Università degli Studi di Trieste – Dipartimento di Ingegneria e Architettura

Progetto di Programmazione Web

Matteo Ferfoglia - 11 aprile 2021

**Link al repository con il codice sorgente:** <https://github.com/matteoferfoglia/progetto-progrweb-2020>

**Link alla versione live del progetto:** <https://progettoprogrweb2020.appspot.com>

All’avvio della piattaforma per la prima volta vi è solo un utente *Administrator*, con username [**admin**] e password [**\*\*\***].

Il progetto implementa tutte le funzionalità richieste dai requisiti ed è stato realizzato basandosi sul materiale delle lezioni, fatta eccezione per alcune porzioni di codice, la cui fonte è comunicata.

# Tecnologie utilizzate

In questo progetto sono stati realizzati una *web application* ed un *client* ***REST***,dimostrativo del funzionamento del *web service* esposto dalla *web application*, entrambi implementati in *Java* (versione 8, utilizzando le specifiche *Java EE 7*), usando ***Maven*** (versione 3.5) come strumento di *build automation*.

Si è utilizzato il *framework* ***Vue.js***(versione 3) per realizzare una *Single Page Application* (*SPA*)che interagisse con il web server, con gli obiettivi di separare la logica dell’applicazione (gestita dal server) dalla rappresentazione dei dati e di fornire agli utenti (i quali accederanno alla piattaforma tramite browser) un’ottima *user experience*, che assomigli a quella di un’applicazione desktop. Si è utilizzato ***axios*** per le interazioni *HTTP* con il *web server*, perché viene fornito con un’ottima documentazione ed è facilmente configurabile ed integrabile nel progetto *Vue* (usando ***npm*** come gestore di pacchetti).

Il *servlet container* utilizzato è ***Google App Engine***. Il progetto *Maven* della *web application* è stato inizialmente generato con *Google Cloud SDK*, usando l’archetipo com.google.appengine.archetypes:appengine-standard-archetype versione 1.0.2, seguendo la documentazione[[1]](#footnote-1). Si è deciso di usare ***Datastore*** di *Google* come *DBMS*, perché è ben integrato con *App Engine* e fornisce un sistema di archiviazione scalabile e robusto, ed ***Objectify*** come *ORM*, vista la sua semplicità d’uso. Si è utilizzato ***Jersey*** (implementazione di riferimento della tecnologia ***JAX-RS***) per realizzare le *API REST* sulle quali si basa l’interazione tra client e server. L’applicazione *Java* fa uso di classi astratte ed “*helper*” nell’intento di mantenere un sufficiente livello d’astrazione tale da permettere, se necessario in futuro, la migrazione a tecnologie diverse.

I dati scambiati tra client e server sono perlopiù in formato ***JSON***: ciò permette il trasferimento delle informazioni con basso overhead e senza imporre un accoppiamento tra client e server; per ragioni di efficienza, fanno eccezione alcune richieste “semplici” a cui il server risponde in formato testo (*text/plain*) e le richieste di caricamento (in formato *multipart/form-data*) e di scaricamento (in formato *application/octet-stream*) dei file[[2]](#footnote-2).

Si è deciso di esporre un *web service* di tipo *REST* perché l’overhead del traffico è minore rispetto a quello che si avrebbe utilizzando *SOAP* (che fa uso di *XML*); inoltre, il progetto *Maven* includeva già le dipendenze necessarie[[3]](#footnote-3).

Il *web-server* è stato configurato (tramite il file *web.xml*) come da documentazione[[4]](#footnote-4) per accettare solo richieste *HTTPS*.

L’invio dei messaggi e-mail è stato implementato utilizzando la ***API Mail*** di *Google App Engine* (come da documentazione[[5]](#footnote-5)).

Si è implementato un meccanismo di autenticazione di tipo *Bearer* [[6]](#footnote-6): dopo che un client si autentica con il server, quest’ultimo gli rilascia un *token* ***JWT*** che viene salvato nella memoria del browser (in *localStorage*) e deve essere presentato nelle successive richieste HTTP (inserendolo *nell’header* “*Authorization”*, preceduto dalla parola “*Bearer”*). Questo meccanismo permetterebbe anche l’integrazione dell’applicazione con servizi esterni, infatti, utilizzando un sistema a chiavi asimmetriche, il server che verifica il token, potrebbe essere diverso da quello che lo ha emesso. Gli utenti possono autenticarsi *i)* fornendo *username* e *password* (le credenziali degli utenti vengono memorizzate in modalità *hashed&salted* e sono stati implementati dei meccanismi per la verifica dell’account, a seguito della registrazione di un nuovo utente, e per il recupero della password) oppure *ii)* tramite il proprio account *Google* (l’utente viene identificato dall’indirizzo e-mail ottenuto tramite ***Firebase***). Come ulteriore meccanismo di sicurezza, contestualmente all’emissione del *token JWT*, il server imposta anche un ***cookie***, con l’attributo *HttpOnly*, il cui valore è un *token* alfanumerico generato casualmente, di cui viene calcolato l’*hash,* utilizzando la password dell’utente (*hashed&password*) come chiave, e tale valore di *hash* viene inserito nel *payload* del token *JWT*. In questo modo, il server può verificare che il *token* venga presentato dallo stesso client a cui era stato rilasciato, andando a ricalcolare la firma del valore del cookie e confrontandola con il valore nel *payload* del *token JWT*. Inoltre, se l’utente associato al *token* modifica la sua password, allora tutti i *token* rilasciati prima della modifica diventano automaticamente invalidi.

