Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Progettazione di un applicativo web-based per il backup dei dati di Google Drive

Relatore:
Chiar.mo Prof.
Danilo Montesi

Presentata da:
Andrea Tumino

Parole chiave: cloud storage backup, web application, Google Drive

IV Sessione
Anno Accademico 2017/2018

Prefazione

Al giorno d'oggi è sempre più diffuso l'utilizzo di sistemi di cloud storage per avere a disposizione in qualsiasi momento e su qualsiasi dispositivo file e documenti per essere visualizzati, modificati ed eseguiti.
Questi sistemi, per garantire sicurezza e affidabilità agli utenti, presentano meccanismi di download dei file nel proprio dispositivo affinché si possa accedere ad essi localmente, ossia senza passare dalla rete, e di backup, ossia di salvataggio del contenuto dell'archivio affinché possa essere completamente scaricato dall'utente o per recuperare file accidentalmente persi o modificati, ripristinando il loro contenuto al backup selezionato.
Ogni sistema di clouding ha un modo diverso di gestire i file dei propri utenti, offrendo servizi come ad esempio la condivisione di essi con altre persone, assegnando loro delle autorizzazioni che ne limitano l'accesso (servizio offerto da Google Drive e DropBox), oppure meccanismi per avere la massima compatibilità e versatilità con software specifici (esempio OneDrive e Microsoft).
Affascinato da questi meccanismi ho deciso anche io di ideare, progettare e implementare un sistema di backup, con servizi che guidano l'utente ad effettuare un salvataggio semplice ed efficace dei propri file.
A tal proposito, scrivo questo documento allo scopo di presentare l'architettura e il funzionamento del progetto che ho sviluppato, chiamato "GoogleDrivePersonalBackup", un software che offre un servizio di backup locale personalizzato, con il quale l'utente può scegliere su quali file fare il backup e archiviarli pacchetti zip che può sia scaricare direttamente che mandarli alla propria mail come allegato; inoltre è possibile far generare al sistema un

insieme di pacchetti per archiviare tutto il contenuto del proprio google drive e non solo.
Il progetto, che si presenta come una web app, permette anche il download di file di Google (ad esempio Google Docs) convertendoli al formato più adeguato.
La tesi è formata da 5 capitoli:

- nel primo Capitolo spiegherò cosa si intende per servizio cloud storage e si confronteranno i servizi più conosciuti, evidenziandone analogie e differenze.

- nel secondo Capitolo verrà descritta la progettazione l'applicativo web "GoogleDrivePersonalBackup", partendo dall'analisi dei requisiti e arrivando ad una descrizione del servizio offerto dall?applicazione.

- nel terzo Capitolo verrà descritta come l'applicazione è stata sviluppata e le tecnologie utilizzate, ponendo particolare attenzione al linguaggio Google Script, dai servizi che offre ai limiti che impone a come tali limiti sono stati superati.

- nel quarto Capitolo verrà descritta l'architettura dell'applicazione, spiegando nel dettaglio tutte le sue componenti e come sono state implementate.

- nel quinto Capitolo si traggono le considerazioni finali e le conclusioni.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

L'archiviazione cloud (o in inglese cloud storage) è uno spazio di archiviazione che viene messo a disposizione degli utenti su una rete, che può essere sia pubblica (internet) che privata (rete domestica o aziendale).
L'archiviazione, nella maggior parte dei casi, viene effettuata in server specifici che si occupano di gestire tutte le risorse al loro interno.
Il servizio di cloud-storage può essere offerto sia gratuitamente che tramite la sottoscrizione di un abbonamento mensile o annuale.
La strategia più usata dalle aziende che offrono servizi di cloud storage è quella di concedere all'utente uno spazio di archiviazione assieme a dei servizi base completamente gratuito, con la possibilità sia di aumentare tale spazio che di aggiungere un insieme di servizi aggiuntivi sottoscrivendo un abbonamento.
Questo modello di servizio, in cui il volume e i costi delle risorse di storage acquistate possono essere regolati in qualsiasi momento, è anche indicato come public cloud storage (cioè come cloud storage pubblico).
Un'alternativa è il cosiddetto private cloud storage (cloud storage privato), che si trova sui server di una rete privata, in genere in una rete aziendale, ed è accessibile solo agli utenti all'interno di quella stessa rete. In questo caso, l'operatore (l'azienda stessa o una terza parte) ha il pieno controllo dello stoccaggio dei dati, ma è anche responsabile dell'hardware e della sua

amministrazione.
Esiste anche una soluzione ibrida, l'hybrid cloud storage, in cui la soluzione cloud privata e interna è collegata a un sistema di archiviazione online esterno. Ciò è particolarmente interessante per le aziende che da un lato necessitano di spazio di memorizzazione interno per i dati sensibili e dall'altro di capacità accessibili al pubblico e facilmente scalabili.
Uno dei vantaggi più importanti dei servizi cloud-storage è che consente di risparmiare sull'acquisto e sulla gestione di hardware di proprietà; Infatti, a meno che non scegliate una soluzione cloud interna, la responsabilità di tutta l'infrastruttura ricade sul provider scelto.
Un altro vantaggio riguarda il backup; infatti l'originale e la copia di backup non sono nello stesso posto, in questo modo si evita di correre il rischio di una perdita completa dei dati in caso di attacchi informatici o guasti dell'hardware. Altri vantaggi del cloud-storage sono:

- flessibilità: è possibile sia usare i servizi cloud-storage che interrompere l'utilizzo in qualsiasi momento. Al contrario, decidendo di fare affidamento sul proprio hardware, di solito non sarà così facile sbarazzarsene una volta che non si ha più bisogno dello stesso spazio di archiviazione.

- scalabilità: la virtualizzazione dell'ambiente di archiviazione consente di scegliere il volume adatto alle vostre esigenze, regolabile in qualsiasi momento.

- disponibilità: l'archiviazione cloud pubblica è disponibile in qualsiasi momento e da qualsiasi dispositivo, purché si disponga di una connessione a internet funzionante.

I meccanismi di cloud storage presentano anche degli svantaggi. In primo luogo va menzionata la dipendenza dalla connessione a Internet o dalla connessione intranet al server di cloud privato interno. Per questo motivo la stabilità della rete gioca un ruolo importante; infatti nel momento in cui la connessione smette di funzionare, anche l'accesso al cloud storage viene meno, Inoltre la larghezza di banda disponibile fa il buono e il cattivo tempo

per la trasmissione regolare dei dati, diventando particolarmente problematica con l'accesso alla rete da dispositivi mobili. Altri svantaggi dei sistemi cloud storage sono:

- dipendenza dal provider: quando si sceglie una soluzione di archiviazione cloud esterna, si diventa automaticamente dipendenti dal provider e non si possono escludere modifiche indesiderate dell'offerta o l'interruzione del servizio;

- sicurezza: l'invio di dati oltre i confini della rete interna comporta rischi per la sicurezza, in quanto non tutti i provider offrono la crittografia dei dati archiviati. Inoltre le infrastrutture utilizzate dai provider sono un interessante bersaglio per gli attacchi informatici;

- protezione dei dati: aspetto non trascurabile riguarda la protezione e tutela dei dati; infatti le varie infrastrutture di archiviazione dei sistemi cloud storage possono trovarsi in posizioni geografiche diverse, con direttive legali diverse che possono andare in conflitto tra di loro, come ad esempio il Regolamento Generale sulla Protezione dei Dati (GDPR) in vigore in Europa dal 25 Maggio 2018 costringe i sistemi cloud storage a trattare in maniera diversa i dati che si trovano in territorio europeo[1].

1.1 Tecnologie utilizzate dai servizi di cloud storage

I servizi cloud storage hanno l'esigenza di gestire miliardi di file, a tal proposito i provider si affidano a un'architettura che in termini di prestazione supera l'archiviazione gerarchica basata su file e directory. Queste architetture si basano su risorse astratte dall'hardware fisico, ovvero virtuali, i quali permettono di avere le prestazioni più elevate poiché rendono possibile l'utilizzo combinato di memorie HDD e SSD.

Le tecnologie di virtualizzazione possono utilizzare server pubblici, ossia accessibili da qualsiasi utente tramite registrazione, sia server privati; in questo caso si parla di sistemi On-premises, ossia architetture software progettate per essere usate in un ambito privato, che può essere sia di un singolo utente che aziendale.
Utilizzando le tecnologie di virtualizzazione è possibile astrarre dai server lo spazio di storage rendendo possibile raggruppare tutto lo spazio virtuale acquisito nel Data Lake, al quale gli utenti accederanno come a un repository unico.
I Data Lake sono spazi di archiviazione che contengono una grande quantità di dati grezzi nel suo formato nativo, da conservare fino a quando non viene elaborato. A differenza delle strutture di dati gerarchiche come cartelle, file, righe e colonne, un lago dati è una struttura di file piatta che conserva la struttura originale dei dati così come è stata inserita.
Ad ogni elemento di un Data Lake viene assegnato un identificatore univoco e un insieme di tag di metadati estesi; in questo modo, quando viene eseguita una query basata su determinati metadati, tutti i dati che presentano tali metadati vengono quindi analizzati.
Esistono due alternative per l'archiviazione dei dati:

- Block Storage;
- Object Storage;

L'archiviazione Block Storage, come suggerisce il nome, struttura i dati in blocchi di dati di uguali dimensioni, ognuno dotato di un proprio indirizzo.
Gli indirizzi delle celle fisiche di memoria vengono astratti, quindi la posizione geografica effettiva delle unità di memorizzazione è irrilevante per la memorizzazione dei file.
La tecnologia Block Storage è stata utilizzata per lo sviluppo di sistemi come Amazon Elastic Compute (EC2) [2], OpenNebula[3] e OpenStack [4]; Il disco rigido o il server su cui si trovano i nuovi dati, come devono essere disposti i singoli blocchi e come funziona l'accesso ad essi possono essere facilmente

regolati tramite software.
Nel cloud, l'archiviazione a blocchi è particolarmente adatta come formato di archiviazione per programmi di database e altre applicazioni che lavorano con dati strutturati.
L'Object Storage memorizza i file come oggetti.
Ogni oggetto riceve un numero di identificazione univoco che può essere utilizzato per la sua gestione accedendo alle applicazioni apposite senza che sia necessario disporre di un controllo di accesso.
L'Object Storage non rende possibile la modifica degli oggetti salvati, qualsiasi modifica assicura la creazione di un nuovo oggetto che contiene la versione personalizzata del file originale; ciò rende lo storage degli oggetti il formato di cloud storage ottimale per le soluzioni di backup e archiviazione.
Inoltre la gestione basata su oggetti è adatta per archiviare file multimediali di sola lettura, come musica, video o film (ad esempio quando si utilizzano servizi di streaming).

1.2 Analogie e differenze tra i sistemi cloud storage più conosciuti

Attualmente la tecnologia cloud-storage è più che consolidata e il numero di utilizzatori è ancora in crescita; infatti è stato stimato che nel 2020 il numero di utenti che usufruiscono di questi servizi raddoppierà rispetto al 2014.
La crescita di questo fenomeno attirò l'attenzione di molte aziende le quali decisero di entrare nel settore e sviluppare il proprio servizio di cloud storage.
Tra le aziende più conosciute nel mondo del cloud-storage abbiamo sicuramente Google, Microsoft e Apple, ma anche altre che sono riuscite ad acquisire fama e prestigio proprio grazie al loro servizio di archiviazione, come ad esempio DropBox e Box.
La nascita di molti servizi di cloud storage portò le aziende sviluppatrici a rendere il proprio servizio sempre più unico in modo da differenziarsi dai

concorrenti e acquisire maggiore vantaggio.
La specializzazione di un servizio cloud storage viene effettuata indirizzandosi verso un'utenza e un uso specifico, che può essere personale o aziendale; tuttavia si possono individuare delle caratteristiche comuni che sono diventate basilari per ogni sistema di cloud storage.
Queste caratteristiche sono:

- accessibilità: possibilità di accedere al proprio archivio da qualsiasi dispositivo;
- usabilità: il servizio deve essere semplice e intuitivo da utilizzare per l'utente;
- sicurezza: i provider devono garantire ai propri clienti massima protezione sui loro dati;
- backup: possibilità di effettuare una copia di riserva del proprio archivio.
- ripristino di file: possibilità di recuperare file persi.

Come detto precedentemente, i provider aggiungono ai loro cloud-storage dei servizi aggiuntivi che rendono il loro prodotto unico e diverso dagli altri.
Tra i servizi aggiuntivi più comuni si hanno:

- condivisione di file e cartelle con altri utenti, come ad esempio Google Drive[5] e Dropbox[6];
- sincronizzazione dei file e delle cartelle, come ad esempio One Drive[7] e Google Drive Stream.

Differenza sostanziale tra i vari servizi cloud storage sta nel pagamento e negli abbonamenti; infatti ogni provider permette all'utente di registrarsi con un determinato account che nella maggior parte dei casi può essere free o premium.

In base al tipo di account selezionato varia lo spazio di archiviazione e i servizi allegati.
La pratica più utilizzata è quella di far registrare l'utente gratuitamente offrendo un periodo limitato di prova dell'account premium, una volta terminato l'account viene portato alla versione free.
Alcuni provider offrono soluzioni di cloud storage differenti in base al bisogno dell'utente come ad esempio Google che offre sia il servizio Google Drive, maggiormente pensato per uso personale o aziendale, che Google Cloud platform, pensato per gli sviluppatori.
Altro servizio importante è il fatto di poter modificare direttamente file e documenti dall'archivio come permette Google Drive integrando servizi come Google Doc o Google Sheets oppure quello di fornire assieme all'abbonamento software che interagiscono direttamente con il proprio storage inserendo automaticamente i nuovi file creati o mantenendo aggiornato l'archivio, tale meccanismo è offerto da OneDrive che comunica con il pacchetto Microsoft Office dell'utente oppure DropBox con DropBox Paper[12].
Nella tabella sono rappresentati alcuni dei servizi cloud storage più conosciuti, evidenziandone le differenze per quanto riguarda le dimensione di storage, prezzi e servizi aggiuntivi:

	Google Drive	**DropBox**	**Box**	**One Drive**	**ICloud Drive**
Dim.e massima di un file	5TB	10GB	250MB gratuito, 5GB a pagamento	10BG	nessuna
Dim. archivio gratis	15GB	2GB	10GB	5GB	5GB
piani tariffari	personal /Enterprise /G Suite	personal /businnes	personal/ businnes	personal/ businnes	personal
prezzi	2,99€/mese 100GB 10,99€/mese 1TB	10,99€/mese 1TB 21,99€/mese 2TB	9€/mese 100GB	2€/mese 50GB 69€/anno 1TB	0,99€/mese 50GB 2,99€/mese 200GB 9,99€/mese 1TB 19,99€/mese 2TB
sistemi operativi supportati	Win Android Ios MacOs	Win, Mac, Linux, Android, iOS, Win Phone	Win, Mac, Android, iOS, Windows Phone, BlackBerry	Win, Mac, Android, iOS, Windows Phone	Mac/iOS
file e cartelle condivise	si	si	per piano businnes	a pagamento	per utenti apple
sinc. cartelle	no	si	no	a pagamento	no

Tabella 1.1: Caratteristiche dei servizi cloud storage più conosciuti, dati prelevati il 22 febbraio 2019, from [5], [6], [7], [9], [10]

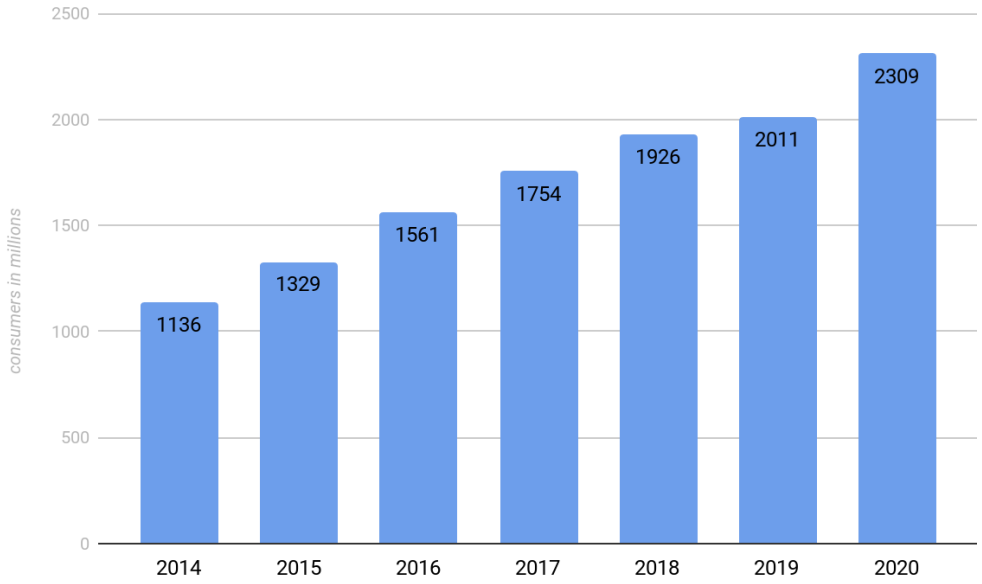

Figura 1.1: Numero di utenti che utilizzano un servizio cloud storage dal 2014 fino al 2020, from [8]

Capitolo 2

Progettazione dell'applicazione di backup

L'obiettivo dell'applicazione è quello di scaricare un numero arbitrario di file presenti nel Google Drive dell'utente e di ottenere un elenco dei suoi contatti presenti in Google Contacts nel proprio dispositivo in modo tale che si possano accedere e modificare localmente, ossia senza l'obbligo di essere connessi ad internet.
La scelta di effettuare un sistema di backup personalizzato invece che "tradizionale" è dovuta dal fatto che effettuare il salvataggio in locale di grandi quantità di dati richiede molto tempo e spesso ad una persona interessa maggiormente scaricare solamente certi file invece che altri per vari motivi, che possono essere per necessità, mancanza di tempo o gusti personali.
Il progetto "GoogleDrivePersonalBackup" offre proprio questo, ossia permette la creazione un insieme arbitrariamente grande di file e inserirli in un unico pacchetto da scaricare.
In questo modo gli utenti possono gestire il backup del proprio archivio in base alle proprie esigenze, dando priorità ai file che ritengono più importanti.
L'attore principale del software è l'utente, che può svolgere le seguenti attività:

- Selezionare un numero arbitrario di file del proprio Google Drive e

richiedere che venga effettuato il download di questi ultimi;

- Una volta terminato il download può salvare i file nel proprio dispositivo;
- Ricevere il pacchetto tramite mail come allegato;
- Richiedere l'elenco di tutti i suoi contatti presenti in Google Contacts;
- Controllare tutti i download effettuati nella sessione corrente, verificando lo stato del pacchetto, i file contenuti all?interno ed eventuali che non è stato possibile scaricare;
- Personalizzare un pacchetto assegnandogli un nome.

