccia MyFileSystemListener. All’avvio viene eseguita una fase di allineamento con i file dello storage, attraverso una funzione ricorsiva che parte analizzando la cartella contenente tutti i file.

Sync ottiene le informazioni della root utilizzando l’apposito metodo del client, poi confronta ogni “figlio” con i file locali:

- se il file è una cartella, la funzione richiama sè stessa specificando come parametro la cartella che sta analizzando;
- se è un file presente sul client, ma assente nello storage, provvede a caricare il nuovo file;
- se è un file presente sullo storage ma assente sul client lo scarica:
- se un file è presente sia nello storage sia nel client, ma hanno differenti date di ultima modifica, viene trasferito il file più recente;

Una volta che i file sono allineati, viene aggiunto un nuovo listener al FileSystemWatcher, per poter rilevare modifiche e aggiunte locali, e viene aggiunto un SyncEventListener al client, per ricevere modifiche remote (apportate da altri client).

Rilevamento operazioni sui file - Interfaccia MyFileSystemWatcher

La rilevazione degli eventi sui file si è rilevata problematica, specialmente a causa del tentativo di mantenere il software cross-platform. Sfortunatamente non esiste un protocollo o una API standard con cui è possibile chiedere al sistema operativo di essere notificati quando il file system viene aggiornato. Java fornisce degli strumenti per il monitoraggio del file system, nel package nio (New Input Output), che automaticamente utilizzano i meccanismi di monitoraggio forniti dai sistemi operativi (come inotify nel caso di Linux), ricadendo su un polling (ovvero un continuo controllo dello stato dei file) quando il wrapper non è implementato oppure se il sistema di monitoraggio non è presente.
Nonostante questi strumenti, il loro risultato non è equivalente se testato su più piattaforme: per esempio, su Mac Os X, quando viene spostato un file, riceviamo l'evento di eliminazione del file, creazione del nuovo file e modifica della cartella padre, mentre su Linux otteniamo prima l'evento di creazione del file nella nuova posizione, e successivamente l'evento di eliminazione nella posizione precedente. Questi problemi sono stati risolti implementando un ordinamento degli eventi ricevuti, in modo da avere un livello di astrazione dal file system.
Se ci fossero problemi su una particolare piattaforma o su un dispositivo specifico, dato che MyFileSystemWatcher è un'interfaccia, sarà sufficiente fornire una nuova implementazione e utilizzare quest'ultima all'interno del programma, senza modificare altro codice.

Realizzazione dello storage - Classe storage e interfaccia GBModule

La classe storage si occupa di fornire il servizio di memorizzazione dei file. Il suo costruttore richiede semplicemente un GBAuth, che verrà utilizzato per connettersi ed autenticarsi al server principale, dato che questo oggetto gestisce una connessione WebSocket propria.
Durante la progettazione di questo componente, è stato scelto di non implementare le funzioni dello storage (la gestione dei file), ma di separare ogni attività in un modulo. Per realizzare questo approccio è stata creata un'interfaccia (chiamata GBModule) che ogni modulo deve implementare, che cerca di imitare la gestione dei life-cycle delle activity e dei componenti delle applicazioni native Android (argomento approfondito nella sezione dedicata). Tutti i componenti vengono poi caricati dinamicamente utilizzando la riflessione, cercando all'interno della classpath tutte le implementazioni di GBModule.
Come risultato, oltre ad avere una buona divisione del codice, abbiamo la possibilità di abilitare e disabilitare i moduli a nostro piacimento, permettendo anche ad altri sviluppatori di definire i loro moduli, aggiungendo nuove funzionalità allo storage senza modificare altro codice.

