Relazione Progetto
Piattaforme Digitali per la Gestione del Territorio

Joshua Micheletti
Matricola: 283057

- Descrizione del servizio implementato dall'API
- Descrizione dei metodi implementati nell'API
- Descrizione del client che sfrutta il servizio
- Descrizione Server Blob Storage Minio

Descrizione del Servizio Implementato dall'API

Il servizio implementa la gestione di utenti, server e file per il trasferimento di file tramite protocollo HTTP.

L'URL del servizio è: `https://projectpdgt.herokuapp.com`

Il programma utilizza librerie esterne per svolgere le sue funzionalità:

- **express**: libreria per la gestione di richieste HTTP e per generare risposte
- **js-sha256**: libreria per la codifica di stringe con algoritmo SHA256
- **cookie-parser**: libreria per la gestione di cookie nelle richieste e risposte HTTP
- **multer**: libreria per la gestione di file ricevuti come corpi di richieste HTTP
- **minio**: API per comunicare con un blob storage server dove salvare i file

Il server di blob storage minio contiene un bucket speciale "info" in cui sono presenti due file: "users.json" e "servers.json".
Questi file contengono la lista di utenti e server registrati al servizio, e sono sincronizzati con le strutture dati nell'API per tenere conto dei dati di utenti e server.

Il servizio di blob storage minio è hostato sul server "mathorgadaorc.ddns.net:9000".

Tramite richieste HTTP, è possibile creare ed eliminare utenti, verificare le proprie credenziali tramite login e ottenere cookie per eseguire un accesso più rapido.

Si possono generare anche nuovi server per contenere i propri file:
Un server consiste in un oggetto con un nome, una password, un proprietario e un bucket nel blob server minio associato.
L'API mette a disposizione metodi per la creazione di nuovi server, il cui proprietario sarà l'utente definito nella richiesta HTTP, rimozione di server e gestione di file all'interno di un server (processi di rimozione di un server, upload di file e rimozione di file è consentito solo al proprietario del server, mentre il download di file è disponibile a chiunque possa accedere al server).

All'interno dell'API, tutte le password vengono composte con un numero (salt), per garantire l'unanimità, ed ogni password viene codificata tramite algoritmo SHA256, così da rendere le password inacessibili dall'API.

Descrizione dei metodi implementati nell'API

L'API si divide in 3 principali funzionalità per rendere la gestione dei file più sicura e privata:

- Gestione utenti
- Gestione server
- Gestione file

Gestione Utenti

Login:

```
GET /login HTTP/1.1
Host: projectpdgt.herokuapp.com
Authorization: Basic ******************
```

Questo metodo serve per verificare la validità delle credenziali di un utente che vuole accedere al servizio.

Richieste GET verso /login rispondono con 200 OK se le credenziali contenute nell'header "Authorization", codificate in Base64, corrispondono a credenziali conosciute dentro il web service.
Nel caso di risposta 200 OK, il server invia anche un messaggio contenente il nome dell'utente collegato.

Se la richiesta non contiene credenziali o se contiene credenziali che non corrispondono a nessun utente registrato, la risposta sarà 403 FORBIDDEN.

Tutte i controlli di accessibilità vengono svolti dalla funzione `attemptAuth(req),` utilizzata in ogni metodo che necessita la verifica della validità dell'utente.

Login Cookie:

```
POST /loginCookie HTTP/1.1
Host: projectpdgt.herokuapp.com
Authorization: Basic ****************
```

OR

```
POST /loginCookie HTTP/1.1
Host: projectpdgt.herokuapp.com
Cookie: auth=****************************
```

Il metodo /loginCookie è un metodo per verificare la correttezza delle credenziali dell'utente collegato, tramite cookie o tramite credenziali codificate in Base64.

Nel caso in cui la richiesta contenga le credenziali nell'header "Authorization" e non un header "Cookie", il programma verifica che le credenziali corrispondano ad un utente registrato. Come nel caso di /login, se l'utente non è registrato, il programma risponde con 403 FORBIDDEN.
Nel caso invece in cui l'utente è contenuto nella lista degli utenti, e la password coincide con quella salvata nel server, viene creata una stringa aleatoria, basata sulla data attuale e codificata in sha256.
Questa stringa rappresenta il cookie che verrà salvato nella lista dei cookie, e passato all'utente nella risposta.
Il metodo risponde con 201 CREATED e fornisce all'utente il proprio nome utente e il cookie ad esso associato.