Poiché l'applicazione utilizza dati provenienti da servizi Google, il suo utilizzo è esclusivo agli utenti che dispongono di un account Google, non è possibile interagire con altri sistemi cloud-computing.
Per accedere all'applicazione l'utente prima deve collegarla al suo account Google, consentendo in questo modo al sistema di prelevare i dati richiesti.
Approfondiamo ora i casi d'uso elencati precedentemente:

- L'utente vede le informazioni principali dei file nel suo Google drive: è necessario che l'utilizzatore dell'app sia in grado di riconoscere i file del suo Google Drive e che sia in grado di scegliere quali inserire nel pacchetto e quali invece escludere.
  La visualizzazione dovrà fornire in modo schematico e sintetico tutte le informazioni necessarie per individuare ogni file in modo univoco;
- L'utente dispone di un meccanismo di selezione per decidere quali file includere nel pacchetto da scaricare: poiché la creazione dei pacchetti è la funzione principale, bisogna fare in modo che l'utente riesca a selezionare i file da inserire.
  Un pacchetto contiene un numero di file che dipende dalle intenzioni dell'utente, quindi bisogna fare in modo che la selezione sia rapida e intuitiva;

- L'utente può controllare i file che sta inserendo nel pacchetto: man mano che viene creato il pacchetto, bisogna informare l'utente del contenuto che sta inserendo senza ostacolare l?esperienza dell'app.
  Il controllo del contenuto deve essere immediato senza far interrompere la procedura di selezione dei file all'utente;

- L'utente può annullare la selezione di un file ed eliminarlo dalla lista di quelli selezionati: bisogna considerare anche il caso in cui l'utente abbia dei ripensamenti su un file inserito e lo vuole togliere e l'applicazione deve fornire un metodo del tutto analogo alla selezione di un file per rimuoverlo dal pacchetto;

- L'utente può cercare i file tramite una ricerca da tastiera: la versione base di Google drive permette di archiviare fino a 15GB, quelle a pagamento arrivano fino a 1TB, questo significa che il numero di file presenti può essere molto grande, toccando anche l?ordine delle migliaia, per questo è necessario facilitare l?utente nella ricerca dei propri file, fornendo una barra di ricerca che permette di filtrare i risultati in base alla parola cercata;

- L'utente ha un modo per interrompere la selezione ed effettuare un download: l'interfaccia utente deve fornire un meccanismo evidente per avviare il download del pacchetto creato;

- L'utente può controllare in qualsiasi momento lo stato del download di un pacchetto: bisogna fare in modo che l'utente sia informato dello stato del download dei propri pacchetti per sapere quando è terminato o se ci sono stati errori;

- L'utente può chiedere al sistema di creare automaticamente dei pacchetti: oltre che il caso di backup personalizzato si vuole anche includere un servizio di backup completo, dando però all'utente il controllo sui download.

A tale scopo si implementa un meccanismo che divide tutto il contenuto del drive in pacchetti di uguale dimensione, dando all'utente la scelta di quali scaricare per prima;

- L'utente può accedere ad uno storico di tutti i download effettuati: bisogna sviluppare una schermata dal quale chi utilizza l'applicazione può vedere tutti i download che ha effettuato durante la sessione in corso.
  L'obiettivo dello storico dei pacchetti non è di semplice consultazione, ma permette all'utente di effettuare nuovamente il download di pacchetti scaricati precedentemente;

- L'utente può ritentare eventuali download falliti o eventualmente eliminarli: può succedere per motivi legati alla rete che un download fallisca, per questo bisogna permette all'utente di gestire tali casi, con procedure che gli permettono di ritentare il download o di eliminare il pacchetto;

- L'utente viene avvisato quando un download viene ultimato;

- L'utente ha la possibilità di creare pacchetti di file e avviarne il download anche mentre è attivo il download di un pacchetto: quando il download di un pacchetto è iniziato, esso non deve interrompere la navigazione, al contrario, il download deve essere del tutto trasparente all'utilizzatore, il quale può continuare il suo backup creando altri pacchetti.

2.1 Analisi dei requisiti

Dopo aver chiarito i casi d'uso, bisogna fare un'analisi dei requisiti, ossia le funzionalità che il nostro sistema deve offrire.
Partiamo dai requisiti funzionali, analizzando quello che l'applicazione deve fare in precise circostanze.
Le varie circostante in cui l'applicazione si può trovare sono:

- attesa: si attende l'arrivo dei dati da parte del server;
- operativo: l'utente può interagire ed effettuare le azioni volute.
- download: è in corso un download

In base alla circostanza in cui si trova il sistema deve essere in grado di avvisare l'utente di quello che sta succedendo.
Nel caso in cui il sistema sia operativo, quindi non si sta attendendo nessuna risposta dal server, deve:

- gestire la visualizzazione grafica dei dati, garantendo un layout semplice e completo. I dati che il sistema deve gestire sono i file, i download e i contatti. Ogni file è caratterizzato da:
  - un'icona che specifica la sua estensione (es .pdf);
  - il suo nome;
  - la sua dimensione in Mb;
  - il suo proprietario;
  - la data di creazione del file in formato GG/MM/YYYY

  La struttura grafica da utilizzare è quella di una tabella dove su ogni riga è contenuto un file con tutte le caratteristiche descritte e su ogni colonna una precisa proprietà.
  Nel caso in cui la dimensione di un file sia troppo piccola, ossia minore di 10Kb, verrà visualizzato il carattere "-" al posto di 0Mb nella cella rispettiva.
  Ogni download è caratterizzato da:
  - un'icona che specifica lo stato del download;
  - lo stato del download
  - il nome del pacchetto zip;
  - i file del pacchetto;

– bottoni che specificano le attività possibili su quel download.

La struttura grafica di un download è dato da una box dove tutti i dati sono visualizzati al suo interno, di conseguenza il layout sarà formato da tante box allineate, una per ogni pacchetto scaricato o da scaricare. Le operazioni che si possono effettuare su un pacchetto sono visibili come bottoni. A seconda dello stato l'icona sarà di forma e colore diversa.
Ogni contatti ha i seguenti dati:

– indirizzo e-mail;

– nome completo;

– NickName;

– numero di telefono.

La gestione grafica dei contatti è esattamente analoga a quella dei file, viene sempre usati infatti una tabelle dove su ogni riga è presente un contatto e su ogni colonna una proprietà.
Poiché i in Google Contacts non sono presenti solo quei contatti che l?utente ha inserito manualmente, ma anche quelli con cui ci sono state conversazioni su GMail, può capitare che certe proprietà non sono definite per un contatto, in tal caso si mette un carattere "-" nella proprietà mancante.

- Inserire un file nel pacchetto al click dell'utente, il quale viene avvisato dell'inserimento attraverso una colorazione della riga rispettiva del file di grigio.

- Rimuovere un file dal pacchetto al click: se un file è stato inserito in un pacchetto, è possibile rimuoverlo cliccandoci sopra.
  Quando viene rimosso un file, il contenuto del pacchetto viene aggiornato.

- Fornire la descrizione dettagliata di un pacchetto con i file da cui è composto ed eventuali file che non è stato possibile scaricare quando l'utente clicca sull?icona relativa.

- Fornire le operazione possibili su un pacchetto zip in base al suo stato, le opzioni possibili sono:

  - in caso di download in corso non è possibile effettuare alcuna operazione, bisogna attendere l'esito del download;
  - in caso di download in coda è possibile eliminarlo dalla coda;
  - in caso di pronto per il download è possibile avviare la richiesta al server di scaricare il pacchetto;
  - in caso di download effettuato con successo si può scegliere se ricevere il pacchetto via mail al proprio indirizzo oppure scaricarlo direttamente tramite browser;
  - in caso di download non riuscito è possibile sia ritentare il download che eliminare il pacchetto.

- Ottenere i dati relativi all'utente da Google Drive e Google contacts: Al caricamento dell'applicazione il sistema deve fare una richiesta ai database di Google per ottenere tutte quelle informazioni che servono all'utente per effettuare il backup, ossia:

  - archivio di Google Drive: tutti i file che l'utente ha nel suo Google Drive, per ognuno si richiede:
    - Id: valore univoco del file;
    - nome del file;
    - dimensione in byte del file;
    - data di creazione del file;
    - proprietario del file;

  - proprietario del file; archivio di Google Contacts: tutti i contatti che l'utente ha nel suo Google Contact. Per ogni contatto bisogna richiedere:
    * indirizzo e-mail;
    * nome;
    * NiackName;
    * numero di telefono

- Avere un bottone per permettere all'utente di terminare la creazione del pacchetto e avviarne il download.

- Avere un sistema di selezione automatica, ossia un modo che permette all'utente di includere tutti i file del suo Drive nel pacchetto senza selezionarli uno ad uno.

- Creare automaticamente pacchetti su richiesta dell'utente che, cliccando su un bottone apposta, avia tale procedura.
  Questa funzione divide tutto l'archivio del Google Drive in pacchetti di dimensioni fissa pronti per essere scaricati.

- Svere delle "etichette" che permettono all'utente di cambiare schermata: l'applicazione presenta 3 schermate:
  - schermata con la tabella dei file;
  - schermata con la tabella dei contatti;
  - schermata con l'elenco dei pacchetti zip creati

  Il sistema deve garantire all'utente un modo per cambiare schermata rapidamente con il click del mouse, di conseguenza bisogna definire dei bottoni per gestirle.

- Avere un metodo di ricerca come sostegno alla selezione dei dati: l'utente deve avere a disposizione una barra di ricerca nella schermata dove inserendo una stringa di testo, la tabella viene aggiornata con tutti quei

file il cui nome contiene la stringa inserita. In barra di ricerca agisce sui dati presenti nella schermata corrente ignorando le altre.

- tenere memoria dei download effettuati durante la sessione: tutti pacchetti creati durante una sessione devono essere memorizzati e disponibili all'utente nella schermata dei download.

Questi requisiti hanno l'obiettivo di offrire all'utente un'esperienza semplice, chiara e intuitiva di quello che il sistema è capace di fare, facendo in modo che con pochi click riesca a svolgere operazioni di download e nel caso anche un backup dei propri dati presenti nel proprio account Google.
Quando il sistema è in attesa, l'utente deve essere avvisato con delle schermate di caricamento dando una stima dei tempi di attesa e cosa l'applicazione sta attendendo.
In questa circostanza l'utente è libero di navigare nella schermata, tuttavia egli non potrà compiere alcuna operazione poiché i dati necessari non sono ancora pronti.
Quando il sistema sta scaricando un pacchetto, quindi è in stato di download, l'utente ha la possibilità di continuare ad effettuare la creazione di pacchetti nuovi, tuttavia quando ne avvia il download, esso viene messo in coda, una volta terminato il download precedente, il primo pacchetto della coda viene prelevato e messo in download.
Per quanto riguarda i contatti, l'utente può effettuare il download senza attendere il termine del pacchetto.
Oltre i requisiti funzionali bisogna specificare le tecnologie che si intende utilizzare ed eventuali vincoli legati alla sicurezza.
Per quanto riguarda le tecnologie si utilizzano i linguaggi di programmazione e di markup tipici del mondo web, poichè l'applicazione lavora esclusivamente su internet in quanto deve comunicare con Google.
Oltre che le tecnologie "tradizionali", ossia quelle maggiormente utilizzate in ambito web (Html, css, Javascript) si utilizza anche un linguaggio creato da Google chiamato Google Script, che permette l'interazione con i dati presenti nella G-Suite dell'utente che utilizza l'applicazione.

Google Script offre anche una piattaforma e una console per lo sviluppo e debugging direttamente accessibile da browser.
Per quanto riguarda il problema sicurezza, esso è interamente gestito da Google, infatti ogni applicazione creata con Google Script presenta una schermata di approvazione che avverte l'utente dei dati che essa va ad utilizzare e come li utilizza.
L'applicazione si avvia correttamente solo nel caso in cui l'utente consente al sistema l'utilizzo dei suoi dati.
Vedremo nel capitolo dedicato alla descrizione di Google Script come generare la schermata di consenso.
L'analisi dei requisiti è completata, ora bisogna iniziare a progettare l'applicazione individuando tutte le sue componenti e le relazioni tra esse.

2.2 Descrizione del servizio

Abbiamo visto nella sezione precedente tutte le azioni che può effettuare l'utente e cosa deve avere il sistema per consentire lo svolgimento di tale attività.
Ora iniziamo la progettazione del servizio, ossia a definire la struttura interna del software e i moduli da cui è composto, specificando non soltanto come essi sono fatti, ma anche descrivendo ad alto livello le relazioni tra essi e come interagiscono tra di loro.
Sicuramente le 2 componenti necessarie sono un'interfaccia utente per visualizzare i dati richiesti ed elementi grafici, come ad esempio bottoni, per avviare le procedure desiderate e una componente back-end che di occupa di interagire con Google per prelevare i dati ed effettuare il download dei file richiesti.
In base ai requisiti analizzati nella sezione precedente, bisogna progettare l'applicazione decidendo come strutturarla a livello di codice e le tecnologie da utilizzare.
Possiamo distinguere i requisiti elencati in 2 categorie:

- visualizzazione dei dati necessari per garantire all?utente un uso semplice e intuitivo dell'applicazione;

- ottenere i dati necessari dal Google Drive e Google Contacts dall?utente.

Per quanto riguarda la visualizzazione e selezione dei file e delle procedure da chiamare, la soluzione risiede nello sviluppare un'interfaccia utente che organizza graficamente in modo semplice e chiaro tutti gli elementi necessari per un'efficace user experience, invece per quando riguarda i meccanismi di download bisogna implementare un sistema che interagisca con i servizi Google per ottenere i dati necessari.
Il modo migliore per soddisfare tutti i requisiti evidenziati è quello di sviluppare il progetto come una web app, con una componente server che comunica con i servizi di Google e una client che interagisce con l'utente.
L'applicazione si presenta come una single-page application e offre le seguenti servizi:

- creare pacchetti zip con i file del proprio Google Drive: è possibile inserire un file nel pacchetto cliccando sull'elemento associato dalla lista caricata nella pagina Html, per togliere un elemento dal pacchetto basta cliccare nuovamente sull?elemento della lista.
  Una volta deciso il pacchetto da scaricare basta cliccare sul bottone "download" a sinistra della schermata.
  Utilizzando la barra di ricerca in alto a sinistra è possibile ricercare un file per nome;

- effettuare il download dei propri contatti presenti su google Contacts e Gmail;

- Visualizzare i downloads effettuati nella sessione corrente, per ogni download è possibile:

  - visualizzare lo stato del download e il contenuto del pacchetto cliccando sull'icona che indica lo stato del download;

  - scaricare direttamente il pacchetto;
  - ricevere il pacchetto via mail al proprio indirizzo GMAIL.

- Far creare automaticamente al sistema un insieme di pacchetti zip per effettuare il backup completo del proprio Google Drive, una volta creati i pacchetti si può decidere quali scaricare e in che ordine.

- Visualizzare sia contatti salvati sul proprio Google Contacts sia quelli con cui si ha avuto una conversazione con il servizio di posta GMail, è possibile utilizzare la barra di ricerca per filtrare i risultati in base al loro indirizzo e-mail.

- Scaricare in formato .pdf l'elenco dei contatti.

Al caricamento della pagina Html l'app fa una richiesta al server per ottenere i dati principali dei file sul Drive, una volta terminato il caricamento apparirà la lista dei file e da quel momento è possibile iniziare ad interagire con l'applicazione.
La parte client-side si occupa di gestire il layout di tutti i dati e di comunicare al server eventuali download da effettuare dagli archivi di Google Drive.
Il client si occupa anche della creazione dei pacchetti e della ricerca di file su barra di ricerca.

2.2.1 Client-side

Le componenti che formano il client sono:

- un template contenente il layout implementato con HTML 5 e Bootstrap versione 4.1.3;

- Foglio di stile utilizzando CSS3 per gestire il layout del template;

- file di script che effettua le chiamate al server e gestiscono il DOM dell'app, l'implementazione verrà effettuata in javascript con il framework J-query versione 3.3.1.

L'interfaccia utente presenta una schermata con una barra di menu, dal quale è possibile scegliere se visualizzare i propri file, download e contacts.
Il menù presenta anche una barra di ricerca, che filtra i contenuti della schermata principale aiutando l'utente nella ricerca di file, download o contatti specifici.
la schermata principale permette la visualizzazione dei dati indispensabili per effettuare i download desiderati e per controllare lo stato dei download svolti.
L'interfaccia utente presenta anche un menu laterale dove è possibile svolgere le seguenti attività:

- seleziona tutto: seleziona automaticamente tutti gli elementi del Drive;
- download automatico: genera automaticamente i pacchetti zip per il download del Drive completo;
- scarica contatti: effettua il download di tutti i contatti.
- visualizzare il pacchetto che si sta creando con la possibilità di scaricarlo in un qualsiasi momento.

2.2.2 Server-side

La parte server-side è scritta in Google Script, un linguaggio di programmazione offerto da Google, e ha l'obiettivo di soddisfare le richieste del client, prelevando i dati relativi ai file e contatti dell'utente.
La scelta di utilizzare il Google Script e non una tecnologia server side come ad esempio Php o nodeJs è dovuta dal fatto che il linguaggio offerto da Google presenta al suo interno delle procedure built-in che permettono di interagire con poche linee di codice con tutta la Google-Suite relativa all'utente, senza dover effettuare chiamate Rest.
Il server è formato da 3 componenti:

- procedure per il corretto avviamento della web-app, caricando il template e avviando gli script di caricamento;

- componente che contiene tutti i servizi che il client utilizza per l'esecuzione dell'applicazione, ossia gli script che comunicano con i dati presenti nella G-suite dell'utente;
- convertitore per convertire file di Google in un'estensione accessibile su ambienti di sviluppo locali.

Abbiamo definito tutti i moduli della nostra applicazione, sia client-side che server-side, vediamo ora come tutte queste componenti sono relazionate e come interagiscono tra di loro.

2.3 Funzionamento dell'applicazione

Appena l'utente accede all'applicazione tramite browser, viene avviato subito il modulo server che si dedica al caricamento della pagina html.
Caricata la pagina, la componente Javascript del client richiede al server i dati per l'avvio, ossia i file e i contatti, il server prende in carico la chiamata e spedisce al client la risposta.
Oltre al caricamento della pagina, abbiamo l'interazione client-server ogni volta che si effettua il download di un pacchetto, per quanto riguarda invece la gestione del DOM e il download dei contatti il client è in grado di agire autonomamente in quanto presenta tutte le procedure apposite incluse in Javascript.
Le richieste di download vengono mandate al server ogni volta che l'utente clicca sui bottoni appositi.
Quando la richiesta viene effettuata il server elabora la risposta e la manda al client in formato JSON.
Ricevuta la risposta il client converte il JSON e aggiorna i dati sull'interfaccia utente.

Figura 2.1: Schermata di caricamento.

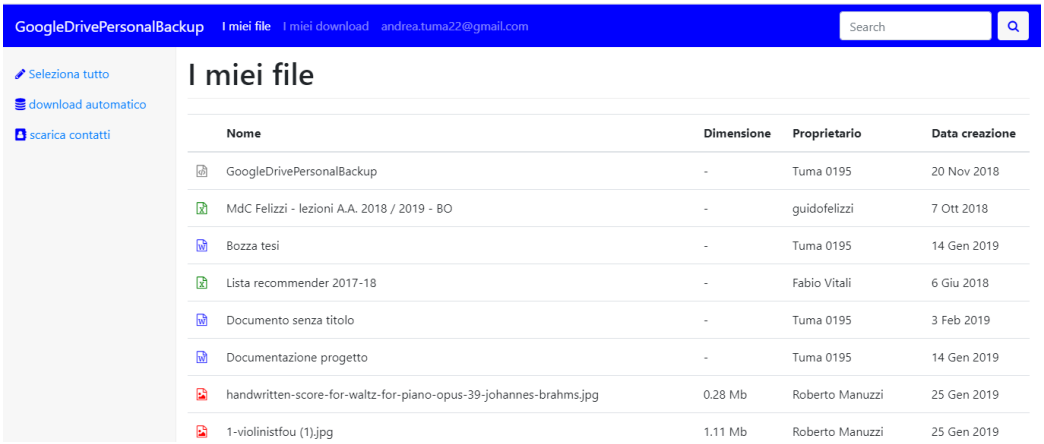

Figura 2.2: Visualizzazione dei file.

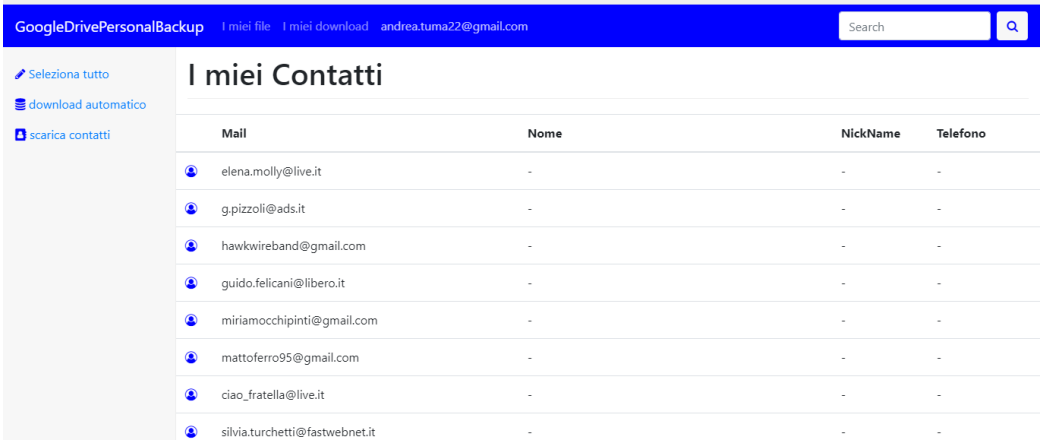

Figura 2.3: Visualizzazione dei contatti.