Per definire un nuovo modulo è sufficiente creare una nuova classe che implementi l'interfaccia GBModule e definirne un costruttore senza alcun parametro. Per aggiungere funzionalità al modulo, possono essere definiti due tipi di listener, sotto forma metodi pubblici:

- un listener per le richieste WebSocket, che deve ricevere come parametro un oggetto di tipo JsonElement (la richiesta) e ritornare sempre un altro JsonElement (la risposta);
- un listener per le richieste HTTPS, che deve ricevere un singolo parametro di tipo HttpExchange e non deve ritornare niente;

È possibile definire un qualsiasi numero di metodi di entrambi i tipi.

Inoltre, ogni modulo deve implementare due metodi: onAttach, chiamato dallo storage quando i suoi listener vengono abilitati e onDetach, quando i listener vengono disabilitati.
Attraverso il metodo onAttach il componente riceve:

- la connessione al database, con la quale può creare tabelle dedicate o visualizzare quelle utilizzate da altri componenti;
- il riferimento al FileSystemWatcher, che può utilizzare per registrare un listener personalizzato o per ignorare un file;
- il riferimento al client intero (oggetto analizzato a breve);
- la configurazione globale del programma;
- una configurazione dedicata al singolo componente (riprende le SharedPreferences di Android), che il componente può utilizzare come preferisce, per salvare piccole proprietà;
  NB: *Queste proprietà sono quelle personalizzabili dall'utente attraverso la modifica del file di configurazione, per altre informazioni o proprietà che non devono essere modificate dall'utente è preferibile l'utilizzo di una specifica tabella del database.*

Una volta concluso l'approfondimento dei moduli, è necessario parlare di come è implementata la sincronizzazione dei file locali del computer su cui viene eseguito lo storage. Pur essendo possibile utilizzare lo stesso codice del client, quindi utilizzando un oggetto Sync con uno StandardGBClient, è stato creato un nuovo client, chiamato InternalClient, che viene passato all'oggetto Sync. Questo perchè se fosse stato utilizzato un client standard, lo storage avrebbe avuto una connessione diretta con sé stesso, sprecando risorse superflue. I metodi del client interno sono implementati utilizzando direttamente il database dello storage, in modo da risparmiare risorse. Dato che il client interno implementa l'interfaccia GBClient, è possibile passare questo client all'oggetto Sync, ottenendo quindi la sincronizzazione del file system su computer storage.

Connessione diretta - Classe HttpsStorageServer e generazione dei certificati

Lo storage deve provvedere la modalità di connessione diretta. Questa avviene attraverso il protocollo HTTPS, che deve quindi essere creato e gestito dall'oggetto storage. Per semplificare la gestione di quest'ultimo, è stato creata la classe HttpsStorageServer, la quale gestisce la generazione del certificato, la gestione delle richieste di tipo CORS (le quali verranno chiarite a breve) e i token per l'autenticazione delle richieste in modalità diretta.

Generazione del certificato

Lo storage genera un certificato HTTPS al primo avvio, per poi riutilizzare sempre lo stesso. Per semplificare la gestione e la creazione dei certificati, HttpsStorageServer utilizza una classe chiamata HttpsCertificateGenerator. Questa utility è in grado di caricare un certificato precedentemente generato e salvato in un *keystore*, oppure di crearne uno nuovo e salvarlo.

Gestione delle richieste CORS

Il CORS (Cross-Origin Resource Sharing) è un meccanismo utilizzato per permettere ai siti web di fare delle richieste HTTP ad un server di dominio diverso da quello che ha servito la pagina web. I browser permettono questo tipo di richieste per contenuti multimediali (immagini e video), fogli di stile (CSS) e script Javascript, ma ostacolano, per motivi di sicurezza, le richieste di tipo Ajax. Nel caso della connessione diretta, necessitiamo proprio di queste richieste, perchè la web app viene fornita dai server di gobox, mentre la richieste HTTPS per i trasferimenti dei file sono rivolte allo storage (un dominio certamente diverso). Per poter permette questo tipo di richieste, il server di diverso dominio, deve rispondere alle richieste con dei particolari Header che indicano i domini di origine permessi (Access-Control-Allow-Origin) e i metodi utilizzabili (Access-Control-Allow-Methods).
Questo viene implementato attraverso un oggetto che permette di creare delle middleware, ovvero degli handler che vengono eseguiti prima dell'handler finale (nel caso di questo client, i listener dei componenti).