Se invece abbiamo già un cookie e lo forniamo come header "Cookie" nella richiesta HTTP, questo metodo userà la stringa associata al cookie per verificare l'identità dell'utente, invece che utilizzare la password.
Una volta verificata l'identità dell'utente, creerà un nuovo cookie per lo stesso utente, e lo fornisce nella risposta 201 CREATED.

Register:

```
POST /users HTTP/1.1
Host: projectpdgt.herokuapp.com
Authorization: Basic ******************
```

Una richiesta POST verso /users rappresenta il tentativo di registrare un nuovo utente.

Se la richiesta non contiene le informazioni necessarie (mancanza di header “Authorization”, mancanza di utente o password), il metodo risponderà con 400 BAD REQUEST.

Nel caso invece in cui il nome utente che si tenta di registrare sia già presente come utente registrato, la risposta sarà: 409 CONFLICT.

Se l’utente che si tenta di registrare è un nuovo utente e la richiesta contiene tutte le informazioni necessarie, il metodo tenta di aggiornare la lista di utenti e di salvarla in una copia remota nel server di blob storage sotto forma di file json.
Se il processo di aggiornamento del file contenente la lista di utenti (“users.json”) fallisce, la risposta sarà: 503 SERVICE UNAVAILABLE.
Nel caso in cui questo processo vada a buon fine, la risposta sarà: 201 CREATED

Una volta creato un nuovo utente, sarà possibile accedere al servizio con le credenziali registrate tramite i metodi /login e /loginCookie.

Delete:

```
DELETE /users HTTP/1.1
Host: projectpdgt.herokuapp.com
Authorization: Basic ******************
```

Inviando una richiesta DELETE verso /users, si tenta di eliminare l’utente definito nell’header “Authorization” dalla lista degli utenti registrati.

Come nel metodo precedente, mancanza di informazioni necessarie (dati dell’utente da eliminare), la risposta sarà 400 BAD REQUEST.

Se invece le informazioni sono corrette ma l’utente specificato non esiste nella lista degli utenti, la risposta sarà 404 FILE NOT FOUND.

Se l’utente specificato esiste ma la password fornita non coincide con quella registrata, la risposta sarà 403 FORBIDDEN.

Nel caso in cui tutte le informazioni necessarie siano presenti e corrette, il programma tenterà di aggiornare la lista degli utenti salvata nel file “users.json”, che terminerà in una risposta 503 SERVICE UNAVAILABLE nel caso in cui il procedimento di aggiornamento del file fallisca, oppure con 200 OK nel caso in cui il procedimento termini senza problemi.

Gestione Server

List servers / login server:

```
GET /servers HTTP/1.1
Host: projectpdgt.herokuapp.com
```

OR

```
GET /servers?serverName=________&serverPassword=________ HTTP/1.1
Host: projectpdgt.herokuapp.com
Authorization: Basic ***************
```

Questo metodo serve per ottenere un elenco dei server disponibili, oppure per effettuare il login in uno di essi.

Nel caso in cui non forniamo informazioni su quale server stiamo cercando di accedere, il programma ritorna 200 OK e una lista dei server registrati sotto forma di un array.

Se invece tentiamo di accedere ad un server specifico, vengono prima verificate le credenziali fornite nell'header "Authorization". Nel caso in cui manchino informazioni o nel caso in cui le informazioni fornite non corrispondano a nessun utente registrato, il metodo risponderà con 403 FORBIDDEN.

Se l'utente è autorizzato ma tenta di accedere ad un server non presente nella lista, il metodo risponde con 404 FILE NOT FOUND.

Se il server selezionato esiste ma la password fornita non corrisponde con quella registrata, la risposta sarà 403 FORBIDDEN.

Se invece tutte le informazioni sono corrette e si ha accesso al server, il programma risponde con 200 OK e fornisce il nome del server a cui si è eseguito l'accesso, concatenato con il nome utente del proprietario del server.