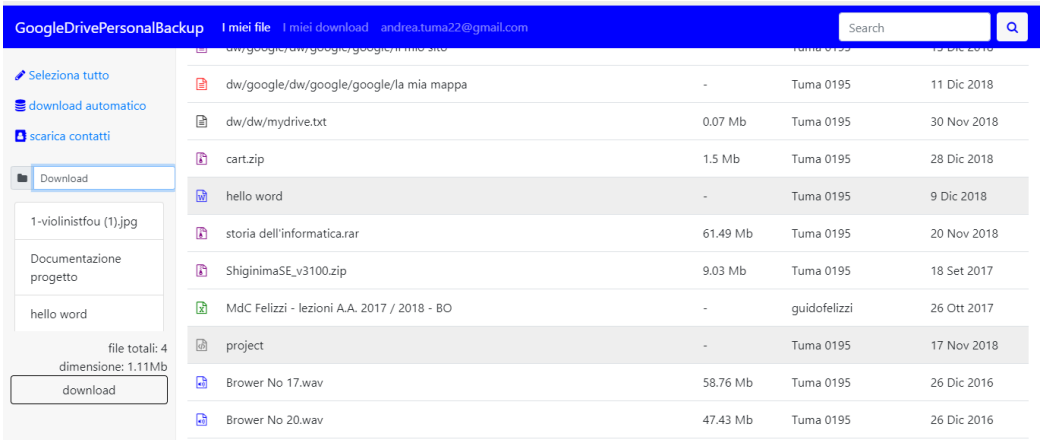

Figura 2.4: Creazione di un pacchetto zip.

Capitolo 3

Implementazione dell'applicazione

Dopo aver fatto chiarezza su come strutturare l'applicazione procediamo verso l'implementazione del progetto.
La piattaforma utilizzata per la stesura del codice è Google Apps Script, servizio gratuito offerto da Google che offre un IDE che permette di creare e implementare i moduli del nostro progetto utilizzando direttamente Google Script senza dover importare alcuna libreria.
Google Apps Script consente anche l'utilizzo dei linguaggi Html, Css e Javascript e permette l'integrazione tramite CDN dei framework desiderati, che nel nostro caso saranno Bootstrap versione 4.1.3, JQuery 3.3.1 e JSPDF 1.5.3.
Oltre che offrire un editor di testo, Google Apps Script offre anche dei servizi per eseguire funzioni specifiche (nel caso si voglia testare una procedura specifica) e per pubblicare l'applicazione creata nei seguenti formati:

- applicazione web: la piattaforma carica il progetto e fornisce un indirizzo url per l'accesso;

- API eseguibile: viene fornito l'id del progetto dal quale è possibile effettuare richieste HTTP;

- applicazione sullo Store di Chrome: lo sviluppatore viene indirizzato ad una pagina dove viene guidato all'inserimento della sua applicazione nello store.
  Per pubblicare il proprio progetto nello store bisogna prima fare una richiesta di verifica dell'applicazione, una volta che Google approva la richiesta e verifica che il codice non presenta procedure "maliziose", come ad esempio la completa eliminazione dei dati dell'utente che accede. Se la verifica viene terminata con successo è possibile pubblicare l'app;

- componente aggiuntivo web: l'applicazione viene pubblicata come un vero e proprio plug-in compatibile con i servizi Google;

- componente aggiuntivo android: permette l'integrazione della propria applicazione con le proprie app android in modo tale che funzioni come componente aggiuntivo;

Tramite Google Apps Script è possibile gestire anche le versioni del proprio progetto, sia creandone di nuove che eliminando quelle superflue.
Le versioni del progetto possono essere usate come librerie per altri progetti.
Google Apps Script offre anche un servizio di debug con una console per visualizzare i Log e uno strumento chiamato Stackdriver per monitorare l'esecuzione del proprio progetto in qualsiasi momento su qualsiasi dispositivo.
Google Stackdriver tuttavia è un servizio aggiuntivo a pagamento e assolutamente facoltativo, infatti è possibile sviluppare e pubblicare applicazioni tramite Apps Script senza utilizzarlo.
Descriviamo ora la tecnologia caratteristica dell'applicazione, il linguaggio Google Script, analizzando le sue caratteristiche, proprietà e i suoi limiti.

3.1 Il linguaggio Google Script

Google Script è un linguaggio di programmazione che consente di implementare script che accedono facilmente ai dati su Google e sui sistemi esterni.

Questi meccanismi sono espressi come oggetti globali con metodi associati, analoghi a oggetti JavaScript.
I servizi che offre Google Script si dividono in 3 categorie:

- I servizi di G-Suite consentono di accedere ai dati dei prodotti G Suite come Drive, Gmail, Calendar, Documenti, Fogli e altri.
  Questi servizi sono separati dalle API create per tali prodotti; in generale, i servizi G Suite hanno un utilizzo più immediato rispetto alle API di G Suite in quanto non bisogna effettuare delle chiamate Rest per richiamare un servizio, ma sono meno performanti; infatti il loro utilizzo è consigliabile solo per i casi d?uso più frequenti, come l'accesso a file, lettura e scrittura di essi.

- I servizi Google avanzati forniscono i mezzi per utilizzare le API di Google all'interno di Apps Script.
  Ciascuno dei servizi Google avanzati è associato a un'API pubblica di Google.
  È possibile accedere a queste API tramite servizi avanzati o semplicemente facendo direttamente le richieste API.
  I servizi di script sono servizi di utilità che non sono collegati a un determinato prodotto G Suite.

- i servizi di utilità permettono di gestire informazioni di registro, creare template HTML, comprimere dati e altro ancora.

3.1.1 Proprietà e caratteristiche di Google Script

Come detto precedentemente, Google Script offre un vasto insieme di servizi per accedere ai dati presenti in tutta la G-suite dell'utente.
La sintassi di Google Script è del tutto uguale a quella di Javascript; infatti è un linguaggio non tipato basato sui prototipi, di conseguenza non esiste il

concetto di classe.
L'unica differenza tra Javascript e Google Script è che quest'ultimo non viene eseguito dal browser, di conseguenza non ha meccanismi di manipolazione del DOM né tantomeno la possibilità di attivare script in corrispondenza di eventi (esempio onClick), ma viene eseguito sugli stessi server e negli stessi data center che eseguono i servizi Google, di conseguenza lo si considera come un vero e proprio linguaggio server-side, come NodeJs[11].
Oltre che l'accesso immediato ai servizi Google, il linguaggio gestire automaticamente i meccanismi Open Authorization o più comunemente OAuth, basta semplicemente inserire in un file dedicato l'url dell'autorizzazione da inserire (disponibile nella documentazione di Google Script).
La comunicazione client-server avviene attraverso chiamate asincrone tramite l'oggetto "google.script" che offre 2 metodi per interagire con il server:

- withFailureHandler(myFailFunction): gestisce il caso di fallimento, se la richiesta fallisce viene eseguita la funzione myFailFunction;

- withSuccessHandler(mySuccessFunction).myRequest(): effettua la richiesta al server, la funzione "myRequest()" è il metodo del server che voglio invocare.
  In caso di successo viene eseguita la funzione "mySuccessFunction()".

Le funzioni che gestiscono i casi di successo o di errore delle richieste sono funzioni implementate ed eseguite dal client, è possibile avere la stessa funzione che gestisce il caso di successo o di errore di multiple richieste.
A differenza di NodeJs, Google Script non ha meccanismi di routing per attivare gli script, ma ha un una procedura per ogni metodo HTTP, quindi per avere un sistema capace di gestire le singole richieste del client bisogna effettuare il parsing dell'url all'interno della funzione e in base ai risultati ottenuti attivare la funzione desiderata.
Le procedure che gestiscono le richieste HTTP sono "doGet(url)" e "doPost(url)" che si attivano rispettivamente ad una richiesta GET e POST da

parte del client.
Analizziamo ora i servizi offerti da Google Script e il loro funzionamento.

I servizi G-suite

I servizi di G Suite sono semplicemente una rappresentazione di tutti i servizi offerti da Google, grazie al quale è possibile interagire con essi sia prelevando i dati che aggiungendone di nuovi o eliminare quelli esistenti.
I servizi di G-Suite sono:

- DriveApp: permette l'accesso al Google Drive dell'utente.
  Utilizzando DriveApp è possibile gestire pienamente ogni file e cartella presente nell'archivio, modificando il suo nome, il suo contenuto e i diritti di accesso su di esso, aggiungendo o rimuovendo sia viewers che editors.
  È possibile anche ottenere qualsiasi informazione relativo ad un file, dal suo nome al suo contenuto a tutti gli account con cui il file è condiviso;
  Il contenuto viene restituito in formato "Blob", il quale contiene dei metodi che permettono il suo manipolamento, questo sarà indispensabile per effettuare le conversioni.

- ContactsApp: permette l'accesso al Google Contacts dell'utente.
  Tramite ContactsApp è possibile ricavare tutte le informazioni desiderate sui contatti dell'utente, ma anche inserire nuovi contatti e modificare quelli esistenti;

- CalendarApp: oggetto che permette di accedere al Google Calendar dell'utente.
  Con CalendarApp è possibile creare, prelevare e modificare calendari e gli appuntamenti su di essi.
  Ogni elemento del calendario è a sua volta un oggetto dal quale si possono ottenere tutte le informazioni relative ad esso;

- DocumentApp: con esso è possibile gestire i documenti presenti nel proprio Google Docs.

Oltre che la modifica dei documenti esistenti, DocumentApp consente anche la creazione di documenti nuovi.
Per modificare il contenuto di un documento DocumentApp offre dei servizi per l'aggiunta di immagini, testo, equazioni e tabelle;

- DataStudioApp: consente la creazione d la modifica dei propri rapporti creati con Google Data Studio;

- FormApp: servizio per la creazione di Form.
  FormApp consente anche di interagire con Google Form ottenendo dati da quelli creati, come ad esempio le domande e le scelte da cui è composto e anche i risultati ottenuti dalla compilazione del Form;

- GmailApp: servizio che permette di integrare i servizi GMail nelle proprie applicazioni.
  Grazie a GmailApp è possibile spedire con l'indirizzo mail dell'utente, impostando il corpo del messaggio sia in formato testo che in formato Html e inserendo un numero arbitrario di allegati.
  Il servizio GMailApp consente anche la ricerca di mail e l'inserimento di mark e segnalini;

- GroupsApp: questo servizio fornisce l'accesso alle informazioni di Google Groups e può essere usato per ottenere informazioni come l'indirizzo email di un gruppo o l'elenco di gruppi in cui l'utente è un membro diretto.
  Tramite GroupsApp è possibile anche ottenere l'elenco dei membri di un gruppo con il loro ruolo;

- LanguageApp: servizio di Google Translate, traduce un testo in formato stringa nella lingua scelta.
  Per effettuare la traduzione è necessario specificare non solo la lingua in cui si vuole tradurre il testo, ma anche la lingua della stringa in input;

- Maps: servizio che consente la creazione di mappe.
  Per ogni mappa creata è possibile inserire segnalini, impostare l'am-

piezza e ottenerle in file Jpg o pdf.
Maps non consente la modifica di mappe create attraverso il tool di Google MyApp;

- SitesApp: permette l'accesso ad ogni progetto implementato con Google Sites, riuscendo ad effettuare modifiche su di esso e ottenere tutte le componenti html di cui il progetto è composto e gestirle a proprio piacimento;

- SlidesApp: servizio analogo a DocumentApp che permette di implementare script che interagiscono con Google Slide creando nuovi file o modificando quelli esistenti;

- SpreadsheetApp: servizio analogo sia a DocumentApp che SlidesApp, gestisce i file di Google Spreadsheet.

I servizi Google avanzati

I servizi Google avanzati forniscono dei meccanismi per accedere alle API pubbliche di Google senza effettuare richieste HTTP.
Questi servizi sono più complicati da usare rispetto a quelli offerti dalla G-Suite, tuttavia offrono delle feature aggiuntive che variano a seconda del servizio invocato.
I servizi avanzati di Google offrono servizi aggiuntivi per lo sviluppo del proprio progetto, come ad esempio servizi di analytics e big query.
Ogni servizio avanzato prima di essere utilizzato deve essere abilitato dall'editor di testo offerto da Google Apps Script.
I servizi G-Suite estesi dai servizi Google avanzati sono:

- Calendar: i servizi avanzati do Google Calendar permettono di personalizzare graficamente il calendario offrendo meccanismi di personalizzazione per ogni evento e la modifica delle impostazioni personali dell'utente;

- Drive: rispetto a DriveApp, i servizi avanzati permettono la creazione di file personalizzati e il loro inserimento nel Drive; infatti il servizio built-in consentiva la creazione soltanto di file Google e l'inserimento di file già esistenti, ossia presenti o nel file-system o in rete.
  Importante è il servizio DriveActivity, che permette la gestione di tutte le attività svolte nel drive non dal proprietario dell'archivio, ma anche quelle di eventuali utenti che hanno file o cartelle condivise;

- GMail: rispetto al servizio built-in, i servizi avanzati offrono l'accesso a informazioni più dettagliate riguardo mail ed etichette.
  I servizi avanzati offrono anche un sistema di notifiche push chiamando un listener al proprio GMail;

- Sheet: oltre che offrire i servizi presenti anche in SpreadsheetApp, con i servizi avanzati si ha un maggiore controllo sui dati del proprio Google Sheet, ottenendo informazioni come il loro tipo, dimensione e opzioni possibili su quel particolare dato;

- Slides: permette la gestione in modo più dettagliato del proprio Google Slides senza aggiungere feature particolari.

I servizi aggiuntivi offerti sono:

- Admin SDK: gestione delle entità in un dominio G-Suite, organizzando gli utenti e i dispositivi connessi al dominio;

- AdSense: gestione del proprio profilo Google AdSense, è possibile anche ottenere un report riguardo le statistiche dell'account AdSense dell'utente su un file di Google Sheet;

- Analytics: gestione del proprio profilo Google Analytics, ottenendo i dati prestazionali del proprio sito web e informazioni sull'account dell'utente.
  Anche Analytics offre un servizio per la creazione di report su Google Sheet;

- BigQuery: gestione del proprio account BigQuery, modificando i progetti, inserendo dati ed eseguendo query;
- Classroom: consente ad amministratori, insegnanti e studenti che utilizzano l'applicazione con tal servizio di gestire corsi presenti nei loro account;
- DoubleClick Campaigns: gestione delle campagne pubblicitarie avviate con DoubleClick Campaign Manager di Google e la creazione dei report per avere un resoconto delle statistiche della propria campagna;
- Mirror: permette all'utente di interagire con Google Glass
- People: consente la creazione, lettura e aggiornamento dei dati dei contatti per l'utilizzatore dell'applicazione e leggere i dati del profilo;
- Shopping Content: permette l'inserimento di acquisti in app attraverso Shopping Content API;
- Tasks: servizio che consente di gestire più approfonditamente i task del proprio account GMail;
- Tag Manager: servizio per gestire i tag di un sito web direttamente dall'applicazione;
- YouTube: integrazione dei servizi YouTube nell'applicazione; tramite questo servizio è possibile inserire un player di YouTube e ottenere informazioni sui video desiderati.

I servizi di script

I servizi di script sono dei meccanismi generici utilizzabili per qualsiasi G-Suite e offrono funzionalità per gestire file html, xml, archivi compressi e servizi di caching e cifratura. I servizi di script offerti sono:

- Base Service: servizi per gestire utenti, file e html dell'applicazione; è presente anche un sistema per gestire i log sia client-side che server-side

con lo StackDriver di Google Apps Script.
Il servizio Base contiene le seguenti classi:

  - Blob: interfaccia per gestire i file;
  - Menu: permette la creazione di menu aggiuntivi per le applicazioni Google;
  - PromptResponse: crea delle schermate di inserimento dati nella UI e gestisce i casi in base alla selezione dell'utente;
  - UI: istanza dell'interfaccia utente per un'applicazione Google e permette di creare script per aggiungere funzionalità come menu, finestre di dialogo e barre laterali;
  - User: informazioni basi dell'utente che sta utilizzando l'applicazione, offre soltanto un meccanismo per ottenere il suo indirizzo E-mail.

- Cache Service: permette l'implementazione di script per gestire le cache delle applicazioni Google.
  Esistono 2 tipi diverse di cache:

  - pubbliche non dipendono da quale utente sta accedendo allo script;
  - private sono per elementi specifici dell'utente, come impostazioni personali o attività recenti.

- Card Service: consente la creazione di card e widget da integrare con l'interfaccia utente.
  Le componenti create con questo servizio funzionano su tutti i dispositivi, sia desktop che mobile senza dover sviluppare diverse versioni;

- Charts Service: permette di implementare script che utilizzano Google Chart Tools per la creazione di immagini interattive, come grafici, tabelle e griglie;

- Content Service: gestione del testo contenuto in form, convertendolo in formato XML o JSON;

- HTML Service: servizio consente gli script di usare testo HTML come valore di ritorno; È possibile con questo servizio ottenere il template dalla stringa Html, da un oggetto in formato Blob o da un file. HTML Service è composto da 3 classi:
  - HtmlOutput: DOM di un documento Html;
  - HtmlOutputMetaTag: rappresentazione dei meta tag di un documento Html;
  - HtmlTemplate: rappresentazione del template Html per la creazione di pagine dinamiche.
- Client-side API: servizi client side per gestire:
  - cronologia ("google.script.history");
  - interazioni con dialogs o sidebar di Google Docs, Sheet oppure Form ("google.script.host");
  - richiesta di esecuzione di script server side (google.script.run);
  - stringa url ("google.script.url").
- JDBC Service: servizio che consente agli script di connettersi a database compatibili con JDBC tra cui Google Cloud SQL, MySQL, Microsoft SQL Server e Oracle;
- Lock Service: servizio utile per la programmazione concorrente, permette la gestione all'accesso a sezione di codice, utile per evitare collisioni quando più utenti accedono ad uno stesso documento o file.
  Lock Service è formato a 2 classi:
  - Lock: rappresentazione di un lock;
  - LockService: gestire gli accessi concorrenti; con questa classe è possibile inserire i lock in documenti, file o porzioni di codice.
- Mail Service: servizio esclusivo per l'invio di mail, non può accedere alla posta ricevuta di un utente.

Tramite Mail Service è possibile monitorare lo stato dei limiti giornalieri imposti da Google Script;

- Optimization Service: servizio per risolvere problemi di programmazione lineare intera;
- Properties Service: servizio che permette di archiviare coppie chiave-valore in formato stringa nell'ambito di uno script o di un documento in cui viene utilizzato un componente aggiuntivo;
- Script: servizio che fornisce l'accesso ai trigger di script e alla loro pubblicazione;
- URL Fetch: servizio che consente agli script di accedere ad altre risorse sul Web attraverso richieste sia HTTP che HTTPS.
  Per utilizzare questo servizio bisogna aggiungere uno scope preciso nel proprio progetto tramite l'editor di testo.
  Le richieste vengono effettuate usando come input l'url della risorsa ed eventuali parametri;
- Utilities Service: servizio che consente di gestire la cifratura, decifratura e formattazione di dati.
  Gli algoritmi di cifratura offerti da Utilities sono Digest, Mac e Rsa.
  Tramite questo servizio è possibile creare dati in formato Blob e archivi compressi in formato zip; sono presenti anche meccanismi per convertire un Blob in una stringa in base64 e viceversa;
- XML Service: servizio per la creazione, navigazione e parsing di documenti XML.