Generazione anteprime - Interfaccia Previewer e implementazioni

Nella WebApp è possibile navigare tra i propri file visualizzandoli sotto forma di lista o di griglia. Quando la visualizzazione della griglia è attiva, delle piccole anteprime (thumbnail) sono visualizzate, precisamente nel caso di:

- immagini è visualizzata un'immagine di ridotta qualità e dimensione;
- documenti PDF viene proposta un'immagine della prima pagina;
- video è mostrata una GIF che rappresenta i primi secondi del filmato;
- canzone, se presente, viene mostrato l'album art;

Per implementare questa funzionalità è stata definita un interfaccia chiamata Previewer, che dichiara i principali metodi di cui un generatore di anteprime deve disporre: un metodo che permette di chiedere al generatore se è in grado di gestire uno specifico tipo di file, e l'effettivo metodo che genera l'anteprima. In questo modo possono essere aggiunti nuovi generatori (per esempio di documenti word con estensione .doc) senza modificare altre parti di codice.

Inoltre, per diminuire il tempo della popolazione della griglia dei file nella WebApp, le anteprime vengono salvate in una cache, per evitare rallentamenti dovuti alla loro generazione.

WebApp

La WebApp richiedeva di poter essere eseguita ovviamente all'interno di un browser, e quindi è stata realizzata utilizzando le tecnologie del web, Javascript come linguaggio di programmazione, HTML per la definizione del layout e CSS per la componente grafica. Un requisito fondamentale è il design responsive che adatta la pagina al tipo di dispositivo su cui viene visualizzata.

Scelte di progettazione

AngularJS

AngularJS è un framework Javascript open source mantenuto da Google utilizzato per la realizzazione di applicazioni web in pagina singola, che segue il pattern MVC. È caratterizzato dall'utilizzo del pattern Dependency Injector, che descrive il modo in cui l'importazione delle dipendenze possa essere effettuata. Nel caso di Angular questo si applica all'importazione di Service e altri oggetti all'interno di altri service, controller o provider.

Service

I Service in Angular sono oggetti che si occupano di raggruppare la logica di una funzione richiesta dall'applicazione, come la comunicazione con un server esterno o la memorizzazione di oggetti all'interno del local storage. Ogni service è un singleton e viene istanziato solo all'effettivo utilizzo (*lazily initialization* pattern).

Controller

Un controller in Angular viene definito attraverso la definizione di una funzione costruttore, il cui scopo è quello di inizializzare e aggiornare lo $scope, l'oggetto che permette di far comunicare la view con la logica del controller.

Strumenti utilizzati

Angular Material

Angular Material è un framework basato su Angular che permette di realizzare una web app in pieno Material Design utilizzando dei semplici tag HTML e classi CSS. Anche questo framework è open source ed è mantenuto da un team di Google. L'utilizzo di questo framework ha semplificato la gestione della grafica e la realizzazione dell'effetto responsive.

UI-Router

Per associare una view ad ogni controller è stata utilizzata la libreria ui-router che ne permette l'associazione attraverso la definizione di nuovi stati. Ogni stato possiede:

- un nome, utilizzato per riferirsi allo stato stesso all'interno dell'applicazione;
- un indirizzo, il quale può contenere anche dei parametri, che permette di identificare lo stato dell'applicazione.
  NB: La definizione di un indirizzo permette di poter raggiungere lo stato aprendo il browser direttamente all'indirizzo specifico.
- il nome del controller;
- il nome del file HTML della view;
- altri parametri aggiuntivi, utili per gestire l'autenticazione e l'autorizzazione;

Strumenti di sviluppo

Grunt

Grunt è un Task Runner, un particolare strumento che permette di automatizzare azioni ripetitive, come la minimizzazione dei file Javascript, l'esecuzione degli unit test etc...
Questo strumento è realizzato in Javascript e viene eseguito al di fuori del browser attraverso NodeJS.
Grunt viene configurato attraverso un apposito file chiamato Gruntfile.json in cui possono essere importati e personalizzati task già esistenti, oppure è possibile crearne di nuovi.