Create Server:

```
POST /servers?serverName=_______&serverPassword=________ HTTP/1.1
Host: projectpdgt.herokuapp.com
Authorization: Basic ****************
```

Una richiesta POST verso /servers genera un nuovo server se corretta.

La richiesta richiede informazioni come il nome del server e la password del nuovo server passati per query, e le credenziali dell'utente passate per header "Authorization".

La mancanza delle credenziali o la loro invalidità (credenziali non esistenti o errate) generano una risposta 403 FORBIDDEN.

La mancanza delle informazioni del server genera una risposta 400 BAD REQUEST.

Se il nome del nuovo server coincide con il nome di un server già registrato, oppure se il nome del server concide con "info", il metodo risponde con 409 CONFLICT.
"info" è un server speciale che contiene i file json che rappresentano la lista di server e la lista di utenti registrati.

Se l'utente è autorizzato e il nuovo server è valido, il programma tenta di creare un nuovo bucket nel server di storage, tenta di generare due file (1.txt e 2.txt, questo per evitare un bug nell'API del server di storage) e infine tenta di aggiornare il file "servers.json" con la nuova lista di server che include quello appena generato.

Ad ognuno di questi passaggi, in caso di errore, il programma ritorna 503 SERVICE UNAVAILABLE.

Nel caso in cui non ci siano errori, il programma ritorna 201 CREATED e il nome del server appena creato.
Il proprietario del server sarà l'utente registrato e fornito nella richiesta.

Ora il server sarà accessibile tramite il metodo GET /servers.

Delete Server:

```
DELETE /servers?serverName=________&serverPassword=________ HTTP/1.1
Host: projectpdgt.herokuapp.com
Authorization: Basic ********************
```

Questo metodo serve per eliminare un server registrato.
La cancellazione di un server eliminerà il contenuto di ogni file al suo interno, e può essere svolta solo dal proprio proprietario.

Il programma controlla che le informazioni di autenticazione fornite siano valide e corrette, nel caso in cui l'utente non sia un utente registrato o manchino informazioni necessarie all'autenticazione, la risposta sarà 403 FORBIDDEN.

Il metodo controlla che ci siano informazioni riguardo al server da eliminare, e che siano valide. Nel caso in cui manchino serverName o serverPassword, il programma risponderà con 400 BAD REQUEST. Mentre se le informazioni sono presenti ma si sta tentando di eliminare un server non registrato, la risposta sarà 404 FILE NOT FOUND.

Successivamente viene controllato che la password del server corrisponda alla password salvata nel file "servers.json" e che l'utente che sta tentando di eliminare il server sia il suo proprietario. Nel caso in cui una di queste due condizioni non venga soddisfatta, il metodo risponde con 403 FORBIDDEN.

Nel caso in cui tutte le informazioni necessarie siano presenti e corrette, il programma tenta di eliminare i file presenti nel server, eliminare il bucket relativo al server eliminato e aggiornare il file "servers.json" rimuovendo le informazioni del server appena rimosso.
Un qualsiasi errore in questa fase genera una risposta 503 SERVICE UNAVAILABLE.

Nel caso in cui non ci siano errori, il metodo risponde con 200 OK.

Gestione File

Check Files:

```
GET /upload?serverName=_________&serverPassword=__________ HTTP/1.1
Host: projectpdgt.herokuapp.com
```

Una richiesta HTTP GET su /upload fornisce una lista di nomi di file presenti dentro al server indicato nella query “serverName”.

Se la richiesta non contiene informazioni riguardanti il server da accedere, la risposta sarà 400 BAD REQUEST.

Se il server indicato nella query “serverName” non è presente come server registrato, il metodo risponderà con 404 FILE NOT FOUND.

Nel caso in cui la password fornita nella query “serverPassword” non corrisponda alla password del server indicato, il programma risponde con 403 FORBIDDEN.

Se tutte le informazioni fornite sono corrette, il metodo tenta di accedere alla lista dei file del bucket corrispondente al server selezionato, per scorrere tutti gli elementi presenti (ignorando i file “1.txt” e “2.txt”).

Nel caso in cui il server blob non risponda correttamente e generi errori, il programma risponderà con 503 SERVICE UNAVAILABLE.