3.1.2 Limiti e restrizioni di Google Script

Nonostante Google Script abbia il pregio di fornire accesso diretto alla G-suite con poche righe di codice, esso impone parecchi limiti e restrizioni agli sviluppatori sia per quanto riguarda la dimensione dei dati che il tempo

di esecuzione.
Nel nostro caso le restrizioni del linguaggio che portano problemi allo sviluppo dell'app sono:

- non è consentito invocare delle procedure su strutture dati di dimensione maggiore di 50 Mb e non può rispondere al client con pacchetti maggiori di 50 Mb;

- l'esecuzione di ogni script deve avere una durata massima di 6 minuti, a meno che non si dispone di un account G Suite Business / Enterprise / EDU, in tal caso il tempo di esecuzione massimo consentito è di 30 minuti;

- La dimensione massima del contenuto totale che si può avere come allegato in GMail è di 25 Mb;

- Ogni script può supportare massimo 10 chiamate concorrenti, in caso di più richieste concorrenti il linguaggio mette in coda le richieste dalla 11esima in poi.

Le restrizioni variano anche a seconda dell'account dello sviluppatore; infatti se si possiede un account Business o Enterprise o Education, il tempo massimo di esecuzione consentito per gli script è di 30 minuti, i limiti di spazio rimangono invariati.
Oltre che i limiti generali ci sono anche i limiti giornalieri, i quali vengono resettati ogni 24 ore.
Tali limiti riguardano il numero massimo di chiamate che si possono effettuare al server e la creazione di contatti, eventi e file Google.
Anche i servizi G-Suite presentano alcune limitazioni; infatti non esistono metodi per effettuare il download di elementi appartenenti a Google Form, Google Maps e Google sites.

Feature	**Consumer**	**G Suite free edition (legacy)**	**G Suite Basic /Gov**	**G Suite Business/ Enterprise/ Education**	**Early Access**
script runtime	6 min/ execution	6 min/ execution	6 min/ execution	30 min/ execution	30 min/ execution
Custom function runtime	30 sec/ execution	30 sec/ execution	30 sec/ execution	30 sec/ execution	30 sec/ execution
Simultan. executions	30	30	30	30	60
Email attachments	250/msg	250/msg	250/msg	250/msg	250/msg
Email body size	200kB/msg	200kB/msg	400kB/msg	400kB/msg	400kB/msg
Email recipients per message	50/msg	50/msg	50/msg	50/msg	50/msg
Email total attachments size	25MB/msg	25MB/msg	25MB/msg	25MB/msg	25MB/msg
Properties value size	9kB/val	9kB/val	9kB/val	9kB/val	9kB/val
Properties total storage	500kB/ property store	500kB/ property store	500kB/ property store	500kB/ property store	500kB/ property store
Triggers	20/user/ script	20/user/ script	20/user/ script	20/user/ script	20/user/ script
Url Fetch response size	50MB/call	50MB/call	50MB/call	50MB/call	50MB/call

Tabella 3.1: Limitazioni di Google Script, from [13]

Feature	**Consumer**	**G Suite free edition (legacy)**	**G Suite Basic /Gov**	**G Suite Bus./ Ent./ Edu.**	**Early Access**
Calendar events created	5000/day	10000/day	10000/day	10000/day	Flexible
Contacts created	1000/day	2000/day	2000/day	2000/day	Flexible
Documents created	250/day	500/day	1500/day	1500/day	Flexible
Email recipients per day	100/day	100/day	1500/day	1500/day	1500/day
Email read/write	20000/day	40000/day	50000/day	50000/day	Flexible
Groups read	2000/day	5000/day	10000/day	10000/day	Flexible
JDBC connection	10000/day	10000/day	50000/day	50000/day	Flexible
JDBC failures	100/day	100/day	500/day	500/day	500/day
Presentat. created	250/day	500/day	1500/day	1500/day	Flexible
Properties read/write	50000/day	100000/day	500000/day	500000/day	Flexible
Spreadsheet created	250/day	500/day	3200/day	3200/day	Flexible
Triggers total runtime	90 min/-day	3 hr/day	6 hr/day	6 hr/day	6 hr/day
Url Fetch calls	20000/day	50000/day	100000/day	100000/day	Flexible

Tabella 3.2: Limitazioni giornaliere di Google Script, from [13]

3.2 Soluzioni ai limiti imposti da Google Script

Google Script impone limiti non trascurabili per il corretto sviluppo dell'applicazione, a tal proposito bisogna elaborare dei meccanismi che permettono di effettuare tutte le procedure descritte durante l'analisi dei requisiti rimanendo entro tali restrizioni.
Per quanto riguarda le limitazioni in spazio non è possibile andare ad intervenire poiché il linguaggio rifiuta tassativamente qualsiasi struttura dati che supera i limiti consentiti, quindi ogni volta che l'utente seleziona un file che non rispetta i limiti, attraverso un alert veniva informato dell'impossibilità di scaricare il file selezionato.
Invece per quanto riguarda il tempo è possibile implementare meccanismi che riescono, anche se in parte, a bypassare tale limitazione permettendo di avere un'apparente esecuzione degli script senza restrizioni temporali.
In GoogleDrivePersonalBackup è stato implementato un metodo per gestire le restrizioni temporali imposte da Google Script, tale procedura è client-side ed è basato sui principi della programmazione concorrente, considerando come critical-section il server e le richieste di download possono essere eseguite solo nel caso in cui il server non stia eseguendo altre richieste.
Per svolgere questo meccanismo si è implementato un oggetto Javascript che svolge la funzione di monitor, ossia controlla la disponibilità della critical section, se ad una richiesta di accesso è libera allora viene approvata, altrimenti la richiesta viene messa in coda e appena la critical section è di nuovo disponibile, viene avviata la richiesta messa in sospeso. La coda è di tipo FIFO (first-in-first-out), di conseguenza la richiesta che viene prelevata dalla coda è quella che da più tempo è in attesa.
Il monitor è stato implementato utilizzando una variabile booleana che indica la disponibilità della critical section e un listener che si attiva ogni volta che tale variabile cambia valore, attivando procedure diverse a seconda del valore letto.
Per quanto riguarda il problema della dimensione massima degli allegati si è deciso di adottare 2 politiche differenti a seconda di come è stato creato un

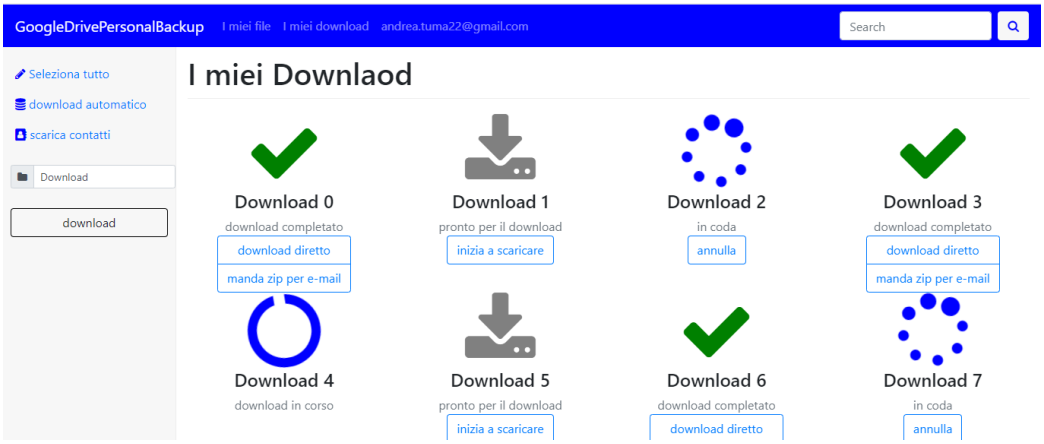

Figura 3.1: Accodamento dei download.

pacchetto:

- se il pacchetto è stato creato manualmente allora prima di avviare la procedura di invio della mail si controlla la dimensione dell'allegato, se supera i 25Mb viene visualizzato un alert che informa l'utente dell'impossibilità di mandare il pacchetto per e-mail;
- Se il pacchetto è stato creato automaticamente non si pone nessun problema in quanto il sistema di creazione automatica è stato fatto in modo che la dimensione dei pacchetti non superi i 25Mb.

Capitolo 4

Architettura del software

L'architettura dell'applicazione è quella di una standard web-app e si basa su una continua comunicazione client-server.
Le 3 componenti del client descritte nel capitolo 2 sono rappresentate nei seguenti file:

- Index.html: file Html che contiene il layout dell'app;

- Stylesheet.html: foglio di stile di Index.html, google script non supporta i file di estensione css, ma permette di unire più documenti Html, per questo motivo il file ha estensione .html e contiene solamente il tag <style> con dentro il css, il comando che effettua l'unione è "?<?!=include('Stylesheet'); ?>" presente nell'head di Index.html;

- scripts.html: insieme di script che effettuano le chiamate al server e gestiscono il dom dell'app.
  Analogamente a "Stylesheet.html", "scripts.html" contiene solo il tag script con all'interno il codice ed è collegato a Index.html tramite il comando "<?!= include('scripts'); ?>" presente alla fine del body del template.

Il server è formato da 3 file:

- onLoad.gs: file che gestisce il corretto avviamento dell'app, caricando il file "Index.html" ed effettuando il linking con i file "Stylesheet.html" e "scripts.html";

- services.gs: file che contiene tutti i servizi che il client utilizza per l'esecuzione dell'app;

- conversions.gs: file che si occupa di gestire tutte le conversioni dei file google in modo da poter essere scaricate.

Oltre che i file per lo sviluppo client e server dell'applicazione, è presente anche un file chiamato chiamato manifest ("appscript.json"), il quale viene creato al momento della creazione del progetto e contiene le seguenti informazioni:

- timeZone: il fuso orario;

- dependencies: le varie dipendenze dell'applicazione, nel nostro caso è presente un solo campo ("enabledAdvancedServices") che ci permette di usare servizi avanzati di Google Drive App;

- webapp: indica le modalità di accesso al progetto in quanto webapp, nel nostro caso l'accesso è consentito a chiunque e l'esecuzione è collegata all'account dell'utilizzatore;

- exceptionLogging: indica dove verranno registrate le eccezioni, nel nostro caso vengono segnate nello Stackdriver;

- oauthScopes: indica tutte le autorizzazioni che l'utente deve concedere al fine di utilizzare l'app.
  Come ogni servizio offerto da Google, infatti, il sistema deve segnalare all'utente che accederà ai suoi dati personali presenti nella Google Suite per avviarsi.

Nel nostro caso gli gli accessi che l'applicazione effettua sono:

- https://www.googleapis.com/auth/userinfo.email: accesso all'indirizzo e-mail dell'utente, senza questo dato è impossibile far funzionare l'app;
- https://www.googleapis.com/auth/drive: accesso a Google Drive;
- https://www.google.com/m8/feeds: permette l'accesso ai contatti dell'utente;
- https://sites.google.com/feeds: accesso a Google Sites, in questo modo è possibile effettuare il download anche dei documenti in Google Sites;
- https://www.googleapis.com/auth/gmail.send: accesso al servizio GMail, indispensabile per offrire il servizio di invio mail con il pacchetto zip allegato;
- https://www.googleapis.com/auth/script.external_request: permesso ad eseguire script esterni;
- https://www.googleapis.com/auth/forms: permesso ad accedere a Google Forms per lo stesso motivo di Google Sites;
- https://www.googleapis.com/auth/documents: accesso a tutti i documenti Google dell'utente.

4.1 Struttura client-side

La parte client-side si occupa di collegare utente e sistema, avviando le richieste in base alle scelte effettuate da chi utilizza l'applicazione.
All'interno della web-app Il client svolge 2 ruoli:

- richiedere al server i dati riguardo il Drive e contatti dell'utente e strutturarli in modo tale da essere chiari e accessibili su interfaccia web;
- offrire meccanismi e procedure per permettere all'utente di effettuare le operazioni di download e backup desiderate.

4.1.1 Variabili globali utilizzate

Introduciamo ora il glossario per indicare le variabili principali della struttura client:

- zip: array di file che apparterranno ad un pacchetto zip, contiene i file che devono essere scaricati;
- files: tutti i file del drive dell'utente, ogni file è rappresentato da:
  - Id: identificativo univoco del file sul Drive;
  - name: il nome del file;
  - size: la dimensione del file in byte;
  - owner: il nome del proprietario del file;
  - dateCreated: la data di creazione del file;
  - parent: nome della cartella in cui il file è contenuto;
  - mimeType: tipo Mime del file.
- folderSize: dimensione della cartella da comprimere e scaricare;
- selectedAllItems: valore booleano che indica se sono stati selezionati tutti gli i file del drive;
- currentFolder: cartella corrente aperta dall'utente, inizializzata a "root", la quale indica la cartella principlale del Drive dell'utente;
- folders: array che contiene le cartelle presenti nel drive dell'utente, ogni elemento di questo array è formato dai seguenti campi:
  - name: nome della cartella;
  - parents: il padre della cartella (dove essa è inserita);
  - owner: il proprietario della cartella;
  - dateCreated: la data di creazione della cartella.

- downloads: array dei download effettuati nella sessione, ogni elemento di questo array è un oggetto con i seguenti campi:
  - name: nome del pacchetto zip (può essere deciso dall'utente);
  - files: il contenuto del pacchetto;
  - status: lo stato del pacchetto, può essere:
    - download in corso, ossia il client aspetta la risposta dal server del termine del download;
    - download in coda, ossia il pacchetto aspetta il completamento di tutti i download prima di lui;
    - download terminato con successo, ossia il download è stato effettuato correttamente;
    - download non riuscito, ossia non è stato possibile completare il download dal server.
- contacts: array con i contatti dell'utente, ogni contatto contiene:
  - mail: indirizzo E-mail;
  - nickName: nickName del contatto;
  - name: il nome completo del contatto;
  - cel: il numero di telefono del contatto.
- fileReady: array di pacchetti che sono stati scaricati dal server, su ogni elemento di questo array è possibile effettuare un download diretto oppure mandarlo alla propria mail.
  Ogni elemento dell'array ha 2 campi:
  - blob: pacchetto zip in formato blob, serve per il download diretto;
  - base: pacchetto zip cifrato in base64, serve per l'invio del file via mail;
- idDownload: variabile inizializzata a 0 che fornisce un id di sessione ai pacchetti zip creati;

- activeList: valore stringa che indica la schermata attiva e di conseguenza la lista da considerare per la funzione di ricerca, può avere i valori di:
  - download: la lista attiva è la lista dei download;
  - files: la lista attiva è quella dei files del drive;
  - contacts: la lista attiva è quella dei contatti.
- queue: array contenente tutti i pacchetti zip che attendono di essere scaricati;
- toGo: oggetto che svolge la funzione di monitor, ossia mette in queue tutti i pacchetti zip da scaricare e appena viene effettuato un download, estrarre un pacchetto da queue e avvia il download.
  Questo oggetto contiene una variabile booleana che ha lo scopo di indicare se è possibile o no effettuare un download e un listener che attiva una funzione ogni volta che il booleano cambia valore.
  Per quanto riguarda il problema causato da Google Form, Google Maps e Google sites è stato risolto creando un documento Html attraverso il servizio avanzato di Google Drive con indirizzo il rispettivo elemento di Maps, Form o Sites.
  Tuttavia questo metodo non permette la modifica locale e i cambiamenti effettuati vengono salvati anche nel file nel cloud.

4.1.2 Procedure implementate

Analizziamo ora tutte le procedure che compongono la struttura client-side:

- Lo script di caricamento viene avviato appena si accede e ha lo scopo sia di richiedere al server tutti i dati necessari per l'esecuzione del'app (file, cartelle e contatti), sia di avvisare l'utente che l'applicazione è in fase di caricamento.

Una volta terminato il caricamento si avvia la procedura "onSuccess(data)" che inserisce i dati ottenuti dal server nelle rispettive tabelle.
Lo script di caricamento ha anche lo scopo di mettere in ascolto la barra di ricerca, inizializzando:

  - "activeList" a "files";
  - una variabile "text" vuota che conterrà la stringa della barra di ricerca;
  - un array "searchedList" che ha lo scopo di contenere i contenuti filtrati dalla ricerca.

- Il meccanismo di ricerca è di tipo dinamico, ossia ogni volta che viene inserito un carattere nella barra di ricerca, esso si avvia ricercando tutti i valori che contengono nel loro nome in caso di "files" e ?downloads", nel loro indirizzo mail in caso di "contacts", la stringa "text".
  Controllando il valore di "activeList", lo script sa quale lista sta visualizzando l"utente e di conseguenza quali dati andare a filtrare.
  Una volta filtrati i dati e inseriti in "searchedList", si avvia una procedura che ha lo scopo di sostituire i valori della tabella nel documento html con quelli di "searchedList", a seconda della lista filtrata, viene invocata una delle 3 procedure seguenti:

  - "populateFiles(list)" che aggiorna la tabella dei file;
  - "populateContacts(list)" che aggiorna la tabella dei contatti;
  - "populateDownload(list)" che aggiorna la tabella dei download.

  Prima di avviare la relativa procedura di inserimento, viene resettata la tabella, onde evitare di far visualizzare all'utente valori non desiderati.
  Se viene cancellato il contenuto della barra di ricerca, e di conseguenza il valore di "text" è la stringa vuota, viene ripristinata la lista di partenza, ossia quella presente al caricamento della pagina.
  Lo script tiene conto anche di eventuali selezioni effettuate dall'utente durante la creazione del pacchetto;

- PopulateFiles(list, folder) è la procedura che aggiorna il dom in base ad una lista data come parametro di input il nome di una cartella stampando nel documento Html solamente i file e cartelle presenti in "folder".
  Tale funzione utilizza come variabili locali:
  - "listItem": un oggetto jQuery che rappresenta una riga della tabella;
  - "ic": un oggetto jQuery che rappresenta la prima colonna della tabella.

  Per ogni elemento di "list", che ricordiamo essere un oggetto con la stessa struttura di un elemento dell'array "files", viene creata la riga rispettiva della tabella inserendo in "listItem":
  - la colonna "ic" che conterrà l"icona restituita dalla procedura "getTypeIcon(id)";
  - la colonna con il nome del file;
  - la colonna con la sua dimensione in Mb, la conversione è effettuata dalla funzione "getMb(value, digit)", se il valore restituito è uguale a 0 al posto della dimensione viene stampato un trattino ("-");
  - la colonna con il proprietario del file;
  - la colonna con la data di creazione del file, parsata dalla funzione "dateParser(date)".

  Una volta terminata la riga viene appesa alla tabella;
- PopulateContacts(list): procedura del tutto analoga a "populateFiles", solo che utilizza i contatti come dati e va a modificare la tabella relativa ai contatti, le modalità di creazione delle righe e colonne della tabella è assolutamente identico a "populateFiles".
- PopulateDownload(list) : procedura che inserisce nella finestra dei download effettuati i downloads presenti nella lista "list". "list" contiene

elementi della stessa struttura di quelli dell'array downloads e per ognuno di essi viene chiamata la funzione "insertDownload(zipItem)" che, come vedremo, ha il compito di aggiungere un download alla finestra dei downloads;

- getMb(value, digit): funzione che approssima un valore numerico "value" ad un numero "digit" di decimali, utilizzato per ottenere il valore in Mb della dimensione dei files.
  La funzione agisce convertendo "value" in stringa, poi ne ricava la sottostringa che contiene "digit" caratteri dopo il punto (".").
  Ottenuta tale stringa, essa viene riconvertita in numero decimale e restituita come valore di ritorno.
  La procedura getMb si basa sulla grammatica di un numero decimale in Javascript, ossia:

$$\begin{gathered} \text{decNum} = \text{num.num} \\ \text{num} = \text{cifra} \mid \text{cifraNum} \\ \text{cifra} = 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{gathered}$$

  In questo modo noi sappiamo che la parte decimale si trova sempre dopo il punto e di conseguenza ci basta trovare la posizione del punto nella stringa che rappresenta il numero da approssimare, e successivamente prendere la sottostringa dall'indice 0 all'indice indexOf(".") + "digit" +1.
  La funzione tiene anche conto di eventuali valori nulli o "undefined", in tal caso "value" viene settato di default a 0;

- dateParser(date): meccanismo che converte la stringa date in formato:

  Day Month NumDay hh:mm:ss TimeZone FullYear

  in una stringa nel seguente formato:

DayNum Month FullYear

La funzione utilizza un oggetto "Date" e un array "monthNames" che contiene il nome dei mesi ordinati (formato stringa).
Il primo passo è convertire la stringa "date" in oggetto "Date", successivamente viene creata la stringa di output utilizzando i seguenti metodi di manipolazione delle date di Javascript:

- getDate() restituisce il giorno;
- getMonth() restituisce il numero del mese (0=gennaio, 11=dicembre); Accedendo al valore dell'array di indice "getMonth()" riusciamo ad ottenere il nome del mese che ci interessa;
- GetFullYear() restituisce l'anno scritto per intero (esempio 2019).

