Introduzione alla programmazione per il web
A.A.2017-2018

Shopping List è un sito web che permette di gestire liste della spesa condivise, mandare messaggi in tempo reale alle persone con cui si sta condividendo una lista e avere una notifica quando un prodotto potrebbe essere da reinserire in lista perché finito.

Shopping List è pensata per massimizzare l'esperienza utente perchè tutti gli elementi con cui l'utente deve interagire solitamente si trovano sulla pagina principale o a pochi click da essa. La gestione asincrona di alcuni aspetti rende l'interazione agile e poco impegnativa.

Shopping List è sicura perchè implementa HTTPS, utilizza filtri Java per impedire accessi non autorizzati a risorse private ed utilizza SHA-256 per crittografare le password all'interno del database, accessibile solo alla macchina su cui è in esecuzione.

Funzionalità

Autenticazione

L'autenticazione avviene mediante form di login. Una servlet permette di autenticarsi controllando email e password. In caso di successo, viene creata una sessione per mantenere lo stato della connessione. Se viene selezionata l'opzione *Ricordami* nel login, viene salvato un token all'interno di un cookie che permette di autenticarsi senza immettere le credenziali anche dopo lo scadere della sessione (30 minuti di inattività).
Per ottenere delle credenziali di accesso si può effettuare la procedura di sign-up. Dopo aver compilato il form per la registrazione al sito si riceve un'email contenente un link per completare la procedura. Questo link comunica al server un token per la conferma dell'indirizzo email. A questo punto si può effettuare il login e iniziare a creare liste e prodotti.
La sicurezza delle proprie informazioni personali è garantita, oltre che dalla crittografia tramite SHA-256 delle password salvate nel database anche dalla presenza di filtri che bloccano tutte le richieste rivolte a risorse che non appartengono all'utente associato alla sessione corrente. Anche l'autenticazione per i web service REST avviene tramite l'identificativo di sessione Java, evitando così l'uso di chiavi di accesso ai servizi.
Dal punto di vista del database gli utenti sono salvati in una tabella che contiene un identificatore univoco, nome, cognome, email e password crittografata. Abbiamo aggiunto inoltre un token che permette di identificare gli utenti che vogliono resettare la loro password: questo viene inviato via email e il link incluso in quest'ultima permette di essere reindirizzati ad un form in cui inserire e confermare la nuova password. Una servlet si prende carico di aggiornare queste informazioni anche sul database.

Funzionalità sincrone

Le principali funzionalità sincrone del nostro sistema sono la gestione (creazione, modifica, eliminazione) delle liste, dei prodotti privati, dei prodotti pubblici, delle categorie e la modifica dei dati del profilo di ogni utente. Esse sono implementate tramite servlet per la parte di controller e JSP-JSTL per la parte di view. Le servlet si occupano dell'accesso ai dati del *model* tramite la creazione di istanze di DAOfactory e degli oggetti DAO, come spiegato nel paragrafo model.

Utente amministratore

Per quanto riguarda il pannello d'amministrazione, abbiamo deciso di svilupparlo in modo che il suo utilizzo sia il più semplice possibile. Infatti il menù in alto contiene le tre sezioni che l'utente amministratore deve gestire (prodotti pubblici, categorie di prodotto e categoria di lista) e un sotto menù sulla destra per il proprio profilo e il collegamento alle pagine del sito. Inoltre le pagine sono strutturate tutte con una barra di ricerca per filtrare i risultati in modo sincrono e una tabella contenente gli elementi delle varie sezioni.

Funzionalità asincrone

La comunicazione asincrona tra server e client (Web Service e WebSocket) avviene tramite il formato Json, per semplicità di utilizzo della libreria Gson lato server e per mantenere semplice la serializzazione e deserializzazione lato client. L'utilizzo di Gson ha permesso di selezionare, per ogni tipo di entità, quali campi serializzare per ridurre il traffico di rete e migliorare le prestazioni.
Viene utilizzato inoltre Vue.js, un Framework JavaScript, per gestire le comunicazioni con i Web Services e WebSocket per rendere alcune funzionalità asincrone, permettendo quindi di eseguire alcune azioni senza il bisogno di dover ricaricare la pagina. La scelta di impiegare funzionalità asincrone è stata dettata dalla nostra voglia di dare maggiore interattività al sito, garantendo un'esperienza fluida nell'utilizzo delle liste. Il framework Vue.js è stato utilizzato per semplificare le operazioni di gestione del *model* lato client e per garantire costante sincronizzazione tra la *view* ed il *model* nel browser web, grazie al *two-way data binding* infatti Vue aggiorna autonomamente i dati *model-view* e viceversa.
Di conseguenza, ogni schermata è costruita tramite una JSP ed in seguito popolata da Vue.js per quanto riguarda la parte di geolocalizzazione, ricerca e autocompletamento, caricamento dei prodotti in lista, modifiche alla lista e creazione rapida dei prodotti, tramite chiamate asincrone ai web services corrispondenti.