Bower

Bower è un package manager utile per gestire le dipendenze Javascript e CSS. Anch'esso è uno strumento realizzato attraverso NodeJS e viene configurato tramite un file chiamato bower.json. La struttura di questo file è molto semplice, si tratta di un semplice oggetto JSON in cui vengono elencate le librerie utilizzate nel progetto. Ogni libreria può essere importata come libreria di development, la quale verrà ignorata durante la minimizzazione del progetto, oppure come dipendenza vera e propria del progetto. Per ogni libreria viene anche specificata la versione di riferimento. Una volta definito il file basterà chiamare bower da terminale, il quale provvederà a scaricare da internet le librerie necessarie, che si troveranno in una cartella chiamata "bower_components".

Yeoman

Yeoman è uno strumento che semplifica la configurazione di Grunt e la creazione della struttura del progetto, ovvero le effettive cartelle e la gestione dei file del codice sorgente. Yeoman sopporta molti tipi di progetti, quindi è necessario scaricare l'apposito *plug-in* per Angular. Una volta scaricato basta chiamarlo da terminale specificando la nostra intenzione:
Digitando 'yeoman angular' ci verranno chieste delle informazioni a proposito del progetto, come il nome e le dipendenze comuni (Bootstrap, AngularResources, etc…).

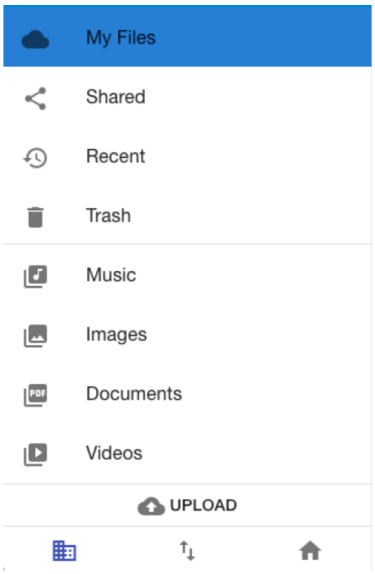
Una volta completata la procedura di configurazione Yeoman creerà automaticamente il file di configurazione per Bower e Grunt.

Implementazione

Per realizzare la WebApp è stato definito un Service che si occupa di fornire la stessa comunicazione WebSocket basata sui messaggi e sulle richieste. In questo caso non è necessario importare una libreria, dato che è stata utilizzata direttamente l'API fornita dal browser.

Una volta definito il service che permette la comunicazione con il server principale, è stato creato il service GBClient, che espone tutti i metodi necessari per poter eseguire le operazioni di gestione dei file presenti sullo storage.

Infine sono stati definiti vari controller, uno per ogni *stato* dell'applicazione. L'utente può cambiare stato semplicemente utilizzando la Sidenav (barra laterale) come mostrato nell'immagine a lato.



Sidenav

Accesso e amministrazione - Login e Settings

Nello stato login viene mostrata la view di login, in cui è possibile accedere con le proprie credenziali oppure creare un nuovo account. Una volta inserite le informazioni di accesso, viene utilizzato il Service GoBoxAuth per eseguire il login comunicando con il server.

Visualizzazione dei file - FileList

Questo stato mostra nella view una lista, ed è appunto l'elenco dei file contenuti nella cartella correntemente visualizzata. Questo è realizzato attraverso un metodo di GoBoxClient che permette di ottenere informazioni sui file, che nel caso di cartelle fornisce l'elenco dei file contenuti.
Per poter fornire gli strumenti di amministrazione dei file è stato creato un Service Clipboard, il quale permette di memorizzare i file selezionati, per poi eseguire l'operazione richiesta dall'utente (come per esempio la navigazione in altre cartelle per la successiva copia dei file).