- getTypeIcon(mimeType): questa funzione restituisce un oggetto Jquery contenente un?icona appartenente alla libreria "Font Awesome" in base alla stringa mimeType presente negli elementi di "files".
  La funzione non è altro che una serie di costrutti "if then else" annidati che controllano il valore di mimeType.
  i tipi mime che getTypeIcon riesce ad individuare sono:

  - documenti Microsoft Office;
  - documenti Google;
  - file multimediali (audio, immagini, video);
  - file pdf;
  - archivi (zip e rar);
  - file di script;
  - file di testo (.txt);
  - file eseguibili (.exe);

  Se la funzione non riconosce un tipo Mime, restituisce un'icona di default.

L'icona viene creata inizializzando una variabile locale "icon" ad un oggetto Jquery contenente il tag <i>, e in base al riconoscimento di mimeType viene aggiunta una classe di Font Awesome che dichiara il tipo di icona e uno stile css per il colore;

- Selected(id): procedura che si avvia al click di un elemento della tabella dei files, inserisce nel pacchetto il file cliccato se non è già stato messo (altrimenti lo toglie) e segnala l'utente dell'avvenuta selezione/deselezione colorando di grigio la riga del file cliccato.
  Il primo passo che la funzione effettua è quello di controllare se la riga cliccata è già stata selezionata.
  Questo controllo si effettua controllando se la riga HTML ha la classe "selected", che stampa la riga della tabella di colore grigio.
  A tal scopo si utilizza il metodo "hasClass(class)" di Jquery che restituisce true se l'oggetto ha la class "class", false altrimenti.
  Se la riga della tabella non è stato selezionata prima, allora si aggiunge la classe "selected" all'elemento con il metodo jQuery "addClass(class)" e si cerca nell'array "files" il file corrispondente all'elemento cliccato.
  Una volta trovato, viene inserito l'elemento di "files" in una variabile locale "itm" e si controlla la sua dimensione per assicurarsi che rispetta i vincoli imposti da Google script.
  Se il vincolo è rispettato allora si colora la riga, "itm" viene inserito nell'array "zip" e viene aggiornato folderSize aggiungendo la dimensione del file appena aggiunto.
  In caso di vincolo di dimensione del file non sia rispettato, la selezione non viene compiuta e viene avvertito l'utente tramite la procedura di alert.
  Se invece la riga era già selezionata si toglie la classe "selected" con il metodo Jquery "removeClass(class)", si elimina dall'array zip il file corrispondente e si aggiorna folderSize sottraendo la dimensione del file tolto.
  infine viene aggiornata la lista del pacchetto zip visibile a sinistra dello

schermo appena si seleziona il primo file dalla tabella e vengono stampati il numero totale di file nel pacchetto (ottenibile dalla proprietà "length" di "zip") e la dimensione totale (folderSize) sotto la lista in modo che sia visibile chiaramente all'utente;

- dwFiles(): funzione che chiede conferma all'utente di scaricare il file selezionato. Essa stampa un oggetto "confirm" che in caso di selezione affermativa ("confirm" == "true") avvia il download tramite la procedura "download()", in caso negativo fa continuare all'utente la navigazione nell'app.

- download(): procedura che avvia il download dei file contenuti nell'array "zip".
  La funzione inizialmento controlla il valore della variabile "toGo":

  - se "toGo" è uguale a "true" allora vuoldire che il server non sta preparando alcun download e quindi si può effettuare la richiesta. Prima di effettuare la richiesta "toGo" viene settato a false per indicare che il server è stato chiamato viene creata una variabile locale chiamata "ZipItem", la quale è formata dai seguenti campi:

    * id: valore di "idDownload";
    * name: nome del pacchetto zip
    * files: l'array "zip";
    * status: lo stato del download inizializzato a "download in corso";

    L'oggetto "ZipItem" creato viene inserito nell'array "downloads" e successivamente viene effettuata la richiesta di download al server tramite l'oggetto google runner, il quale ha lo scopo di avviare la procedura "success(data)" in caso di esito positivo o "onFailure(data)" in caso di esito negativo;

  - Se "toGo" è uguale a "false" allora il pacchetto viene messo in coda inserendolo nell'array "queue".
    Per l'inserimento in "queue" possiamo avere 2 casi:
    * il pacchetto zip da inserire in coda si trova nell'array "downloads": questo caso succede quando si effettua la creazione automatica dei pacchetti zip, la quale verrà approfondita successivamente.
      Si ricerca il pacchetto zip in downloads e inserito in "queue";
    * il pacchetto non si trova in "downloads": in questo caso viene creato un pacchetto "zipItem" analogo a quello descritto in precedenza, viene aggiornato "idDownload" incrementando il suo valore di 1 e infine l'intero pacchetto viene inserito in "queue".

  Infine viene aggiornata la schermata di download tramite la funzione insertDownload(id), svuotato il contenuto di "zip", tolta la classe "selected" a tutte le righe della tabella dei files selezionate e avvertito l'utente dell'avvenuto inizio del download tramite la funzione "showSnackBar".

- success(data): procedura avviata al momento in cui il server manda una risposta al client in seguito ad una richiesta di download.
  Per avvisare i download in coda che il server è disponibile "toGo" viene impostato a "True", successivamente si utilizza una variabile file formata da:
  - "base": il pacchetto zip codificato in base64;
  - "blob": il pacchetto zip in formato Blob, ottenuto tramite la procedura b64ToBlob(b64Data).

  Se la stringa in base 64 viene decodificata con successo "file" viene inserito in "fileReady", aggiornato lo stato del download con la stringa "msg" contenuto nel pacchetto di risposta del server (parametro "data" passato in input) e di conseguenza la schermata di download;

- b64ToBlob: funzione importata dalla libreria "b64-to.blob" di NPM che prende in input un file codificato in una stringa in base64 e la decodifica nel blob relativo[14];

- directDownload(index): avvia il download diretto (download di un file via browser) del pacchetto zip presente in posizione "index" nell'array "fileReady".
  La procedura preleva il campo "blob" del pacchetto, genera l'url associato tramite la il metodo "createObjectURL(blob)" dell'oggetto "URL" di Javascript e infine avvia il download utilizzando comando "window.location = URL".
  Il funzionameto di questo script è dovuto da come Javascript gestisce le risorse rappresentate da un url; infatti "window.location" reindirizza il browser alla risorsa specificata dall?url alla parte destra dell'assegnamento, in caso di documento html "window.location" si limita a far apparire il documento nel browser, ma nel caso di una risorsa che il browser non riesce a gestire (esempio i file zip), la procedura ne avvia il download;

- sendMail(index): funzione analoga a "directDownload()" che invece che effettuare il download invia per e-mail il pacchetto zip di posizione index nell'array "fileReady".
  l'invio della mail viene effettuata dal server, "sendMail()" si limita a fare la richiesta di invio della mail.
  Il pacchetto viene mandato codificato in base64 (campo "base" dell'elemento di "fileReady"), assieme al suo nome e tramite la variabile runner viene avviata la richiesta.
  Tramite la funzione showSnackBar viene avvertito l'utente che l'invio della mail è in corso e il relativo esito, sia in caso di successo che di insuccesso;

- showDownloads(): procedura che ha lo scopo di mostrare la finestra con i download in corso ed effettuati dall'utente.

La procedura si basa sulle funzioni Jquery "hide()" e "show()" che mostrano e nascondono gli elementi del dom.
Oltre che organizzare le finestre nascondendo quella dei contatti e quella dei files, viene anche aggiornato "activeList" a "downlaods" per abilitare la barra di ricerca sull'array "downloads";

- showContacts(): procedimento del tutto analogo a "showDownloads";
  In questa funzione ad "activeList" viene assegnato il valore di "contacts";

- showFile(): procedimento analogo a "showContacts()" e "showDownloads()". Il valore di "activeList" viene settato a "files";

- openFolder(name): funzione che si occupa di visualizzare tutto il contenuto di una specifica cartella, viene attivata quando l'utente clicca su una cartella presente nel suo drive.
  La procedura utilizza 2 variabili globali "fil" e "fol" dove inserisce rispettivamente file e folder contenute nella cartella cliccata.
  Per visualizzare i file e cartelle la funzione chiama le procedure "populateFiles" e "populateFolders".
  Infine viene aggiornato il percorso del Drive sopra la tabella;

- selectAll(): funzione che seleziona/deseleziona automaticamente tutte le righe della tabella dei files.
  Per decidere l'operazione da effettuare tra selezionare o deselezionare gli elementi, viene controllato il valore di "selectedAllItems":

  – se la variabile è uguale a "False" allora bisogna selezionare gli elementi aggiungendo la classe "selected" e aggiornando le dimensioni del pacchetto in modo analogo alla funzione "selected(id)";
  – se la variabile è uguale a "True" allora viene tolta la classe "selected" agli elementi.

- updateDownloads(): funzione che aggiorna la schermata di download in base al contenuto dell'array downloads.

Molto semplicemente "updateDownloads()" non fa altro che eliminare tutto il contenuto della schermata di downloads e successivamente richiama la funzione "insertDownload(zipItem)" su ogni elemento dell'array "downloads";

- insertDownload(zipItem): funzione che inserisce nella schermata di download il pacchetto zipItem.
  Per inserire il pacchetto nella schermata di download in modo tale che sia chiara all'utente è creare una box html che contiene:

  – nome del pacchetto;

  – stato del download;

  – azioni possibili per il pacchetto.

  A tal scopo viene utilizzata una variabile locale ?box? che sarà la versione html di "zipItem". Le altre variabili locali utilizzate sono:

  – icon: elemento <i> contenente l'icona dello stato del download e la chiamata della funzione "showInfo(id)" quando esso viene cliccato;

  – Options: array che contiene le possibili azione che si possono effettuare sul pacchetto, le operazioni dipendono dallo stato di download del pacchetto.
    Ogni elemento di "options" è un array di 2 elementi:

    * comando che appare scritto sul bottone, quindi il testo visibile all'utente;

    * Nome vero e proprio della funzione, ossia la procedura invocata.

  – BtnGroups: gruppo di bottoni per attivare le varie funzione contenute in "options".

  Inizializzate le variabili viene utilizzato costrutto "switch case" per inserire l'icona adeguata e le rispettive operazioni possibili, i casi sono:

  - "pronto per il downlaod": il pacchetto è pronto per essere scaricato, l'unica operazione possibile è la procedura "download from(id)";
  - "download in corso": il server sta elaborando la richiesta di download, non è possibile effettuare alcuna operazione, bisogna attendere l'esito del download.
  - "in coda": il server è occupato il pacchetto attende il proprio turno, è possibile rimuoverlo dalla coda tramite la procedura "removeFromQueue()".
  - "download completato": il download è stato completato con successo, è possibile effettuare un download diretto del pacchetto ("directDownload(id)") oppure spedirlo al proprio indirizzo di posta ("sendMail(id)");
  - "download fallito": il download non è stato completato, questo accade quando:
    * ci sono problemi di connessione.
    * Il pacchetto ottenuto dal server è vuoto (può accadere quando si inseriscono nel pacchetto file di cui non si hanno i diritti).

  Le 2 operazioni possibili sono ritentare il download("retry()") o annullarlo rimuovendo l'elemento dalla schermata ("remove(id)").
  Le icone utilizzate appartengono alla liberia "Font Awesome".
  Terminata la scelta dell'icona e delle operazioni viene popolata la variabile "box" e poi inserita nella schermata dei download;

- getIcon(id): funzione che restituisce l'icona di un pacchetto zip nell'array "downloads" di posizione id.
  In base allo stato del pacchetto si crea l'icona in modo analogo alla funzione "insetDownload", usando una variabile locale "icon" che contiene il tag <i> con la classe della libreria "font awesome" per avere l'icona specifica;

- remove(id): rimuove un download dall'array "downloads" e tramite la funzione "updateDownloads()" aggiorna la schermata dei download. Questo metodo può essere chiamato dall'utente nel caso in cui il download di un pacchetto zip sia fallito e si desidera eliminarlo;

- removeFromQueue(id): funzione analoga a "remove(id)" che rimuove l'elemento di indice "id" dall'array "queue".
  Prima di eliminare l'elemento, il suo stato viene settato da "in coda" a "pronto per il download" per avvisare l'utente che il download del pacchetto non è più in attesa.
  Alla fine viene aggiornata la finestra dei download tramite "updateDownloads()";

- showInfo(id): procedura che mostra le informazioni su un pacchetto zip, viene avviata quando si clicca l'icona dello stato di download di un pacchetto zip.
  Le informazioni mostrate sono:

  - Nome del pacchetto;
  - Stato del download del pacchetto;
  - Contenuto del pacchetto (numero di file presenti ed elenco dei file);
  - eventuali files che non è stato possibile scaricare.

  Il principio base del funzionamento di "showInfo(id)" è quello di creare elementi Html con le informazioni sul pacchetto e inserirli nell'apposita box del documento "index.html". Le variabili locali utilizzate sono:

  - obj: elemento di downloads di indice "id" passato come parametro;
  - var icon: icona dello stato del download di obj, ottenibile tramite la procedura "getIcon(id)";
  - iconBox: elemento dove verrà inserita la variabile icon;
  - status: elemento <p> contenente lo stato di download di "obj";

- name: elemento <p> contenente il nome di "obj"
- notAuthoredFiles: lista dei files che non è stato possibile includere nel pacchetto;
- fileList: lista dei files inseriti con successo nel pacchetto;
- zipLenght: lunghezza del pacchetto, ottenibile sottraendo alla lunghezza dell'array "files" di obj la lunghezza dell'array "notAuth".

Una volta inizializzate le variabili vengono effettuati 2 cicli for per popolare con il nome dei files le rispettive liste "fileList" e "notAuthoredFiles".
Per quanto riguarda il popolamento della lista "nothAuthoredFiles" basta creare l"apposito elemento html <li> per ogni elemento di "notAuth", ma per quanto riguarda "fileList" la situazione è più complicata; infatti bisogna utilizzare l'array "files" di "obj" e per ogni elemento bisogna controllare che non sia presente anche in "notAuth", solo in tal caso viene inserito.
Per effettuare il controllo si utilizza la funzione "findItemById(arr, id)" che restituisce l'indice dell'elemento presente in arr di indice ?id? se esiste, altrimenti restituisce "-1".
Infine, tramite la funzione "append()" di Jquery, tutte le variabili create e popolate adeguatamente vengono inserite nella box html designata e mostrata all'utente tramite la procedura "show()" di Jquery.

- closeInfo(): procedura che chiude la schermata delle informazioni su un pacchetto zip.
  La funzione si avvia al click dell'icona di chiusura presente nella finestra e utilizza il metodo "hide()" di Jquery;

- retry(id): permette all'utente di effettuare nuovamente un download fallito in precedenza senza ricreare il pacchetto da zero;

- autoDownload(): funzione che si occupa della creazione automatica dei pacchetti zip.
  La procedura utilizza 2 variabili globali.

  - downloadArr: un array di pacchetti, conterrà tutti i pacchetti zip creati, in poche parole questa variabile conterrà tutto l'archivio di Google Drive;
  - sizep: contatore per conoscere la dimensione di un pacchetto, serve per la creazione dei pacchetti.

  L'obiettivo di questa funzione è quello di coprire tutto il Google Drive creando il minor numero di pacchetti possibile.
  La prima idea può essere quella di mettere tutto l'archivio in un unico pacchetto zip, tuttavia ciò è impossibile per le restrizioni imposte da Google Script e andrebbe contro l?obiettivo dell'applicazione, ossia quello di fornire uno strumento di backup personalizzato sia totale che parziale.
  L'altra alternativa è quella di decidere una dimensione fissa uguale per tutti i pacchetti e di riempire ogni pacchetto il più possibile, ossia che la dimensione totale dei file in un pacchetto si avvicina il più possibile alla dimensione stabilita.
  Per nascondere all'utente i limiti di Google Script, la dimensione dei pacchetti è stata impostata nel sistema ed è uguale a 25 Mb.
  Il motivo per cui la dimensione è stata impostata a 25 Mb e non di più non è per i limiti di Google Script (infatti il linguaggio permette di creare pacchetti zip anche superiori a 50Mb) ma per fare in modo che ogni pacchetti creato possa essere sia scaricato da browser che essere spedito per e-mail, infatti GMail impone un limite di massimo di 25Mb di dimensione totale degli allegati.
  Gli utilizzatori di GMail sanno che è possibile inserire allegati nelle mail superiori a 25Mb, ma per fare questo il sistema di posta non inserisce i file "fisicamente", ma li inserisce nel google drive e in allegato inserisce

il link ai file, tuttavia l'applicazione effettua il backup di Google Drive e non avrebbe senso mandare per e-mail il link ad un archivio che si vuole backuppare.
La prima fase quindi è un problema analogo al famoso problema dello zaino, il quale consiste di riempire uno zaino di dimensione n con un numero m di oggetti di dimensione d tale che la somma di tutti i d ¡ n che m sia massimo.
Nel nostro caso lo zaino è il pacchetto zip e i nostri elementi sono i file, per ogni pacchetto dobbiamo inserire il maggior numero di file senza superare i 25Mb.
Per implementare questo meccanismo è stata utilizzata una tecnica greedy, quindi prima i file vengono ordinati in ordine crescente, poi si prelevano gli elementi dell'array e si inseriscono su "zip" e si aggiorna sizep.
Quando sizep raggiunge i 25 Mb, il contenuto di zip viene inserito in una cella di downloadArr, questo significa che il pacchetto è stato creato.
Resettato il valore di sizep si e svuotato l'array zip si procede con il pacchetto successivo.
Una volta creati i pacchetti viene chiesto all'utente di procedere informandolo sul numero di pacchetti creati.
Se l'utente conferma potrà vedere nella schermata di download i pacchetti creati con il nome inserito più un numero che indica la posizione del pacchetto.
Ogni pacchetto ha lo stato "pronto per il download", è possibile scaricare un pacchetto cliccando il bottono sottostante;

- downloadFrom(id): funzione che scarica un pacchetto zip nell'array "downloads" di indice id, utilizzata per avviare il download dei pacchetti in coda o per riavviare eventuali downloads falliti.
  Il funzionamento è praticamente uguale alla funzione "download"; infatti la chiamata al server e le rispettive funzioni di callback sono le

stesse, come anche le procedure di aggiornamento degli elementi html;

- showSnackBar(text): funzione che mostra il testo in input "text" tramite un oggetto "snackbar" (tipico delle applicazioni android).
  L'oggetto snackBar è stato implementato in HTML e css e imita sia l?aspetto che il funzionamento della sua versione originale presente nelle librerie di sviluppo di applicazioni Android.
  La sua costruzione HTML è semplicemente un tag che ha come contenuto il testo del messaggio e tramite il css viene gestisce sia l"aspetto che le rispettive animazioni di entrata e di uscita.
  Il tempo di apparsa dello SnackBar viene gestito dalla funzione stessa utilizzando il costrutto di Javascript "setTimeout(function, delay)" che rimpiazza la classe "show" (responsabile dell'apparizione l'oggetto) con la stringa vuota; in questo modo lo SnackBar scompare dalla schermata dell'app.

