Università degli Studi di Trieste – Dipartimento di Ingegneria e Architettura

Progetto di Programmazione Web

Matteo Ferfoglia - 12 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>

*Credenziali per l’accesso*: username [**admin**] e password [**\*\*\***]

In questo progetto sono stati realizzati una *web application* ed un *client* ***REST***(dimostrativo del funzionamento del ***web service*** *REST* esposto) entrambi implementati in *Java* (versione 8, utilizzando le specifiche *Java EE 7*), usando ***Maven*** (versione 3.5) come strumento di *build automation*. 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, seguendo la documentazione[[1]](#footnote-1).

Si sono usati ***Datastore*** di *Google* come *DBMS*, perché è ben integrato con *App Engine* e fornisce un sistema di archiviazione scalabile e robusto, ed ***Objectify*** (versione 5) come *ORM*, per la sua semplicità d’uso. Si è utilizzato ***Jersey*** (implementazione di riferimento della tecnologia ***JAX-RS***) per realizzare le *API REST* (documentate con ***Swagger***) sulle quali si basa l’interazione tra client e server. L’invio dei messaggi e-mail sfrutta la ***API Mail*** di *Google App Engine*.

Si è realizzata una *Single Page Application* (*SPA*) utilizzando il *framework* ***Vue.js***(versione 3), insieme a ***Bootstrap*** (versione 4) per lo stile grafico e ***Vue Router*** (versione 4) per la navigazione tra i componenti. La *SPA* gestisce la rappresentazione dei dati, ottenuti dal *web server* per mezzo di richieste asincrone effettuate sfruttando il pacchetto ***axios***, che è stato scelto per l’ottima documentazione e la facilità di configurazione ed integrazione nel progetto *Vue* (usando ***npm*** come gestore di pacchetti). I dati scambiati tra client e server sono perlopiù in formato ***JSON***, che garantisce basso *overhead* e non impone alcun accoppiamento tra client e server; tuttavia, 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 meccanismo di autenticazione implementato è di tipo *Bearer* ed è basato sui *token* ***JWT***, le cui proprietà di integrità ed autenticità sono garantite grazie alla firma apposta dal server al momento dell’emissione. Gli utenti possono autenticarsi fornendo *username* e *password* oppure tramite il proprio account *Google* (l’utente viene identificato dall’indirizzo e-mail ottenuto tramite ***Firebase***). Le *password* degli utenti vengono memorizzate in modalità *hashed and salted* (per evitare di fornirle in chiaro ad un eventuale utente malintenzionato che ne sia venuto a conoscenza). Sono stati implementati dei meccanismi per la verifica dell’account, a seguito della registrazione di un nuovo utente, e per il recupero della password.

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. Il *web-server* è stato configurato per accettare solo richieste *HTTPS*, per garantire integrità e riservatezza delle comunicazioni. L’applicazione implementa un meccanismo di prevenzione degli attacchi ***CSRF***: ogni *form* usato per modificare lo stato del sistema include un *token CSRF*, che viene anche aggiunto nel *payload* di un *JWT*, memorizzato come valore di un ***Cookie***. Tutti i *cookie* sono configurati con gli attributi *SameSite* (impostato a “*Lax*”, cosicché vengano allegati solo alle richieste nel contesto del sito di origine), *HttpOnly* (per evitare che vi si possa accedere tramite *script* lato client) e *Secure* (affinché vengano inviati solo con il protocollo HTTPS). È stata inoltre utilizzata la *dependency Maven* org.owasp.encoder:encoder per prevenire le vulnerabilità *XSS*.

Il comando mvn clean package, eseguito dal terminale posizionato nella cartella di un progetto *Maven*, ne permette la compilazione. Il progetto *Vue* viene automaticamente compilato ed aggiunto durante la fase di *package* della *web-application*. La documentazione (incluso ***Javadoc***) per i progetti *Maven* può essere generata con il comando mvn site, dopo averne eseguito la compilazione. L’applicativo *client REST* può essere eseguito con il comando mvn exec:java.

1. <https://cloud.google.com/appengine/docs/standard/java/using-maven#creating_a_new_app> [↑](#footnote-ref-1)
2. Il codice per l’upload ed il download dei file è tratto rispettivamente da <https://stackoverflow.com/a/25889454> e <https://stackoverflow.com/a/12251265>. L’utilizzo del *Content-Type* “*multipart/form-data”* ha richiesto la dipendenza org.glassfish.jersey.media:jersey-media-multipart. Il client *REST* invia i dati in formato *multipart/form-data* (codice adattato da <https://stackoverflow.com/q/24637038>). Il codice per il download in di un documento tramite *JavaScript* è tratto (ed opportunamente adattato) da <https://stackoverflow.com/q/33247716>, quello per l’upload da <https://stackoverflow.com/a/43014086>. [↑](#footnote-ref-2)
3. *Jersey* è anche usato per rispondere alle richieste della *SPA*.

   Tutti i link indicati risultano funzionanti al 10 aprile 2021. [↑](#footnote-ref-3)