Nella *web application* sono stati implementati dei ***filtri*** che, applicati nell’ordine specificato nel file *web.xml*, intercettano le richieste *HTTP* per verificare l’autorizzazione del client prima di procedere.

L’applicazione implementa un meccanismo di prevenzione degli attacchi ***CSRF***: ogni *form* usato per modificare lo stato del sistema include un campo *input* di tipo *hidden* contenente un *token CSRF* alfanumericogenerato casualmente; lo stesso valore viene anche salvato nel *payload* di un *token JWT* (autenticità ed integrità dati dalla firma del server apposta sul token) salvato in un *cookie*. Tutti i *cookie* creati dal server sono configurati con l’attributo *SameSite* impostato a “*Lax*”, per evitare che vengano allegati alle richieste “*cross-site*” [[7]](#footnote-7), e sono di tipo *HttpOnly,* per evitare che vi si possa accedere tramite *JavaScript*, e *Secure*, cosicché vengano inviati solo con il protocollo HTTPS.

Si fa uso della *dependency Maven* org.owasp.encoder:encoder per prevenire le vulnerabilità *XSS*.

La documentazione (incluso ***Javadoc***) per i progetti *Maven* può essere generata con il comando **mvn site** , dopo avere eseguito la compilazione.

# Configurazione per la compilazione del codice sorgente

Il repository indicato, da clonare nel proprio computer, contiene tre sottocartelle con i codici sorgente *i)* della *Java “Google App Engine” Web Application*, *ii)* del client *REST* e *ii*) del progetto *Vue* che realizza l’interfaccia per il browser. I primi due sono progetti *Java* (si richiede *Java 8*), realizzati usando *Maven* (la versione minima richiesta è la 3.5) come strumento di *build automation*: una volta installato quest’ultimo, tali progetti possono essere compilati posizionando il terminale nella cartella del progetto di interesse (la stessa in cui è presente il file *pom.xml*) ed eseguire il comando **mvn clean package** . Ad ogni progetto è allegato un file *README.md* con maggiori informazioni sul progetto stesso. L’applicativo client *REST*, dopo essere stato compilato, può essere eseguito con finalità dimostrativa usando il comando **mvn exec:java** . La *web application*, dopo essere stata compilata[[8]](#footnote-8), può essere eseguita nel server di sviluppo locale di *Google App Engine* con il comando **mvn appengine:run** (ciò richiede l’installazione di *Google Cloud SDK*, che avviene in modo automatico). La compilazione del progetto “*Client REST*” produce un file *JAR* nel quale sono “assemblate” tutte le dipendenze per il funzionamento e può essere direttamente distribuito agli utilizzatori, con il requisito che questi abbiano *Java 8* installato.

1. <https://cloud.google.com/appengine/docs/standard/java/using-maven#creating_a_new_app> (visitato il 10 aprile 2021). [↑](#footnote-ref-1)
2. Il codice per l’upload ed il download dei file è stato tratto rispettivamente da <https://stackoverflow.com/a/25889454> e <https://stackoverflow.com/a/12251265> (visitati il 10 aprile 2021). L’utilizzo del *Content-Type* “*multipart/form-data”* ha richiesto la dipendenza org.glassfish.jersey.media: jersey-media-multipart (tratto da: <https://stackoverflow.com/a/29227044>) e la modifica del file di configurazione *web.xml* (fonte: <https://stackoverflow.com/a/25312655>). Il client *REST* invia i dati in formato *multipart/form-data* come descritto in <https://stackoverflow.com/q/24637038>. Il codice per il download di un documento tramite codice *JavaScript* è tratto (ed opportunamente adattato) da <https://stackoverflow.com/q/33247716>, quello per l’upload in formato *multipart/form-data* da <https://stackoverflow.com/a/43014086>. [↑](#footnote-ref-2)
3. *Jersey* è anche usato per rispondere alle richieste della *SPA*. [↑](#footnote-ref-3)
4. <https://cloud.google.com/appengine/docs/standard/java/config/webxml#secure-urls> (visitato il 10 aprile 2021). [↑](#footnote-ref-4)
5. <https://cloud.google.com/appengine/docs/standard/java/mail/sending-mail-with-mail-api#sending_email_messages> [↑](#footnote-ref-5)
6. Idea tratta da <https://metamug.com/article/security/jwt-java-tutorial-create-verify.html> (visitato il 10 aprile 2021). [↑](#footnote-ref-6)
7. Fonte: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite#lax> [↑](#footnote-ref-7)
8. La compilazione dei file sorgenti del progetto *Vue* avviene automaticamente durante la fase di *package* della *web application* (grazie al *plugin Maven* com.github.eirslett:frontend-maven-plugin, codice tratto da: <https://stackoverflow.com/a/62535901> e da <https://stackoverflow.com/q/28584442>). [↑](#footnote-ref-8)