Condivisione, Ricerca e Cronologia - Shared, Filter e Recent

Questi tre stati hanno un'implementazione molto simile, dato che mostrano tutti una lista che viene popolata da un vettore fornito da uno dei metodi del GoBoxClient. L'unica differenza riguarda la view, che si adatta fornendo gli strumenti disponibili.

Progressive Web App

La Progressive Web App è un'applicazione ibrida, realizzata con gli strumenti del web (HTML, CSS e JS) e alcune nuove tecnologie. Attraverso la definizione di un file Manifest nel formato JSON, in cui devono essere raccolte le caratteristiche dell'applicazione, è possibile trasformare una WebApp o un qualsiasi sito web in un'applicazione per smartphone.
Le caratteristiche da inserire nel file Manifest sono:

- un vettore di icone da associare all'applicazione (le icone devono avere diverse dimensioni, per poter meglio adattarsi alle varie definizioni degli schermi dei dispositivi);
- il nome dell'applicazione;
- il modo in cui l'applicazione verrà mostrata (schermo intero, all'interno del browser, etc…);
- l'indirizzo che deve essere caricato al primo avvio;
- il colore dominante del tema dell'applicazione, per modificare gli elementi del sistema avendo come effetto una miglior integrazione;

Esempio di file Manifest:

```
{
 "name": "GoBox WebApp",
 "short_name": "GoBox",
 "icons": [{
  "src": "images/icons/icon-128x128.png",
   "sizes": "128x128",
   "type": "image/png"
  }],
 "start_url": "https://gobox-simonedegiacomi.c9users.io/dev/#/files/1",
 "display": "standalone",
 "background_color": "#3E4EB8",
 "theme_color": "#2F3BA2"
}
```

Applicazione Android

Logo Android Studio

Per la realizzazione dell'applicazione Android relativa al backup automatico delle fotografie è stato utilizzato l'ambiente di sviluppo ufficiale Android Studio e il software di emulazione Genymotion.

Sviluppo Android

Lo sviluppo di un'applicazione nativa risulta più impegnativo rispetto alla realizzazione di una WebApp. Pur portando meno benefici, poichè è necessario realizzare un'applicazione per ogni piattaforma (obbligando inoltre l'utente ad installare un nuovo software), è a volte necessaria, a causa delle *ancora* limitate capacità dei browser, come per esempio il monitoraggio della galleria del dispositivo.

Android Manifest

L'Android Manifest è un file di tipo XML nel quale vengono incluse le informazioni principali che servono a descrivere la propria applicazione. Questo file viene consultato dal sistema operativo Android durante l'installazione e l'esecuzione dell'applicazione.

Le principali informazioni che devono essere inserite nel file Manifest sono:

- le Activity: Nel file Manifest devono essere elencate tutte le activity dell'applicazione, specificando nome e identificando quella principale;
- i permessi: Devono essere inclusi tutti i permessi di cui l'applicazione necessita, come la possibilità di utilizzare internet o la visualizzare la galleria dell'utente;
- i Broadcast Receiver: Sono particolari classi i cui specifici metodi vengono chiamati durante la ricezione di eventi generati da Android o da altre applicazioni;

Intent

Gli intent (letteralmente tradotto con "intento") sono delle operazioni che le applicazioni possono creare e inviare. Ogni applicazione può rimanere in ascolto di un determinato Intent, per poter eseguire del codice durante un preciso evento. Uno degli Intent più utilizzati è quello che viene generato per la visualizzazione di un'activity. Esempi di Intent possono essere il completamento della procedura di avvio dello smartphone, la riproduzione di una canzone o la creazione di una notifica.