Altrimenti se non vengono incontrati errori, la risposta sarà un file json contenente i nomi di tutti i file contenuti nel server specificato.

Upload File:

```
POST /upload?serverName=_________&serverPassword=________ HTTP/1.1
Host: projectpdgt.herokuapp.com
Authorization: Basic ************
Content-Length: N
Content-Type: multipart/form-data;
boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="avatar";
filename="FILENAME / DIRECTORY"
Content-Type: application/octet-stream


(data)
----WebKitFormBoundary7MA4YWxkTrZu0gW
```

Una richiesta POST verso /upload risulta nell'upload del file definito nel body della richiesta tramite form-data, caricato sul server definito dalla query della richiesta ed eseguito dall'utente specificato nel header "Authorization".

Come nella richiesta in precedenza, se mancano informazioni sul server selezionato, il metodo risponderà con 400 BAD REQUEST. Nel caso in cui il server specificato non esista, la risposta sarà 404 FILE NOT FOUND.

Se la password del server è errata oppure se le credenziali di accesso definite in "Authorization" sono errate o mancanti, oppure nel caso in cui l'utente che sta compiendo la richiesta non sia il proprietario del server specificato, la risposta del servizio sarà 403 FORBIDDEN.

Una volta verificata la correttezza dei dati di identificazione di utente e server, e validati i permessi di upload, il metodo cerca tra i file già presenti nel server, e nel caso in cui trova un file con lo stesso nome del file che si tenta di caricare, la risposta sarà 409 CONFLICT.

Se invece il file è un nuovo file e non crea nessun conflitto, il metodo tenta di caricare il file nel bucket corrispondente al server di destinazione. Nel caso in cui ci sia un errore, la risposta sarà 503 SERVICE UNAVAILABLE.

Altrimenti risponderà con 201 CREATED.

Download File:

```
GET /download?serverName=_____&serverPassword=_____&fileName=_____ HTTP/1.1
Host: projectpdgt.herokuapp.com
Authorization: Basic **********
```

Con questa richiesta è possibile scaricare tramite protocollo HTTP il file denominato da “fileName”, appartenente al server “serverName”.

Mancanza di informazioni necessarie come le query “serverName”, “serverPassword” e “fileName” generano una risposta 400 BAD REQUEST.

Se le credenziali utente definite in “Authorization” non sono corrette o se la password del server non corrisponde a quella registrata, il programma risponderà con 403 FORBIDDEN.

A questo punto il metodo tenta di ottenere la lista di file presenti nel server e scorrere tutti gli elementi fino a trovare il file indicato.

Un qualsiasi errore in questo procedimento causato dal server di blob storage genererà una risposta 503 SERVICE UNAVAILABLE, mentre nel caso in cui il file indicato non sia presente nel server selezionato, la riposta sarà 404 FILE NOT FOUND.

Nel caso in cui il file esista e non ci siano errori da parte del server di blob storage, il programma genera una risposta contenente il file selezionato come corpo della risposta.

Delete File:

```
DELETE /upload?serverName=_____&serverPassword=_____&fileName=____ HTTP/1.1
Host: projectpdgt.herokuapp.com
Authorization: Basic *************
```

Una richiesta DELETE su /upload genera la rimozione del file “fileName” sul server “serverName” compiuta dall’utente specificato in “Authorization”.

Nel caso in cui manchino informazioni come le 3 query necessarie, la risposta del servizio sarà 400 BAD REQUEST.

Se non esistono server registrati con il nome specificato da “serverName”, il metodo risponde con 404 FILE NOT FOUND.

Se l’utente non è autorizzato, la password del server fornita non è corretta oppure nel caso in cui l’utente che genera la richiesta non è il proprietario del server selezionato, il programma risponderà con 403 FORBIDDEN.

Se tutte le informazioni sono presenti e corrette, il programma tenta di ricavare la lista di file presenti nel server, per controllare se il file che si cerca di rimuovere è presente.

Se ci sono errori da parte del server Minio, la risposta sarà 503 SERVICE UNAVAILABLE.

Se il file non è presente nel server selezionato, il metodo risponderà con 404 FILE NOT FOUND.