4.2 Struttura server-side

Il server ha l'obiettivo di comunicare con i database di Google per ottenere tutti i dati relativi al Drive e Contacts dell'utente e rispondere alle richieste del client.
A seconda della richiesta il server richiede a Google le informazioni necessarie tramite le funzioni che offre il linguaggio Google script, elabora i dati ottenuti e manda la risposta il client in formato JSON.
Le parti che compongono il server sono 3:

- "OnLoad.gs" che si occupa di effettuare il caricamento del documento HTML;
- "Services.gs" che gestisce le richieste del client;
- "conversions.gs" che effettua le conversioni dei file di Google in formati equivalenti che possono essere visualizzati e modificati anche in ambiente globale.

4.2.1 Procedure implementate

"OnLoad.js" è formato semplicemente da 2 funzioni che vengono invocate al caricamento della pagina e hanno lo scopo di unire i file javascript e css al documento html e caricare nel browser quest'ultimo.
Le 2 le procedure sono funzioni incluse in Google script e sono:

- doGet(e): procedura che si avvia ogni volta che il client fa una richiesta di metodo GET al server.
  Per mandare la pagina html viene utilizzato l'oggetto "HtmlService", che permette di gestire i documenti html dell'applicazione.
  Tramite il metodo "getTemplateFromFile(file)" viene generato il template partendo dal file html e con la procedura "evaluate()" viene inizializzata la funzione di merge con i file css e javascript e genera l'output Html visualizzabile dal browser.
  Il parametro "e" è l'oggetto che contiene l'url inserito dal client nella richiesta, nel nostro caso non serve perchè la pagina Html da caricare è sempre la stessa, indipendentemente dalla stringa di url;

- include(fileName): funzione che effettua l'inclusione dei file javascript e css nell'html, tale funzione è invocata nel file "index.html" e inserisce i file "stylesheets" e "scripts" nelle posizioni rispettive alla chiamata della procedura nel documento HTML.
  Anche in questo caso viene utilizzato HtmlService richiamando il metodo "createHtmlOutputFromFile(filename)" che genera l'output e successivamente "getContent()" per prelevare il contenuto del documento.

"Services.gs" è la parte del server in comunicazione con il client, si occupa di ricevere le richieste ed elaborare una risposta in formato JSON interrogando i server di Google Drive per ottenere i dati necessari.
Le richieste che "Services.gs" soddisfa sono:

- richiesta dell'elenco dei file contenuti nel Google Drive dell'utente;

- richiesta dell'elenco dei contatti contenuti nel Google Contacts dell'utente;
- creazione di un pacchetto zip con un elenco di file arbitrario, delegando a "conversions.gs" il compito di effettuare le dovute conversioni nel caso di file Google;
- spedire una mail con allegato il pacchetto zip scelto dall'utente, quando è possibile.

Descriviamo ora nel dettaglio le procedure che soddisfano tali richieste:

- la creazione dell'elenco dei file e contatti appartenenti al Drive e Contacts dell'utente è affidata alla funzione "getFiles()": questa procedura utilizza principalmente 2 variabili locali, "data" che sarà l'array che conterrà l'elenco dei file e "contact" che contiene i contatti.
  La directory root del drive viene prelevata utilizzando la funzione "getRootFolder()", la quale utilizza l'oggetto DriveApp di Google Script per ottenere la directory desiderata. I contatti vengono prelevati in maniera del tutto analoga alla cartella root, utilizzando però l'oggetto ContactsApp. La prima computazione che viene effettuata da "getFiles()" è quella estrapolare da ogni file le informazioni necessarie per permettere al client di svolgere le procedure di integrazione con il Dom. Per ogni file viene creato un oggetto (identificato nel codice dalla variabile "file") con questi campi:
  - id: necessario per identificare un file in modo univoco
  - name: il nome del file:
  - size: la dimensione del file in byte;
  - owner: il proprietario del file;
  - dateCreated: la data di creazione del file;
  - mimeType: il tipomime del file.

Una volta creato tale oggetto, esso viene messo nell'array "data".
Tutti i campi sono facilmente ottenibili grazie ai meccanisi built-in di Google Script, che offre un?interfaccia molto intuitiva per gestire tutti i dati di file e cartelle di Google Drive.
Analogamente viene applicata la stessa procedura per i contatti.
Per ogni contatto viene prelevato:

- indirizzo e-mail principale;
- nome;
- nickName;
- numero di telefono.

Infine viene creato il messaggio JSON da spedire al client (variabile "msg") contenente:

- indirizzo mail dell'utente;
- array data;
- array contacts.

Questo oggetto viene convertito in JSON e spedito al client;

- La creazione del pacchetto zip è affidata alla funzione "downloadFiles()": per creare il pacchetto bisogna tenere in considerazione che l'utente può scaricare soltanto i file di cui ha il permesso (ricordiamo che oltre che i file inseriti nel drive dall'utente ci sono anche i file che ha in condivisione con altri utenti) e che i pacchetti zip non ammettono 2 file con lo stesso nome.
  Per ottenere il contenuto di ogni file l'interfaccia di Google Script presenta un metodo chiamato "getBlob()" che restituisce il contenuto di un file in formato Blob, di conseguenza è possibile iterare questa procedura per ogni file per ottenere il rispettivo Blob.

Prima di richiedere il Blob, viene controllato se il file può essere scaricato dall'utente, per fare questo si utilizza la procedura "isDownloadble(file)" che restituisce "True" se il file è scaricabile, altrimenti false.
Una volta verificati l'autorizzazione al download, Il contenuto blob viene inserito in un array chiamato "blobs". Prima di spedire all'utente l'array "blobs" viene controllato per correggere eventuali nomi uguali; per fare questo per ogni file si controlla se esiste un altro file nell'array con lo stesso nome e se viene trovato, allora viene aggiunto un carattere al nome del file in considerazione;

- L'invio della mail è affidato alla funzione "sendMail(zip)": la procedura richiama il servizio GMailApp che prepara una mail che ha come destinatario l'indirizzo di posta dell'utente, come body un messaggio standard e come allegato il parametro di input "zip".
  L'input è un oggetto in formato JSON con i seguenti campi:

  – name: il nome del pacchetto, assegnato dall'utente al momento della creazione;

  – data: contenuto del pacchetto codificato in b64.

  Tramite la classe "Utilities" si ottiene il blob del pacchetto zip e con "GMailApp.sendMail()" si crea e spedisce la mail;

- il download di una cartella viene effettuato dalla procedura "downloadFolder(folderName)" che ricerca nel drive la cartella di nome folderName, poi tramite "Utilities.zip()" viene creato il pacchetto.
  Per creare il pacchetto bisogna estrapolare tutti i file della cartella, a tal proposito si utilizza la funzione "getBlob(folder, path)".
  Questa procedura esplora la struttura ad albero della cartella "folder" passata come parametro e inserisce all'interno di un array tutti i file assieme al loro percorso ("path") in modo tale da mantenere l'ordine dei file durante la compressione della cartella.

"Conversions.gs" si occupa delle conversioni; infatti per poter scaricare i file di Google è necessario che vengano convertiti in formati analoghi in modo da poter essere eseguiti localmente.
Il principio base con cui opera "Conversions.gs" è il seguente:

- Preleva l'url del file da convertire nel formato desiderato;
- Crea un altro file utilizzando l'url e l'OauthToken della sessione in corso per l'autenticazione dell'utente;
- restituisce il file creato in formato "Blob".

Tale meccanismo è stato implementato utilizzando i servizi Avanzati di Google, in quanto i servizi G-Suite non consentivano la creazione di file.
Descriviamo ora le funzioni implementate in questa componente:

- convert(documentId): cuore di "Conversions.gs", è la funzione chiamata da "service.gs" per effettuare le conversioni.
  La funzione preleva il file di id "documentId" e il suo rispettivo tipo Mime.
  Lo sviluppo poi è un semplice "switch case" basato sul tipo Mime del file.
  Le conversioni che la procedura effettua sono:
  - Google Docs in docx (Microsoft Word);
  - Google Sheets in xlsx (Microsoft Excel);
  - Google Slides in pptx (Microsoft Powerpoint);
  - Google Drawing in jpeg;
  - Google Forms, Google Sites e Google Map in html;
  - Google Script in JSON.

  Nel caso di default si restituisce semplicemente il "Blob" del file;

- getBlob(documentId, type): procedura che crea il file di id "documentId" convertito in tipo Mime "type" e restituisce il "Blob".
  Vengono utilizzate 4 variabili locali:

  – "file" che contiene il file di id "documentId", viene utilizzata la procedura "Drive.Files.get(id)" appartenente al servizio avanzato di Google Drive;

  – "url" che contiene l'url del file di tipo mime Type, ottenibile con "ExportLinks", array che contiene tutti gli url del file selezionato per ogni tipo Mime che Google ammette per quel determinato file;

  – "oauthToken": token riferito alla sessione in corso dell'utente, ottenibile tramite il metodo "getOauthToken()" della classe "ScriptApp";

  – "response": file creato utilizzando la variabile "url".
  Viene chiamato il servizio di Google Script "UrlFetch", che consente di ottenere una risorsa specificando l'url. Viene aggiunto anche un header con il token per autenticare l'utente, in questo modo il sistema riconosce il proprietario del file e ne consente il prelievo.

  La procedura restituisce il blob di "response" effettuando "response.getBlob()".

Conclusioni

L'obiettivo principale di questo lavoro era quello di fornire un servizio che permette agli utenti di Google Drive di gestire il backup dei loro file in base alle loro esigenze. Il risultato finale è una web-app che consente di creare pacchetti zip di dimensioni arbitraria con all?interno i file selezionati dal proprio Google Drive e successivamente di scaricare tali pacchetti; in questo modo l'utente ha pieno controllo del backup dando priorità a ciò a cui è interessato maggiormente.
La possibilità di effettuare un backup diviso in pacchetti consente anche di salvare sul proprio dispositivo i file durante il processo, in tale modo è possibile avere sotto controllo lo svolgimento del processo potendo rimediare ad eventuali errori senza dover ricominciare da capo.
L'applicazione creata consente anche di effettuare un backup dei propri contatti, salvati su un file in formato pdf consultabile direttamente da un qualunque dispositivo con un visualizzatore pdf all'interno.
Per lo svolgimento di tale lavoro si è ricorso ad utilizzare gli strumenti che Google offre per gli sviluppatori, tra cui il linguaggio Google Script che ha permesso l'implementazione di procedure che interagiscono direttamente con i database di Drive con poche righe di codice senza dover effettuare chiamate Rest.
Lo sviluppo dell'applicazione si può dividere in 2 step:

- il primo riguarda la progettazione server-side per interagire con Google Drive;

- il secondo la progettazione client-side per la creazione dell?interfaccia utente.

Avendo usato come ambiente di sviluppo Google Apps Script, è stato possibile integrare facilmente tutti i meccanismi di autenticazione e di sicurezza secondo gli standard Google e questo ha permesso di avere un prodotto che può essere integrato sia con Google Chrome come estensione, sia come componente aggiuntivo di Google. Attualmente GoogleDrivePersonalBackup non è presente nello store di Google Chrome in quanto è ancora in fase di verifica da parte di Google.
Una volta vertificata sarà possibile scaricare l'applicazione come estensione di Google Chrome, fino a quel momento è possibile accedere al servizio tramite il termine sottostante:

GoogleDrivePersonalBackup

Appendice A

OnLoad.gs

```
//carica la pagina index.html
function doGet(e) {
    return HtmlService.createTemplateFromFile('index')
        .evaluate();
}

//inclusione file Stylesheet in html
function include(filename) {
    return HtmlService.createHtmlOutputFromFile(filename)
        .getContent();
}
```

Appendice B

Services.gs

```
function getRootFolder() {
    return DriveApp.getRootFolder();
}

function getFiles() {
    var root = getRootFolder();
    var data = [];
    var contacts = ContactsApp.getContacts();

    var files = DriveApp.getFiles();
    while (files.hasNext()) {
        var file = files.next();
        var parents = [];
        var p = file.getParents();
        while (p.hasNext()) {
            var par = p.next().getName();

            parents.push(par);
        }
        file = {
            id: file.getId(),
            name: file.getName(),
            size: file.getSize(),
            parent: parents,
```

```
            owner: file.getOwner().getName(),
            dateCreated: file.getDateCreated(),
            mimeType: file.getMimeType()
        };
        data.push(file);
    }
    for (var i = 0; i < contacts.length; i++) {
        var phones;
        try {
            phones = (contacts[i].getPhones()[0].getPhoneNumber
                ());
        } catch (err) {}
        contacts[i] = {
            id: contacts[i].getId(),
            mail: contacts[i].getEmails()[0].getAddress(),
            nickName: contacts[i].getNickname(),
            name: contacts[i].getFullName(),
            cel: phones
        }
        var folders = DriveApp.getFolders();
        var fold = [];
        while (folders.hasNext()) {
            var f = folders.next();
            var parents = []
            var p = f.getParents();
            while (p.hasNext()) {
                var par = p.next().getName();

                parents.push(par);
            }
            var directory = {
                name: f.getName(),
                parents: parents,
                owner: f.getOwner().getName(),
                dateCreated: f.getDateCreated()
            }
            fold.push(directory);
        }
```

```
        var msg = {
            array: data,
            user: Session.getActiveUser().getEmail(),
            contacts: contacts,
            folders: fold
        }
        return JSON.stringify(msg);
    }
}

function getMyDrive(rootFolder, path) {
    Logger.log(rootFolder);
    var blobs = [];
    var files = rootFolder.getFiles();
    while (files.hasNext()) {
        var file = files.next();
        file = {
            id: file.getId(),
            name: file.getName(),
            size: file.getSize(),
            owner: file.getOwner().getName(),
            dateCreated: file.getDateCreated(),
            mimeType: file.getMimeType()
        };
        blobs.push(file);
    }
    var folders = rootFolder.getFolders();
    while (folders.hasNext()) {
        var folder = folders.next();
        var fPath = " ";
        blobs = blobs.concat(getMyDrive(folder, fPath));
    }
    return blobs;
}
```

```
//prende un array di file in input, lo conferte in blob e crea
   lo zip, ignora i file di cui l'utente non Ã¨ o owner o
   editors
function downloadFiles(zipItem) {
    var blobs = [];
    var notAuthorized = [];

    for (var i = 0; i < zipItem.files.length; i++) {
        try {
            if (isDownloadble(zipItem.files[i].id)) {
                var blob = convert(zipItem.files[i].id);
                blobs.push(blob);
            } else {
                notAuthorized.push(zipItem.files[i]);
            }
        } catch (err) { notAuthorized.push(zipItem.files[i]); }
    }


    for (var i = 0; i < blobs.length; i++) {
        for (var j = 0; j < blobs.length; j++) {
            if (i != j) {
                if (blobs[j].getName() == blobs[i].getName()) {
                    blobs[j].setName(blobs[j].getName().split
                        (".")[0] + "()." + blobs[j].getName().
                        split(".")[1]);
                }
            }
        }
    }

    //caso pacchetto vuoto, tutti i file inseriti non possono
       essere scaricati
    if (blobs.length != 0) {
        var zipped = Utilities.zip(blobs, zipItem.name + '.zip')
           ;
        zipped = Utilities.base64Encode(zipped.getBytes());
        var statusText = "download completato";
```

```
    } else {
        var zipped = "noData";
        var statusText = "download non riuscito";
    }

    var response = {
        folder: zipItem.id,
        msg: statusText,
        notAuth: notAuthorized,
        item: zipped
    }

    return JSON.stringify(response);
}


//controlla se l'utente ha i diritti per scaricare il file con
   id fileId
function isDownloadble(fileId) {
    var editors = DriveApp.getFileById(fileId).getEditors();
    var owner = DriveApp.getFileById(fileId).getOwner();
    var okDownload = false;
    for (var i = 0; i < editors.length; i++) {
        if (editors[i].getEmail() == Session.getActiveUser().
           getEmail()) okDownload = true;
    }
    if (owner.getEmail() == Session.getActiveUser().getEmail())
       okDownload = true;
    return okDownload;
}

//manda una mail all'user con allegato un pacchetto zip
function sendGMail(zip) {
    zip = JSON.parse(zip);
    var b = Utilities.newBlob(Utilities.base64Decode(zip.data),
       "application/zip").setName(zip.name + ".zip");
    GmailApp.sendEmail(Session.getActiveUser().getEmail(), "
       drive test", "Download your drive here", {
```

```
        attachments: [b],
        name: 'Automatic Emailer Script'
    });
    return JSON.stringify("ok");
}

function test() {
    Logger.log(DriveApp.getFolders().next());
}


function downloadFolder(item) {
    var folder = DriveApp.getFoldersByName(item.name).next();
    var zipped = Utilities.zip(getBlobs(folder, ''), folder.
       getName() + '.zip');
    zipped = Utilities.base64Encode(zipped.getBytes());
    var statusText = "download completato";

    var response = {
        folder: item.id,
        msg: statusText,
        item: zipped
    }

    return JSON.stringify(response);
}

function getBlobs(rootFolder, path) {
    var blobs = [];
    var names = {};
    var files = rootFolder.getFiles();
    while (files.hasNext()) {
        var file = files.next();
        file = convert(file.getId());
        var n = file.getName();
        while (names[n]) { n = '_' + n }
        names[n] = true;
        blobs.push(file.setName(path + n));
```

```
    }
    names = {};
    var folders = rootFolder.getFolders();
    while (folders.hasNext()) {
        var folder = folders.next();
        var n = folder.getName();
        while (names[n]) { n = '_' + n }
        names[n] = true;
        var fPath = path + n + '/';
        blobs.push(Utilities.newBlob([]).setName(fPath));
        blobs = blobs.concat(getBlobs(folder, fPath));
    }
    return blobs;
}
```

Appendice C

conversions.gs

```
function convert(documentId) {
    var fl = DriveApp.getFileById(documentId);
    var mime = fl.getMimeType();
    switch (mime) {

        case MimeType.GOOGLE_DOCS:
            var blob = getBlob(documentId, MimeType.
                MICROSOFT_WORD)
                .setName(fl.getName() + '.docx');
            break;

        case MimeType.GOOGLE_SHEETS:
            var blob = getBlob(documentId, MimeType.
                MICROSOFT_EXCEL)
                .setName(fl.getName() + '.xlsx');
            break;

        case MimeType.GOOGLE_SLIDES:
            var blob = getBlob(documentId, MimeType.
                MICROSOFT_POWERPOINT)
                .setName(fl.getName() + '.pptx');
            break;

        case MimeType.GOOGLE_DRAWINGS:
```

```
            var blob = getBlob(documentId, MimeType.JPEG)
                .setName(fl.getName() + '.jpeg');
            break;

        case MimeType.GOOGLE_FORMS:
            var blob = UrlFetchApp.fetch(DriveApp.getFileById(
                documentId).getUrl())
                .getAs(MimeType.HTML).setName(fl.getName() + '.
                    html');
            break;

        case 'application/vnd.google-apps.map':
            var blob = UrlFetchApp.fetch(DriveApp.getFileById(
                documentId).getUrl())
                .getAs(MimeType.HTML).setName(fl.getName() + '.
                    html');
            break;

        case MimeType.GOOGLE_SITES:
            var blob = UrlFetchApp.fetch(DriveApp.getFileById(
                documentId).getUrl())
                .getAs(MimeType.HTML).setName(fl.getName() + '.
                    html');
            break;

        case 'application/vnd.google-apps.script':
            var blob = UrlFetchApp.fetch(DriveApp.getFileById(
                documentId).getUrl())
                .getAs(MimeType.PLAIN_TEXT).setName(fl.getName()
                    + '.json');
            break;

        default:
            var blob = DriveApp.getFileById(documentId).getBlob
                ();
            break;
    }
```

```
    return blob;
}

function getBlob(documentId, type) {
    var file = Drive.Files.get(documentId);
    var url = file.exportLinks[type];
    var oauthToken = ScriptApp.getOAuthToken();
    var response = UrlFetchApp.fetch(url, {
        headers: {
            'Authorization': 'Bearer ' + oauthToken
        }
    });
    return response.getBlob();
}
```