Activity

Un'activity è la schermata con cui l'utente interagisce con l'applicazione. Per creare un'activity è necessario creare una nuova classe che estenda la classe Activity. Ad ogni activity è possibile associare un file XML per descrivere il layout dell'interfaccia utente. Infine l'activity deve essere registrata all'interno del file AndroidManifest.

Parte fondamentale della creazione di un'activity riguarda il suo *life-cycle*, ovvero le varie fasi di esecuzione che attraverserà. Android definisce queste fasi per ottimizzare la gestione del processore e della batteria, evitando sprechi di risorse per le activity non utilizzate dall'utente.
Un'activity viene a conoscenza del proprio stato attuale attraverso l'esecuzione, da parte di Android, di alcuni metodi della classe Activity di cui è possibile fare l'ovverride.

I metodi principali sono:

- onCreate: Metodo chiamato all'apertura dell'applicazione, in cui l'applicazione deve caricare l'interfaccia utente dal file XML;
- onStart: Viene eseguito subito dopo onCreate nel caso dell'avvio dell'applicazione, oppure viene chiamato dopo che l'applicazione è stata nello stato onStop;
- onResume: Viene eseguito dopo onStart, e nel caso in cui l'activity era precedentemente nello stato onPause;
- onPause: Viene eseguito quando l'activity viene nascosta da un'altra activity, come un popup o una notifica;
- onStop: Viene eseguito quando l'applicazione non è più visibile sullo schermo, come l'azione di ritorno alla schermata home oppure alla ricezione di una chiamata;
- onDestroy: Metodo chiamato quando Android ha deciso di chiudere l'applicazione. In questo metodo devono essere rilasciate tutte le risorse utilizzate;

Fragment

I Fragment sono dei componenti grafici che possono essere riutilizzati e combinati in una Activity. Ogni Fragment ha un life-cycle, direttamente influenzato da quello dell'Activity che contiene il Fragment. Il contenuto del Fragment può essere modificato in run-time.

Preferences

Le SharedPreferences sono delle mappe chiave-valore che è possibile utilizzare per salvare la configurazione delle impostazioni e le preferenze dell'utente. Ogni applicazione ne riceve una, e ogni componente può vedere le stesse Preferences all'interno della stessa applicazione.

Services

I Services possono essere paragonati a delle Activity senza interfaccia utente. Vengono eseguiti quindi in background e possono essere utilizzati per creare notifiche, eseguire operazioni lunghe e esose di risorse (upload di file, elaborazione dati) o rimanere in ascolto di eventi di cui non sono disponibili gli Intent.

Esistono due tipi di Service:

- Started Service: È il service più semplice, ed è possibile solamente avviarlo. Quando il Service ha completato il suo lavoro, si ferma;
- Bound Service: Questo permette, una volta avviato, di essere controllato, attraverso un oggetto che permette di fare comunicare chi ha avviato il Service (per esempio una Activity) con il Service stesso;

Ogni Service deve essere dichiarato nel file Manifest.

Implementazione

Per implementare l'applicazione è stato importato, attraverso Gradle e JitPack, il progetto GoBoxJavaApi, e quindi non c'è stato bisogno di ridefinire le stesse classi per potersi collegare al proprio storage.
L'applicazione è formata da un'unica Activity contenente un PreferenceFragment, un particolare Fragment che viene sempre definito tramite un file XML, ma anziché definire il layout dei vari componenti, permette di definire semplicemente le varie impostazioni. Android automaticamente interpreterà i tipi di impostazioni mostrando il miglior componente per la modifica di quella preferenza.
I componenti che Android include non sono però sufficienti per la selezione della cartella di destinazione. Per questa impostazione è stato creato un dialog apposito, che estende la classe FragmentDialog. All'interno di questo dialog viene mostrata una *RecyclerView* che mostra le cartelle presenti sul proprio storage. Dalla lista è possibile scegliere o creare una nuova cartella.
Infine è stato creato il service che viene eseguito in background. Il servizio ha una logica molto semplice: all'avvio del servizio, ovvero quando il metodo onCreate viene chiamato, viene creato e registrato un nuovo ContentListener. In questo modo quando una nuova foto verrà eseguita, verrà chiamato uno specifico metodo di questo oggetto.
Sfortunatamente il listener dell'evento non riporta il nome del file della fotografia acquisita, quindi è stato necessario consultare l'oggetto che gestisce la galleria di Android. Questo oggetto, il MediaStorage permette di essere consultato attraverso un cursore, lo stesso che si utilizza nei database di Android. Una volta trovato il file, all'interno di un AsyncTask, viene creata un'istanza di StandardGBClient che verrà poi utilizzato per fare l'upload.