Se il file esiste, il programma tenta di eliminarlo dal bucket corrispondente al server selezionato. Il successo di questa operazione genera una risposta 200 OK, altrimenti 503 SERVICE UNAVAILABLE.

Oltre a questi metodi, sono presenti metodi aggiuntivi di utility (GET /check per controllare il contenuto dei file users.json e servers.json, GET /hash per verificare il funzionamento della funzione di codifica sha256, POST /minio per cambiare il server minio).

Il più importante tra questi è il metodo che risponde alla richiesta:

```
POST /minio?serverAddress=___________&serverPort=_______ HTTP/1.1
Host: projectpdgt.herokuapp.com
```

Questa richiesta consente di cambiare l'indirizzo del server di blob storage Minio.
L'indirizzo viene specificato nella query "serverAddress" e la porta viene specificata nella query "serverPort". L'indirizzo finale risulterà: "serverAddress:serverPort".
Al momento ci sono 4 server Minio tra cui scegliere, i dati non sono condivisi tra server:

- **mathorgadaorc.ddns.net:9000**
- **solidgallium.ddns.net:9000**
- **solidgallium.ddns.net:9002**
- **solidgallium.ddns.net:9004**

I DNS sono forniti dal servizio "NO-IP".

Nel momento in cui il server minio viene aggiornato, le variabili interne che contengono la lista di utenti registrati e la lista di server disponibili vengono automaticamente aggiornate con i contenuti dei file "users.json" e "servers.json" nel bucket "info" del nuovo server, così da poter accedere al nuovo server correttamente e senza reiniziare l'API.

Descrizione del Client che Sfrutta il Servizio

Nella repository è presente un client scritto in Python per generare le richieste HTTP e sfruttare le funzionalità messe a disposizione dall'API.

Il programma è interagibile tramite frecce direzionali per spostarsi nell'interfaccia e Invio per selezionare

Il client sfrutta librerie esterne:

- **requests**: libreria per generare e ricevere richieste e risposte HTTP
- **base64**: libreria per codificare informazioni in base64
- **re**: libreria per implementare operazioni regolari
- **os**: libreria per interagire con il sistema operativo
- **keyboard**: libreria per interagire con la tastiera
- **sys**: libreria per utilizzare chiamate di sistema
- **stdiomask**: libreria per censurare la password a tempo di digitazione
- **time**: libreria per utilizzare funzioni di sleep
- **termcolor**: libreria per rappresentare testo di colore diverso al terminale
- **tkinter**: libreria per creare una finestra di dialogo e scegliere un file da caricare
- **termios**: libreria per pulire il buffer di input da tastiera per Ubuntu 20.04
- **msvcrt**: libreria per pulire il buffer di input da tastiera per Windows 10

Il programma si divide in 3 principali schermate:

- **Login**
- **Login Server**
- **Contenuto Server**

Login

La schermata di login è quella che da accesso alla verifica dei dati personali (credenziali), consente di verificare il propri dati e generare un cookie, che verrà salvato in un file locale chiamato "cookie.txt" ed utilizzato al prossimo avvio per eseguire l'accesso automaticamente. Consente di registrare nuovi utenti, di eliminare utenti esistenti e di uscire interamente dal programma.

Le opzioni disponibili in questa schermata sono:

- **Login**: richiesta GET /login. Se accettata, fornisce accesso alla pagina successiva.
- **Login with Cookie**: richiesta GET /loginCookie. Se accettata, fornisce accesso alla pagina successiva, genera un file cookie.txt ed esegue l'accesso automatico al prossimo utlizzo.
- **Register**: richiesta POST /users. Se accettata, fornisce accesso alla pagina successiva e genera una nuova coppia utente e password nella lista di utenti.

- **Delete**: richiesta DELETE /users. Se accettata, rimuove l'utente specificato dalla lista di utenti registrati.
- **Close**: termina il programma.

Login Server

In questa schermata è possibile creare nuovi server, il cui proprietario sarà l'utente collegato al momento della creazione. E' possibile accedere ad un server esistente ed elencato nella lista dei server disponibili. Oppure eseguire il Logout per ritornare alla pagina precedente.