Appendice D

Index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
        scale=1, shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="">


    <title>DriveApp</title>
    <link rel="canonical" href="https://getbootstrap.com/docs
        /4.0/examples/dashboard/">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/
        ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.
        com/bootstrap/4.2.1/css/bootstrap.min.css" integrity="
        sha384-
        GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKMp0DgiMDm4iYMj70gZWKYbI706tWS
        " crossorigin="anonymous">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery
        /3.3.1/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.
```

```
        js/1.14.3/umd/popper.min.js" integrity="sha384-
        ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/
        l8WvCWPIPm49" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap
        /4.1.3/js/bootstrap.min.js" integrity="sha384-
        ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/
        JmZQ5stwEULTy" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf
        /1.5.3/jspdf.debug.js"></script>

    <!-- Custom styles for this template -->
    <?!=include('Stylesheet'); ?>
</head>

<body>
    <nav class="navbar navbar-expand-sm bg-dark navbar-dark
        fixed-top" id="main-menu">
        <button class="navbar-toggler navbar-toggler-right
            hidden-lg-up" type="button" data-toggle="collapse"
            data-target="#navbarsExampleDefault" aria-controls="
            navbarsExampleDefault" aria-expanded="false" aria-
            label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
        <a class="navbar-brand" href="#">
            GoogleDrivePersonalBackup</a>

        <div class="collapse navbar-collapse" id="
            navbarsExampleDefault">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item">
                    <a class="nav-link active" href="#" onClick
                        ="showFiles()" id="home">I miei file <
                        span class="sr-only">(current)</span></a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#" onClick="
                        showDownloads()" id="myD">I miei download
```

```
                    </a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" id="profile" href="#"
                        onClick="showContacts()" id="myC">Profile
                        </a>
                </li>
            </ul>
            <form class="form-inline mt-2 mt-md-0">
                <input class="form-control mr-sm-2" type="text"
                    id="search-bar" placeholder="Search">
                <div class="btn" id="search"><i class="fa fa-
                    search"></i></div>
            </form>
        </div>
    </nav>

    <div class="container-fluid">
        <div class="row">
            <nav class="col-sm-3 col-md-2 hidden-xs-down bg-
                faded sidebar">
                <ul class="nav nav-pills flex-column">
                    <li class="nav-item">
                        <a class="nav-link" href="#" onClick="
                            selectAll()"><i class="fa fa-pencil
                            menu-icon"></i> Seleziona tutto <span
                             class="sr-only">(current)</span></a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#" onClick="
                            autoDownload()"><i class="fa fa-
                            database menu-icon"></i> download
                            automatico</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#" onClick="
                            downloadContacts()"><i class="fa fa-
                            address-book menu-icon"></i> scarica
```

```
                contatti</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" id="folder" href="#"
                    onClick="downloadFolder()"> </a>
            </li>
        </ul>
        <div class="input-group input-group-sm mb-3
            download-click">
            <div class="input-group-prepend">
                <span class="input-group-text" id="
                    inputGroup-sizing-sm"><i class="fa fa
                    -folder"></i></span>
            </div>
            <input type="text" class="form-control" id="
                folder-name" name="folder-name" value="
                Download" aria-label="Small" aria-
                describedby="inputGroup-sizing-sm">
        </div>
        <div class="container-fluid" id="current-folder
            ">
        </div>
        <div class="nav nav-pills flex-column download-
            click">
            <div class="text-right" id="file-counter"></
                div>
            <div class="text-right" id="file-size"></div
                >
            <button class='btn tn-default' style="border
                : solid black 1px;" onClick='dwFiles()'>
                download</button>
        </div>
    </nav>

    <main class="col-sm-9 offset-sm-3 col-md-10 offset-
        md-2 pt-3">
        <h1 id="title"> <span id="Il mio Drive" onClick
            ="openFolder(id)">Il mio Drive</span> </h1>
```

```
        <div class="table-responsive" id="file-list">
            <table class="table table-hover" id="files
                ">
            <thead>
            <tr>
            <th> </th>
            <th>Nome</th>
            <th>Dimensione</th>
            <th>Proprietario</th>
            <th>Data creazione</th>
            </tr>
            </thead>
            <tbody id="files-table-body">
            </tbody>
            </table>
        </div>
        <section class="row text-center placeholders" id
            ="my-downloads">
        </section>
        <section class="row placeholders" id="my-
            contacts">
        <div class="table-responsive" id="file-list">
        <table class="table table-hover" id="files">
            <thead>
            <tr>
            <th> </th>
            <th>Mail</th>
            <th>Nome</th>
            <th>NickName</th>
            <th>Telefono</th>
            </tr>
            </thead>
            <tbody id="contacts">
            </tbody>
            </table>
        </div>
        </section>
    </main>
```

```
            <nav class="col-sm-3 col-md-2 hidden-xs-down bg-
                faded sidebar-right" id="zip-info">
            </nav>
        </div>
    </div>
    <div id="load-box text-center">
        <div class="container-fluid text-center loader">
            <i class="fa fa-circle-o-notch fa-spin"></i>
            <h2> caricamento drive in corso</h2>
            <br>
            <h3> il caricamento puo durare qualche minuto </h3>
        </div>
    </div>
    <div id="snackbar">Some text some message..</div>

    <!-- Modal -->
<div class="modal fade" id="myModal" role="dialog">
    <div class="modal-dialog">

      <!-- Modal content-->
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close" data-dismiss="
              modal">&times;</button>
          <h4 class="modal-title">Non Ã¨ stato possibile
              scaricare i seguenti file </h4>
        </div>
        <div class="modal-body">
          <p>Errore dovuto alla loro dimensione maggiore di 50
              Mb.</p>
          <p class="container container-fluid" id="errorFiles">
          </p>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-default" data-
              dismiss="modal">Close</button>
        </div>
```

```
        </div>

      </div>
    </div>

      <?!= include('scripts'); ?>
</body>

</html>
```

Appendice E

stylesheet.css

```
        /*
* Base structure
*/
    /* Move down content because we have a fixed navbar that is
       50px tall */

    body {
        padding-top: 50px;
    }
    /*
* Typography
*/

    h1 {
        margin-bottom: 20px;
        padding-bottom: 9px;
        border-bottom: 1px solid #eee;
    }
    /*
* Sidebar
*/

    .sidebar {
        position: fixed;
```

```
    top: 51px;
    bottom: 0;
    left: 0;
    z-index: 1000;
    padding: 20px;
    overflow-x: hidden;
    overflow-y: auto;
    /* Scrollable contents if viewport is shorter than
       content. */
    border-right: 1px solid #eee;
}
/* Sidebar navigation */

.sidebar {
    padding-left: 0;
    padding-right: 0;
}

.sidebar .nav {
    margin-bottom: 20px;
}

.sidebar .nav-item {
    width: 100%;
}

.sidebar .nav-item+.nav-item {
    margin-left: 0;
}

.sidebar .nav-link {
    border-radius: 0;
}

.bg-faded {
    background-color: #f7f7f7;
}
```

```
    .sidebar-right {
        position: fixed;
        top: 51px;
        bottom: 0;
        right: 0;
        z-index: 1000;
        padding: 20px;
        overflow-x: hidden;
        overflow-y: auto;
        /* Scrollable contents if viewport is shorter than
           content. */
        border-right: 1px solid #eee;
    }
    /*
* Dashboard
*/
    /* Placeholders */

    .placeholders {
        padding-bottom: 3rem;
    }

    .placeholder img {
        padding-top: 1.5rem;
        padding-bottom: 1.5rem;
    }
    /*other*/

    #current-folder,
    #zip-items {
        max-height: 35%;
        overflow: scroll;
    }

    #zip-item {
        cursor: default;
    }
```

```
#zip-item:hover {
    opacity: 1;
}

 ::-webkit-scrollbar {
    display: none;
}

.download-click {
    margin: 10px;
}

#folder-name {
    margin-right: 10px;
}

.table {
    text-align: left!Important;
}

.navItem: hover {
    color: white;
}

.bg-dark {
    background-color: blue!important;
    color: white!important
}

.fa-spin,
.fa-check,
.fa-warning,
.fa-download {
    font-size: 100px;
    cursor: pointer;
}

.fa-window-close,
```

```
#title span {
    cursor: pointer;
}

#title span:hover,
#title span:active {
    color: blue;
}

.fa-window-close:hover {
    color: blue;
}

.fa-spin,
.menu-icon {
    color: blue;
}

.fa-download {
    color: grey;
}

.fa-check {
    color: green;
}

.selected {
    background-color: #EEEEEE;
}

#info-icon {
    cursor: default;
}

.file-item:hover {
    cursor: pointer;
    opacity: 0.8;
}
```

```
.fa-warning {
    color: red;
}

#search {
    color: blue;
    background-color: #f7f7f7;
}
/*animations */

.animate-bottom {
    position: relative;
    -webkit-animation-name: animatebottom;
    -webkit-animation-duration: 1s;
    animation-name: animatebottom;
    animation-duration: 1s
}

@-webkit-keyframes animatebottom {
    from {
        bottom: -100px;
        opacity: 0
    }
    to {
        bottom: 0px;
        opacity: 1
    }
}

@keyframes animatebottom {
    from {
        bottom: -100px;
        opacity: 0
    }
    to {
        bottom: 0;
        opacity: 1
```

```
    }
}

//snackbar
#snackbar {
    visibility: hidden;
    min-width: 250px;
    margin-left: -125px;
    background-color: #333;
    color: #fff;
    text-align: center;
    border-radius: 2px;
    padding: 16px;
    position: fixed;
    z-index: 1;
    left: 50%;
    bottom: 30px;
    font-size: 17px;
}

#snackbar.show {
    visibility: visible;
    -webkit-animation: fadein 0.5s, fadeout 0.5s 2.5s;
    animation: fadein 0.5s, fadeout 0.5s 2.5s;
}

@-webkit-keyframes fadein {
    from {
        bottom: 0;
        opacity: 0;
    }
    to {
        bottom: 30px;
        opacity: 1;
    }
}

@keyframes fadein {
```

```
    from {
        bottom: 0;
        opacity: 0;
    }
    to {
        bottom: 30px;
        opacity: 1;
    }
}

@-webkit-keyframes fadeout {
    from {
        bottom: 30px;
        opacity: 1;
    }
    to {
        bottom: 0;
        opacity: 0;
    }
}

@keyframes fadeout {
    from {
        bottom: 30px;
        opacity: 1;
    }
    to {
        bottom: 0;
        opacity: 0;
    }
}
/* Small devices (landscape phones, 544px and up) */

@media (min-width: 544px) {
    .class-name {
        font-size: 16px;
    }
}
```

```
/* Medium devices (tablets, 768px and up) */

@media (min-width: 768px) {
    .class-name {
        font-size: 30px;
    }
}
/* Large devices (desktops, 992px and up) */

@media (min-width: 992px) {
    .class-name {
        font-size: 40px;
    }
    /* Extra large devices (large desktops, 1200px and up)
       */
    @media (min-width: 1200px) {
        .class-name {
            font-size: 48px;
        }
    }
```

Appendice F

scripts.js

```
var zip = []; //array di file che apparterranno ad un pacchetto
    zip
var files = []; //i file del drive dell'utente
var folders = [];
var folderSize = 0; //dimensione della cartella da comprimere
var selectedAllItems = false;
var currenFolder = "root";
var downloads = []; //array dei download effettuati nella
    sessione
var contacts = []; //array con i contatti dell'utente
var fileReady = []; //array di pacchetti zip pronti per essere
    scaricati
var idDownload = 0; //id di un download coincide con la
    posizione dell'elemento nell'array downloads
var activeList; //indica la lista che sta visualizzando l'utente
    , o i file, o i download o i contatti
var queue = []; //array per gestire i download in coda
var toGo = { //gestisce la concorrenza, quando un download
    finisce ne avvia un altro
    aInternal: true,
    aListener: function(val) {},
    set a(val) {
        this.aInternal = val;
        this.aListener(val);
```

```
    },
    get a() {
        return this.aInternal;
    },
    registerListener: function(listener) {
        this.aListener = listener;
    }
}

//al cambio della variabile a di toGo si avvia lo script, avvia
   un download in coda, in caso di download fallito segnala il
   download fallito e continua
toGo.registerListener(function(val) {
    if (toGo.a == true && queue.length != 0) {
        var obj = queue.shift();
        downloadFrom(obj.id);
        setTimeout(function() {
            if (obj.status == "download in corso") {
                obj.status = "download non riuscito";
                updateDownoads();
                toGo.a = true;
            }
        }, 650000);
    }
});



function findItemById(arr, id) {
    for (item of arr) {
        if (id == item.id) return arr.indexOf(item);
    }
    return -1;
}


//al caricamento del documento HTML faccio una richiesta a
   google script per avere i file del drive e contatti dell'
```

```
    utente
$(function() {
    $(".download-click").hide();
    $("#my-downloads").hide();
    $("#my-contacts").hide();
    $("#zip-info").hide();
    activeList = "files";
    var runner = google.script.run.withFailureHandler(onFailure)
        ;
    runner.withSuccessHandler(onSuccess).getFiles();
});

//ricerca dinamica da tastiera
$(function() {
    var text;
    var searchedList = [];
    $("#search-bar").keyup(function() {
        text = $("#search-bar").val();
        if (activeList == "files") $("#files-table-body").empty
            ();
        else if (activeList == "contacts") $("#contacts").empty
            ();
        else $("#my-downloads").empty();

        if (text) {
            //caso file
            if (activeList == "files") {
                for (item of files) {
                    if (item.name.includes(text)) searchedList.
                        push(item);
                }
                for (item of folders) {
                    if (item.name.includes(text)) searchedList.
                        push(item);
                }
                populateFilesAll(searchedList);
                populateFoldersAll(searchedList);
            }
```

```
            //caso contacts
            else if (activeList == "contacts") {
                for (item of contacts) {
                    if (item.mail.includes(text)) searchedList.
                        push(item);
                }
                populateContacts(searchedList);
            }
            //caso downloads
            else {
                for (item of downloads) {
                    if (item.name.includes(text)) searchedList.
                        push(item);
                }
                populateDownload(searchedList);
            }
        } else {
            if (activeList == "files") {
                populateFiles(files, "Il mio Drive");
                populateFolders(folders, "Il mio Drive");
            } else if (activeList == "contacts")
                populateContacts(contacts);
            else populateDownload(downloads);
        }

        searchedList = [];
    });
});

//recupero file del drive e contatti con successo
//data Ã¨ un array di oggetti con id, nome e dimensione del file
function onSuccess(data) {
    var data = JSON.parse(data);
    console.log(data);
    for (item of data.array) {
        if (item.size < 0) data.array.splice(indexOf(item), 1);
    }
    $(".loader").hide();
```

```
    $("#profile").html(data.user);
    files = data.array;
    contacts = data.contacts;
    folders = data.folders;
    if (data.array.length != 0) {
        populateFiles(files, "Il mio Drive");
    }
    if (contacts.length != 0) {
        populateContacts(contacts);
    }
    if (folders.length != 0) {
        populateFolders(folders, "Il mio Drive");
    }
}

function populateContacts(contacts) {
    for (item of contacts) {
        var listItem = $("<tr></tr>").addClass("animate-bottom
            file-item").attr({ id: item.id, onClick: "selected(id
            )" });
        var ic = $("<td></td>").addClass("text-left");
        var contactLogo = $("<i></i>").addClass("fa fa-user-
            circle-o").css("color", "blue");
        $(ic).append(contactLogo);
        $(listItem).append(ic);
        if (item.mail != null) $(listItem).append("<td>" + item.
            mail + "</td>");
        else $(listItem).append("<td> - </td>");
        if (item.name) $(listItem).append("<td>" + item.name +
            "</td>");
        else $(listItem).append("<td> - </td>");
        if (item.nickName) $(listItem).append("<td>" + item.
            nickName + "</td>");
        else $(listItem).append("<td> - </td>");
        if (item.cel) $(listItem).append("<td>" + item.cel + "</
            td>");
        else $(listItem).append("<td> - </td>");
        $("#contacts").append(listItem);
```

```
    }
}

function populateFiles(files, folder) {
    for (item of files) {
        if (item.parent.length == 0 || item.parent[0] == folder)
            {
            var listItem = $("<tr></tr>").addClass("animate-
                bottom file-item").attr({ id: item.id, onClick: "
                selected(id)" });
            for (element in zip) {
                if (zip[element].name == item.name) $(listItem).
                    addClass("selected");
            }
            var ic = $("<td></td>");
            $(ic).append(getTypeIcon(item.mimeType));
            $(listItem).append(ic);
            $(listItem).append("<td>" + item.name + "</td>");
            if (getMb(item.size / 1024 / 1024, 2) != 0) $(
                listItem).append("<td>" + getMb(item.size / 1024
                / 1024, 2) + " Mb</td>");
            else $(listItem).append("<td> - </td>");
            $(listItem).append("<td>" + item.owner + "</td>");
            $(listItem).append("<td>" + dateParser(item.
                dateCreated) + "</td>");
            $("#files-table-body").append(listItem);
        }
    }
}

function populateFilesAll(files) {
    for (item of files) {
        var listItem = $("<tr></tr>").addClass("animate-bottom
            file-item").attr({ id: item.id, onClick: "selected(id
            )" });
        for (element in zip) {
            if (zip[element].name == item.name) $(listItem).
                addClass("selected");
```

```
        }
        var ic = $("<td></td>");
        $(ic).append(getTypeIcon(item.mimeType || "folder"));
        $(listItem).append(ic);
        $(listItem).append("<td>" + item.name + "</td>");
        if (getMb(item.size / 1024 / 1024, 2) != 0) $(listItem).
            append("<td>" + getMb(item.size / 1024 / 1024, 2) + "
             Mb</td>");
        else $(listItem).append("<td> - </td>");
        $(listItem).append("<td>" + item.owner + "</td>");
        $(listItem).append("<td>" + dateParser(item.dateCreated)
             + "</td>");
        $("#files-table-body").append(listItem);
    }
}

function populateFoldersAll(fold) {
    for (item of fold) {
        var listItem = $("<tr></tr>").addClass("animate-bottom
            file-item").attr({ id: item.name, onClick: "
            openFolder(id)" });
        var ic = $("<td></td>");
        $(ic).append(getTypeIcon("folder"));
        $(listItem).append(ic);
        $(listItem).append("<td>" + item.name + "</td>");
        $(listItem).append("<td> - </td>");
        $(listItem).append("<td>" + item.owner + "</td>");
        $(listItem).append("<td>" + dateParser(item.dateCreated)
             + "</td>");
        $("#files-table-body").append(listItem);
    }
}

function downloadFolder() {
    var zipItem = {
        id: idDownload,
        name: currentFolder,
        files: "null",
```

```
            status: "download in corso"
        }
        toGo.a = false;
        idDownload++;
        downloads.push(zipItem);
        var runner = google.script.run;
        runner.withFailureHandler(failure);
        runner.withSuccessHandler(success).downloadFolder(zipItem);
        insertDownload(zipItem);
        showDownloads();
        showSnackBar("download di " + currentFolder + " in corso");
        $("#current-folder").empty();
        zip = [];
        $(".file-item").removeClass("selected");
        $("#file-counter").empty();
        $("#file-size").empty();
    }



    function populateFolders(fold, folder) {
        for (item of fold) {
            console.log(item.parents[0]);
            if (item.parents.length == 0 || item.parents[0] ==
                folder) {
                var listItem = $("<tr></tr>").addClass("animate-
                    bottom file-item").attr({ id: item.name, onClick:
                     "openFolder(id)" });
                var ic = $("<td></td>");
                $(ic).append(getTypeIcon("folder"));
                $(listItem).append(ic);
                $(listItem).append("<td>" + item.name + "</td>");
                $(listItem).append("<td> - </td>");
                $(listItem).append("<td>" + item.owner + "</td>");
                $(listItem).append("<td>" + dateParser(item.
                    dateCreated) + "</td>");
                $("#files-table-body").append(listItem);
            }
```