Sviluppi Futuri

Protocollo WebRTC

Il protocollo WebRTC (Web Real-Time Communication) è un nuovo protocollo realizzato e mantenuto da Google, Mozilla e Opera per semplificare la realizzazione di applicazioni che necessitano una comunicazione Real-Time tra più client. Infatti, necessita di un server solamente per scambiare delle informazioni tra i client (in una fase detta *signaling*), che verranno poi utilizzate per creare una connessione *P2P* (peer-to-peer). Rispetto alla struttura di comunicazione attualmente implementata per la realizzazione della piattaforma, questo protocollo permette di ridurre il carico di lavoro dei server, permettendo di stabilire il collegamento Direct anche quando lo storage si trova sotto una rete NAT.

Logo WebRTC

Firebase

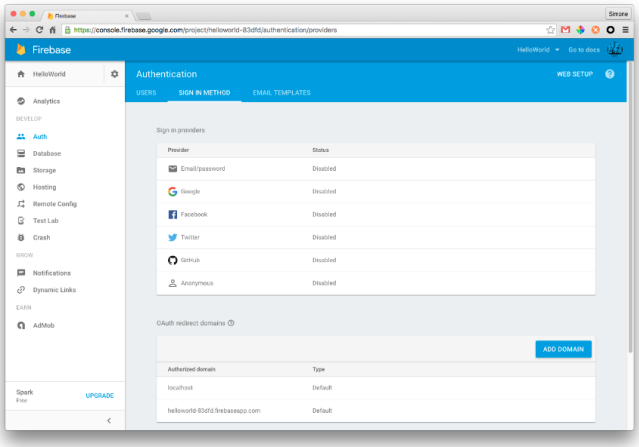
Logo Firebase

Firebase è un servizio che fornisce diverse funzioni, tra cui un sistema di autenticazione e autorizzazione, un database di tipo non relazionale, uno spazio di archiviazione, gestione delle notifiche push e hosting della propria applicazione. È stato recentemente acquisito da Google e il servizio viene fornito proprio attraverso i loro server.

Con la sostituzione del protocollo di comunicazione attualmente utilizzato, in favore di WebRTC, non è più necessario il server principale, dato che il suo restanti compiti riguarderebbero unicamente l'autenticazione e il *signaling*. Questi compiti possono essere gestiti direttamente da Firebase.

Autenticazione

Con Firebase è possibile gestire i propri utenti fornendo un'autenticazione composta da email e password, oppure utilizzando servizi comuni, come account Google, Facebook, Twitter e Github. L'autenticazione attraverso questi servizi è di tipo OAuth2. Permettendo all'utente la possibilità di autenticarsi con più servizi è possibile fornire una migliore esperienza d'uso.