- **New Server**: richiesta POST /servers. Se accettata, genera un nuovo server con nome e password fornite dall'utente e il cui proprietario sarà l'utente collegato, e si collega automaticamente al nuovo server creato, dando accesso alla schermata successiva.
- **Login Server**: richiesta GET /servers. Se accettata, fornisce accesso al server selezionato grazie alla password fornita dall'utente, e mostra la schermata successiva.
- **Logout**: esegue il logout dall'account corrente. Elimina ogni cookie salvato e ritorna alla schermata precedente.

Contenuto Server

Questa è la schermata che fornisce funzionalità come la visualizzazione dei file contenuti nel server selezionato, opzioni di download, upload e cancellazione di file. Possibilità di cancellare il server stesso, eseguire il logout oppure terminare il programma.

- **Upload File**: richiesta POST /upload. Solo il proprietario del server può caricare file. Se accettata, il file verrà caricato nella lista di file del server e nel server Minio.
- **Download File**: richiesta GET /download. Chiunque abbia accesso al server può utilizzare questa funzionalità. Se accettata, verrà eseguito il download del file selezionato e salvato nella cartella "download" presente nella directory del client. Se la cartella non esiste, verrà creata.
- **Delete File**: richiesta DELETE /upload. Solo il proprietario del server ha accesso a questa funzione. Se accettata, il file selezionato verrà rimosso dalla lista dei file del server, e dal bucket Minio associato al server selezionato.
- **Change Server**: esegue il logout dal server e ritorna alla schermata precedente.
- **Delete Server**: richiesta DELETE /servers. Solo il proprietario del server ha accesso a questa funzione. Se accettata, il server selezionato verrà rimosso dalla lista dei server, il bucket Minio associato verrà eliminato e con se, tutti i file al suo interno verranno cancellati. Ritorna alla schermata precedente.
- **Logout**: esegue il logout dall'account corrente. Elimina ogni cookie salvato e ritorna alla schermata precedente.
- **Close**: termina il programma.

Inclusi nel client sono presenti degli script per Windows e per Ubuntu per installare automaticamente tutte le dipendenze necessarie per il programma, e per creare un file di avvio “launch” per avviare il software.

Sono presenti anche script per disinstallare tutto il software di dipendenza del programma e lo script di avvio “launch”.
Prestare molta attenzione durante la disinstallazione, nel caso in cui il programma tenti di disinstallare software utilizzato dall’utente.

Questo software è stato testato su macchine con Windows 10 e Ubuntu 20.04.

Nel caso di Windows 10, è presente un errore nel software per pulire il buffer di input del terminale “msvcrt”. In seguito ad una chiamata del terminale “cls” (operazione per pulire il terminale), i primi input da tastiera non vengono correttamente catturati dalla libreria msvcrt. Di conseguenza alcune volte input come “Invio” vengono registrati più volte del previsto. Fornendo ulteriori input (frecce direzionali per esempio) prima di premere “Invio”, si può evitare questo problema.

Nessun errore simile riscontrato in Ubuntu 20.04 utilizzando la libreria “termios”.

Descrizione Server Blob Storage Minio

Il servizio sfrutta un server di blob storage presso l'indirizzo "mathorgadaorc.ddns.net:9000", utilizzando il software Minio.

Lo spazio di archiviazione è diviso in bucket (contenitori).

Ogni contenitore prende il nome del server a cui fa riferimento.

Dentro ogni contenitore sono presenti due file (generati al momento della creazione del bucket): "1.txt" e "2.txt". Questi file vengono generati dall'API al momento della creazione di un nuovo server per evitare un bug nell'API Minio: un operazione di accesso ad un bucket tramite API javascript, risulta in un errore nel caso in cui ci siano 1 o meno elementi.
Creando questi due file (nascosti all'utente) si evita questo problema.

Nel server Minio è presente un bucket speciale chiamato "info". Questo bucket contiene i file "servers.json" e "users.json", utilizzati per salvare la lista di utenti e password e la lista di server, password e proprietari che rappresentano gli utenti e i server registrati ed utilizzabili.

Il codice sorgente dell'API e del client sono disponibili su github sotto licenza GNU GPT3: ProjectPDGT