```
    }
}

function populateDownload(download) {
    for (item of download) {
        insertDownload(item);
    }
}

function openFolder(name) {
    console.log(name);
    var fil = [];
    var fol = [];
    currentFolder = name;
    $("#files-table-body").empty();
    $("#folder").html("<i class='fa fa-folder menu-icon'></i>
       scarica " + currentFolder);
    for (item of files) {
        if (item.parent[0] == name) fil.push(item);
    }
    for (item of folders) {
        if (item.parents[0] == name) fol.push(item);
    }
    var title = $("#title").html().split(" / ");
    title.push("<span id='" + name + "' onClick='openFolder(id)
       '>" + name + "</span>");
    var titleStr = " ";
    for (item of title) {
        if (!item.includes(name)) titleStr += item + " / ";
        else {
            titleStr += item;
            break;
        }
    }
    $("#title").html(titleStr);
    if (name == "Il mio Drive") {
        populateFiles(fil, "Il mio Drive");
        populateFolders(fol, "Il mio Drive");
```

```
    } else {
        populateFiles(fil, name);
        populateFolders(fol, name);
    }
}

function dateParser(date) {
    const monthNames = ["Gen", "Feb", "Mar", "Apr", "Mag", "Giu
        ",
        "Lug", "Ago", "Set", "Ott", "Nov", "Dic"
    ];
    var d = new Date(date);
    return d.getDate() + " " + monthNames[d.getMonth()] + " " +
        d.getFullYear();
}

//in base al tipo del file restituisce un icona
function getTypeIcon(mimeType) {
    var icon = $("<i></i>");
    if (mimeType == "folder") $(icon).addClass("fa fa-folder").
        css("color", "black");
    else if (mimeType == "application/zip" || mimeType == "
        application/rar" ||
        mimeType == "application/octet-stream") icon.addClass("
            fa fa-file-archive-o").css("color", "purple");
    else if (mimeType.split("/")[0] == "image" || mimeType == "
        application/vnd.google-apps.drawing" ||
        mimeType == "application/vnd.google-apps.photo") $(icon)
            .addClass("fa fa-file-image-o").css("color", "red");
    else if (mimeType == "text/plain") $(icon).addClass("fa fa-
        file-text-o").css("color", "black");
    else if (mimeType.split("/")[1] == "pdf") $(icon).addClass("
        fa fa-file-pdf-o").css("color", "red");
    else if (mimeType.split("/")[0] == "audio") $(icon).addClass
        ("fa fa-file-audio-o").css("color", "blue");
    else if (mimeType.split("/")[0] == "video") $(icon).addClass
        ("fa fa-file-video-o").css("color", "blue");
    else if (mimeType.split("/")[1].includes("script")) $(icon).
```

```
        addClass("fa fa-file-code-o").css("color", "grey");
    else if (mimeType == "application/vnd.google-apps.document"
        ||
        mimeType == "application/vnd.openxmlformats-
            officedocument.wordprocessingml.document" ||
        mimeType == "application/msword") $(icon).addClass("fa
            fa-file-word-o").css("color", "blue");
    else if (mimeType == "application/vnd.google-apps.
        spreadsheet" ||
        mimeType == "application/vnd.openxmlformats-
            officedocument.spreadsheetml.sheet" ||
        mimeType == "application/vnd.ms-excel") $(icon).addClass
            ("fa fa-file-excel-o").css("color", "green");
    else if (mimeType == "application/vnd.google-apps.
        presentation" ||
        mimeType == "application/vnd.openxmlformats-
            officedocument.presentationml.presentation" ||
        mimeType == "application/vnd.ms-powerpoint") $(icon).
            addClass("fa fa-file-powerpoint-o").css("color", "
            orange");
    else if (mimeType == "application/vnd.google-apps.map") $(
        icon).addClass("fa fa-file-text-o").css("color", "red");
    else if (mimeType == "application/vnd.microsoft.portable-
        executable") $icon.addClass("fa fa-file").css("color", "
        grey");
    else if (mimeType == "application/vnd.google-apps.form" ||
        mimeType == "application/vnd.google-apps.site") $(icon).
        addClass("fa fa-file-text-o").css("color", "purple");
    else icon.addClass("fa fa-file-o").css("color", "blue");

    return icon;
}

//fallimento di una richiesta a google
function onFailure(data) {
    alert(data);
}
```

```
//download contatti
function downloadContacts() {
    var doc = new jsPDF();
    var elementHandler = {
        '#ignorePDF': function(element, renderer) {
            return true;
        }
    };
    var source = document.getElementById("contacts");
    doc.fromHTML(
        source,
        15,
        15, {
            'width': 180,
            'elementHandlers': elementHandler
        });

    doc.save("contacts.pdf");
}

//selezione del file cliccato
function selected(id) {
    if (!$('#' + id).hasClass("selected")) {
        var itm;
        for (item of files) {
            if (item.id === id) {
                itm = item;
                break;
            }
        }
        if (itm.size < 52428800) {
            $('#' + id).addClass("selected");
            zip.push(itm);
            folderSize += item.size;
            $(".download-click").show();
        } else {};
    } else {
        $('#' + id).removeClass("selected");
```

```
        var index = zip.indexOf(id)
        zip.splice(index, 1);
        if (zip.length == 0) {
            $(".download-click").hide();
        }
        folderSize -= item.size;
    }
    $("#current-folder").empty();
    for (item of zip) {
        $("#current-folder").append("<li class='list-group-item
            '>" + item.name + "</li>");
    }
    $("#file-counter").text("file totali: " + zip.length);
    $("#file-size").text("dimensione: " + getMb(folderSize /
        1024 / 1024, 2) + "Mb");
}

function getMb(value, digit) {
    digit++;
    value = value || 0;
    value = value.toString();
    if (value.indexOf(".") > 0) {
        value = value.substring(0, value.indexOf(".") + digit);
    }
    value = parseFloat(value);
    return value;
}

function dwFiles() {
    var conf = confirm('vuoi scaricare i file selezionati?');
    if (conf == true) {
        download();
    } else {}
}

function download() {
    if (toGo.a == true) {
        if (zip.length != 0) {
```

```
            var zipItem = {
                id: idDownload,
                name: $("#folder-name").val(),
                files: zip,
                status: "download in corso"
            }
            toGo.a = false;
            idDownload++;
            downloads.push(zipItem);
            var runner = google.script.run;
            runner.withFailureHandler(failure);
            runner.withSuccessHandler(success).downloadFiles(
                zipItem);
            insertDownload(zipItem);
            showDownloads();
            showSnackBar("download di " + zipItem.name + " in
                corso");
            $("#current-folder").empty();
            zip = [];
            $(".file-item").removeClass("selected");
            $("#file-counter").empty();
            $("#file-size").empty();
        } else {
            alert("non hai selezionato nessun file");
        }
    } else {
        var zipItem = {
            id: idDownload,
            name: $("#folder-name").val(),
            files: zip,
            status: "in coda"
        }
        downloads.push(zipItem);
        showDownloads();
        showSnackBar("download di " + zipItem.name + " in coda")
            ;
        idDownload++;
        queue.push(zipItem);
```

```
        updateDownloads();
    }
    folderSize = 0;
}


//downalod terminato con successo
function success(data) {
    var file;
    toGo.a = true;
    data = JSON.parse(data);
    downloads[data.folder].status = data.msg;
    downloads[data.folder].notAuth = data.notAuth;
    try {
        var blob = b64toBlob(data.item, 'application/zip');
        file = {
            base: data.item,
            blob: blob
        }
    } catch (err) {

    }

    fileReady[data.folder] = file;

    updateDownloads();
    showSnackBar("download di " + downloads[data.folder].name +
        " completato");
}

//errore nel download
function failure(data) {
    data = JSON.parse(data);
    downloads[data.folder].status = "download non riuscito";
    updateDownloads();
    $(".file-item").removeClass("selected");
}
```

```
//coversione da una stranda in base64 al relativo blob: b64Data
   la stringa da convertire, contentType l'estensione del file
function b64toBlob(b64Data, contentType, sliceSize) {
    contentType = contentType || '';
    sliceSize = sliceSize || 512;

    var byteCharacters = atob(b64Data);
    var byteArrays = [];

    for (var offset = 0; offset < byteCharacters.length; offset
       += sliceSize) {
        var slice = byteCharacters.slice(offset, offset +
            sliceSize);

        var byteNumbers = new Array(slice.length);
        for (var i = 0; i < slice.length; i++) {
            byteNumbers[i] = slice.charCodeAt(i);
        }

        var byteArray = new Uint8Array(byteNumbers);

        byteArrays.push(byteArray);
    }

    var blob = new Blob(byteArrays, { type: contentType });
    return blob;
}

function directDownload(index) {
    var blob = fileReady[index].blob;
    var blobUrl = URL.createObjectURL(blob);
    window.location = blobUrl;
}

function sendMail(index) {
    var base = fileReady[index].base;
    base = {
        name: $("#folder-name").val(),
```

```
        data: base
    }
    var runner = google.script.run.withFailureHandler(
        showSnackBar("c'Ã¨ stato un problema con l'invio della
        mail"));
    runner.withSuccessHandler(showSnackBar("mail inviata")).
        sendGMail(JSON.stringify(base));
    showSnackBar("invio mail in corso");
}

//mostra finestra dei downloads effettuati
function showDownloads() {
    activeList = "downloads";
    $('#my-downloads').show();
    $('#file-list').hide();
    $('#Myd').addClass("active");
    $('#Myc').removeClass("active");
    $('#home').removeClass("active");
    $('#load-box').hide();
    $('#my-contacts').hide();
    $('#title').text("I miei Downlaod");
}

function showContacts() {
    activeList = "contacts";
    $('#my-downloads').hide();
    $('#file-list').hide();
    $('#Myd').removeClass("active");
    $('#Myc').addClass("active");
    $('#home').removeClass("active");
    $('#load-box').hide();
    $('#my-contacts').show();
    $('#title').text("I miei Contatti");
}

//mostra lista dei file
function showFiles() {
    activeList = "files";
```

```
        $('#my-downloads').hide();
        $('#Myd').removeClass("active");
        $('#Myc').removeClass("active");
        $('#home').addClass("active");
        $('#file-list').show();
        $('#my-contacts').hide();
        $('#title').html("<span id='Il mio Drive' onClick='
            openFolder(id)'>Il mio Drive</span>");
    }

    //seleziona tutti i files nel drive
    function selectAll() {
        folderSize = 0;
        if (!selectedAllItems) {
            $(".file-item").addClass("selected");
            selectedAllItems = true;
            zip = files;
            $(".download-click").show();
            $("#current-folder").empty();
            for (item of zip) {
                $("#current-folder").append("<li class='list-group-
                    item'>" + item.name + "</li>");
                folderSize += item.size;
            }
            $("#file-counter").text("file totali: " + zip.length);
            $("#file-size").text("dimensione: " + getMb(folderSize /
                1024 / 1024, 2) + "Mb");
        } else {
            $(".file-item").removeClass("selected");
            selectedAllItems = false;
            zip = [];
            $(".download-click").hide();
            $("#current-folder").empty();
        }
    }

    //aggiornamento dello stato dei download
    function updateDownloads() {
```

```
    $("#my-downloads").empty();
    for (item of downloads) {
        insertDownload(item);
    }
}

//inserisce icona di download con lo stato del download e le
   operazioni disponibili
function insertDownload(zipItem) {
    var box = $("<div></div>").addClass("col-6 col-sm-3
       placeholder").attr("id", zipItem.id);
    var icon = $("<i></i>").attr("onClick", "(function(){
       showInfo(" + zipItem.id + ");})()");
    var options = [];
    var btnGroups = $("<div></div>").addClass("btn-group-
       vertical");
    switch (zipItem.status) {
        case "download in corso":
            $(icon).addClass("fa fa-circle-o-notch fa-spin");
            break;
        case "pronto per il download":
            $(icon).addClass("fa fa-download");
            options = [
                ["inizia a scaricare", "downloadFrom(" + zipItem
                   .id + ")"]
            ];
            break
        case "download completato":
            $(icon).addClass("fa fa-check");
            options = [
                ["download diretto", "directDownload(" + zipItem
                   .id + ")"],
                ["manda zip per e-mail", "sendMail(" + zipItem.
                   id + ")"]
            ];
            break;
        case "download non riuscito":
            $(icon).addClass("fa fa-warning");
```

```
            options = [
                ["riprova", "retry(" + zipItem.id + ")"],
                ["annulla", "remove(" + zipItem.id + ")"]
            ];
            break;
        case "in coda":
            $(icon).addClass("fa fa-spinner fa-spin");
            options = [
                ["annulla", "removeFromQueue(" + zipItem.id + ")
                    "]
            ];
            break;
    }
    $(box).append(icon);
    $(box).append("<h4>" + zipItem.name + "</h4>");
    $(box).append("<div class='text-muted'>" + zipItem.status +
        "</div>");
    for (item of options) {
        $(btnGroups).append("<button type='button' class='btn
            btn-outline-primary' onClick=(function(){" + item[1]
            + ";})()>" + item[0] + "</button>");
    }
    $(box).append(btnGroups);
    $("#my-downloads").append(box);
}

function getIcon(id) {
    var zipItem = downloads[id];
    var icon = $("<i></i>").attr("id", "info-icon");
    switch (zipItem.status) {
        case "download in corso":
            $(icon).addClass("fa fa-circle-o-notch fa-spin");
            break;
        case "pronto per il download":
            $(icon).addClass("fa fa-download");
            break
        case "download completato":
            $(icon).addClass("fa fa-check");
```

```
            break;
        case "download non riuscito":
            $(icon).addClass("fa fa-warning");
            break;
        case "in coda":
            $(icon).addClass("fa fa-spinner fa-spin");
            break;
    }

    return icon;
}

function remove(id) {
    downloads.splice(id, 1);
    updateDownloads();
}

function removeFromQueue(id) {
    downloads[id].status = "pronto per il download";
    queue.splice(queue.indexOf(downloads[id]), 1);
    updateDownloads();
}

//mostra informazioni su un pacchetto zip
function showInfo(id) {
    var obj = downloads[id];
    var icon = getIcon(id);
    var iconBox = $("<div></div>").addClass("text-center");
    var status = $("<p></p>").addClass("container-fluid text-
        center").html("<h5>" + obj.status + "</h4>");
    var name = $("<p></p>").addClass("container-fluid text-
        center").html("<h5>" + obj.name + "</h3>");
    var notAuthoredFiles = $("<ul></ul>").addClass("list-group")
        .attr("id", "non-authored-files");
    var fileList = $("<ul></ul>").addClass("list-group").attr("
        id", "zip-items");
    var zipLenght = obj.files.length;
    if (obj.stauts == "download completato" || obj.status == "
```

```
    download non riuscito") {
     var zipLenght = obj.files.length - obj.notAuth.length;
     for (var i = 0; i < obj.files.length; i++) {
         if (findItemById(obj.notAuth, obj.files[i].id) ==
             -1) {
              var listItem = $("<li></li>").addClass("list-
                  group-item animate-bottom file-item").attr("
                  id", "zip-item").text(obj.files[i].name);
              $(fileList).append(listItem);
         }
     }
     for (item of obj.notAuth) {
         var listItem = $("<li></li>").addClass("list-group-
             item animate-bottom file-item").attr("id", "not-
             authored-item").text(item.name);
         $(notAuthoredFiles).append(listItem);
     }
} else {
     for (var i = 0; i < obj.files.length; i++) {
         var listItem = $("<li></li>").addClass("list-group-
             item animate-bottom file-item").attr("id", "zip-
             item").text(obj.files[i].name);
         $(fileList).append(listItem);
     }
}

$(iconBox).append(icon);
$("#zip-info").empty();
$("#zip-info").append("<i class='fa fa-window-close' aria-
    hidden='true' onClick='closeInfo()'></i>");
$("#zip-info").append(iconBox);
$("#zip-info").append(name);
$("#zip-info").append(status);
$("#zip-info").append("<h6>contenuto pacchetto (" +
    zipLenght + " files)</h6><hr>");
$("#zip-info").append(fileList);
if (obj.notAuth) {
    (obj.notAuth.lenght != 0) ? $("#zip-info").append("<h6>
```

```
            non Ã¨ stato possibile scaricare i seguenti files</h6
            ><hr>"): $("#zip-info").append("<h6>tutti i files
            sono stati scaricati correttamente</h6><hr>");
        $("#zip-info").append(notAuthoredFiles);
    }
    $("#zip-info").show();

}

function closeInfo() {
    $("#zip-info").hide();
}


function retry(id) {
    downloadFrom(id);
}


//divide il drive in pacchetti, notifica l'utente del nummero di
    pacchetti che riceverÃ  e su conferma avvia il download
function autoDownload() {
    var downloadArr = []; //array con i pacchetti da spedire all
        'utente
    var sizep = 0; //contatore per la dimensione di ogni
        pacchetto
    var errorFiles = []; //tutti i file > 25Mb vengono inseriti
        qua

    //ordinamendo in ordine crescente di dimensione dei file (
        tecica greedy)
    files.sort(function(a, b) {
        return a.size - b.size
    });

    //creazione pacchetti
    for (item of files) {
        if (sizep + item.size < 26214400) {
```

```
            zip.push(item);
            sizep += item.size;
        } else {
            if (zip.length != 0) downloadArr.push(zip);
            zip = [];
            sizep = 0;
            if (item.size < 52428800) {
                zip.push(item);
                sizep = item.size;
                downloadArr.push(zip)
            } else {
                errorFiles.push(item);
            }
        }
    }

    var conf = confirm("verranno creati " + downloadArr.length +
        " pacchetti, continuare?");
    if (conf == true) {
        showDownloads();
        for (item of downloadArr) {
            var zipItem = {
                id: idDownload,
                name: $("#folder-name").val() + " " +
                    downloadArr.indexOf(item),
                files: item,
                status: "pronto per il download"
            }
            console.log(zipItem);
            idDownload++;
            insertDownload(zipItem);
            downloads.push(zipItem);
        }
        if (errorFiles.length != 0) {
            for (item of errorFiles) $("#errorFiles").append("<p
                >" + item.name + "</p>");
            $("#myModal").modal();
        }
```

```
    } else {}
}

function downloadFrom(id) {
    if (toGo.a == true) {
        toGo.a = false;
        var runner = google.script.run;
        runner.withFailureHandler(failure);
        runner.withSuccessHandler(success).downloadFiles(
            downloads[id]);
        showDownloads();
        showSnackBar("download di " + downloads[id].name + " in
            corso");
        $("#current-folder").empty();
        zip = [];
        $(".file-item").removeClass("selected");
        $("#file-counter").empty();
        $("#file-size").empty();
        downloads[id].status = "download in corso";
        updateDownloads();
    } else {
        downloads[id].status = "in coda";
        queue.push(downloads[id]);
        updateDownloads();
    }
}


//mostra una box sbnackbar con testo text
function showSnackBar(text) {
    var x = document.getElementById("snackbar");
    $(x).html(text);
    x.className = "show";
    setTimeout(function() { x.className = x.className.replace("
        show", ""); }, 3000);
}
```

Bibliografia

[1] GDPR: *https://gdpr-info.eu/*

[2] Amazon EC2: *http://aws.amazon.com/ec2/*

[3] B. Sotomayor, R. S. Montero, I.M. Llorente, I. Foster: *Virtual Infrastructure Management in Private and Hybrid Clouds, J. IEEE Internet Computing, vol. 13, no. 5, Sept.- Oct. 2009*

[4] OpenStack: *http://openstack.org/*

[5] Google Drive: *https://www.google.com/drive/*

[6] Dropbox: *https://www.dropbox.com/*

[7] One Drive: *https://onedrive.live.com/about/it-it/*

[8] *https://www.statista.com/statistics/499558/worldwide-personal-cloud-storage-users/*

[9] Box: *https://www.box.com/it-it/home*

[10] ICloud Drive: *https://www.apple.com/icloud/*

[11] Node js: *https://nodejs.org/en/*

[12] DropBox paper : *https://www.dropbox.com/paper*

[13] *https://developers.google.com/apps-script/guides/services/quotas#current_limitations*

[14] *https://www.npmjs.com/package/b64-to-blob*