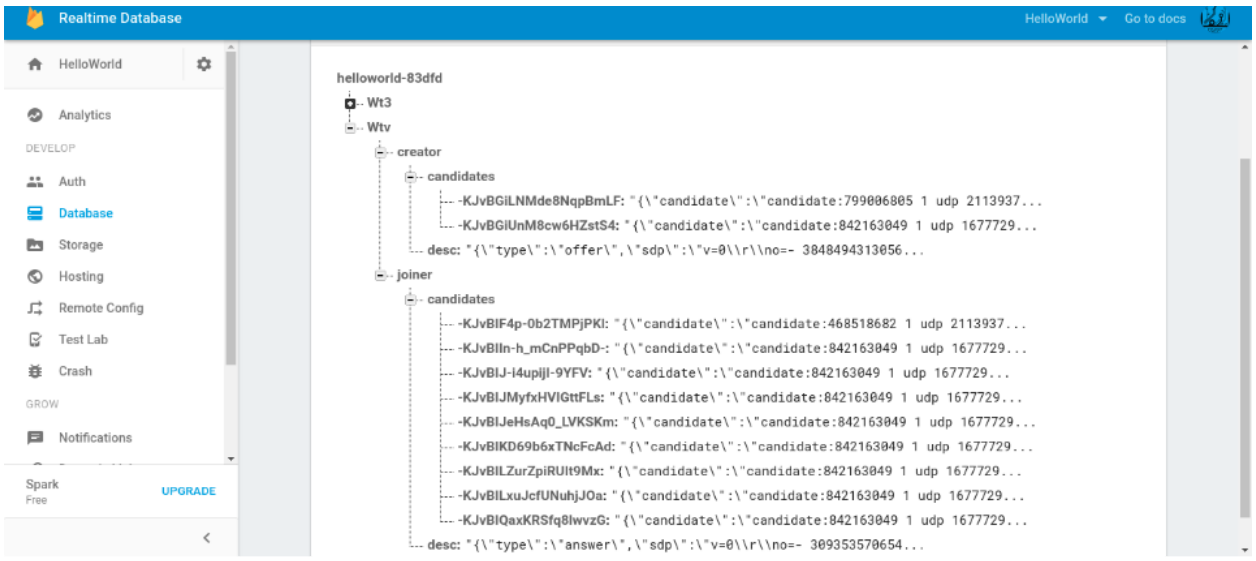


Amministrazione dei metodi di autenticazione di Firebase

Inoltre utilizzando questo sistema di autenticazione, le procedure per il recupero della password o ripristino dell'account sono già implementate, e necessitano solamente di un'interfaccia grafica da implementare.

Database per il signaling

Il database fornito da Firebase vanta la caratteristica di essere *real-time*, ovvero dedicato all'aggiornamento dei dati e alla sincronizzazione immediata delle modifiche. In questo modo è possibile utilizzarlo per poter salvare temporaneamente i dati necessari per completare la procedura di handshake richiesta da WebRTC.
Essendo un database non relazionale, tutti i dati vengono gestiti come oggetti ed è possibile accedere ad essi utilizzando le diverse API per le varie piattaforme (Android, iOs o JavaScript per le WebApp) o utilizzando il web service RESTful. La visualizzazione dei dati attraverso la console e l'utilizzo del web service utilizzano la notazione JSON.



Visualizzazione dei dati contenuti nel database. In questo caso relativi ad un handshake WebRTC

Conclusioni

Realizzando GoBox ho potuto acquisire nuove conoscenze e sperimentare nuove tecnologie, approfondendo inoltre quelle studiate durante il periodo scolastico. La piattaforma non è ancora pronta per poter essere rilasciata al pubblico di internet, ma implementando le idee contenute negli "sviluppi futuri" raggiungerà presto uno stato utilizzabile.

Sitografia

- Android Developers, Getting Started
  https://developer.android.com/index.html
- AngularJS, Getting Started
  https://angularjs.org/
- Angular Material, Elements Guide
  https://material.angularjs.org/latest/
- GoDoc
  https://godoc.org/
- GoLang, Getting Started
  https://golang.org/
- Oracle Java Documentation
  https://docs.oracle.com/javase/tutorial/
- Progressive Web App
  https://developers.google.com/web/progressive-web-apps/
- The Go Blog
  https://blog.golang.org/
- WebRTC
  https://webrtc.org/
- WebSocket
  https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API