Gestione lista corrente

La gestione degli elementi nelle liste è affidata per intero a Vue.js nel client, l'impostazione base della pagina viene effettuata in JSP ma al caricamento della pagina Vue.js richiede l'intera lista al server che serve le risorse attraverso Web Service. L'utente anonimo viene identificato con un token salvato come cookie al primo utilizzo della piattaforma, l'utente registrato è invece identificato da un id cifrato nell'url, così da non esporre gli identificatori interni al database al pubblico. Ogni lista viene caricata al caricamento della pagina ed a intervalli di 2 secondi per garantire che gli oggetti e le quantità sulla lista siano sempre aggiornate. Gli elementi possono essere aggiornati in quantità ed eliminati attraverso un modale Bootstrap. La ricerca ed i suggerimenti di ricerca sono possibili grazie all'utilizzo di chiamate AJAX che forniscono suggerimenti in tempo reale all'utente ed inoltre permettono l'impiego di animazioni altrimenti impossibili da eseguire tramite JSP. Dalla ricerca è possibile aggiungere direttamente un prodotto e visualizzarne informazioni dettagliate. L'autocompletamento permette inoltre di aggiungere rapidamente un prodotto non presente nel database ed immediatamente aggiungerlo alla lista corrente.

Notifiche sui negozi nelle vicinanze

I suggerimenti sulla prossimità di negozi affini alle categorie di liste gestite dall'utente sfruttano la geolocalizzazione messa a disposizione da HTML5 e il servizio *Places Graph*, API RESTful di proprietà di

Facebook. Le richieste a tale API avvengono tutte tramite un Client JAX-RS incapsulato in un Web Service RESTful che esegue le richieste a *Places Graph* per conto del client dell'utente che periodicamente interroga il *GeolocationWebService* inviandogli la posizione dell'utente e aspettando la risposta, che conterrà le informazioni dei negozi che corrispondono alla ricerca. La scelta di implementare le richieste a *Places Graph* server-side deriva dalla necessità di mantenere semplice il servizio di suggerimenti sui negozi, senza dover richiedere l'accesso a facebook dell'utente o senza esporre la chiave di accesso della nostra applicazione del servizio.
Lato client, le informazioni vengono visualizzate su di un modale che, grazie all'ausilio di Vue.js, viene riempito con le informazioni provenienti dal server, in particolare nome, indirizzo, sito web e immagine del negozio associato alle categorie di lista dell'utente.
Per quanto riguarda l'utente anonimo abbiamo semplificato il salvataggio della categoria di lista con l'uso di *Local Storage*, così che la richiesta di informazioni sui negozi avvenga semplicemente mediante il nome della categoria.

Chat

Ad ogni lista è associata una chat di gruppo, in cui tutti gli utenti che hanno i permessi di visualizzare la lista stessa, possono inviare e leggere i messaggi degli altri utenti. Dal punto di vista dell'implementazione abbiamo scelto le Websocket, in modo da permettere la comunicazione bidirezionale tra client e server e ottenere così una chat in tempo reale. Dal punto di vista dell'utente dunque non c'è alcun bisogno di ricaricare la pagina per vedere i nuovi messaggi in entrata, infatti vengono notificati sia tramite un badge sulla sidebar che direttamente nella chat, se quest'ultima è aperta nel momento dell'invio.
Per lo scambio di messaggi tra client e server è stato necessario implementare un piccolo protocollo per il formato dei messaggi (in particolare per le operazioni permesse: "FETCH_CHAT", "SEND_CHAT", ecc…), per definire cosa succede all'apertura della connessione, in caso di errore o segnalare client-side se ci sono nuovi messaggi da leggere.

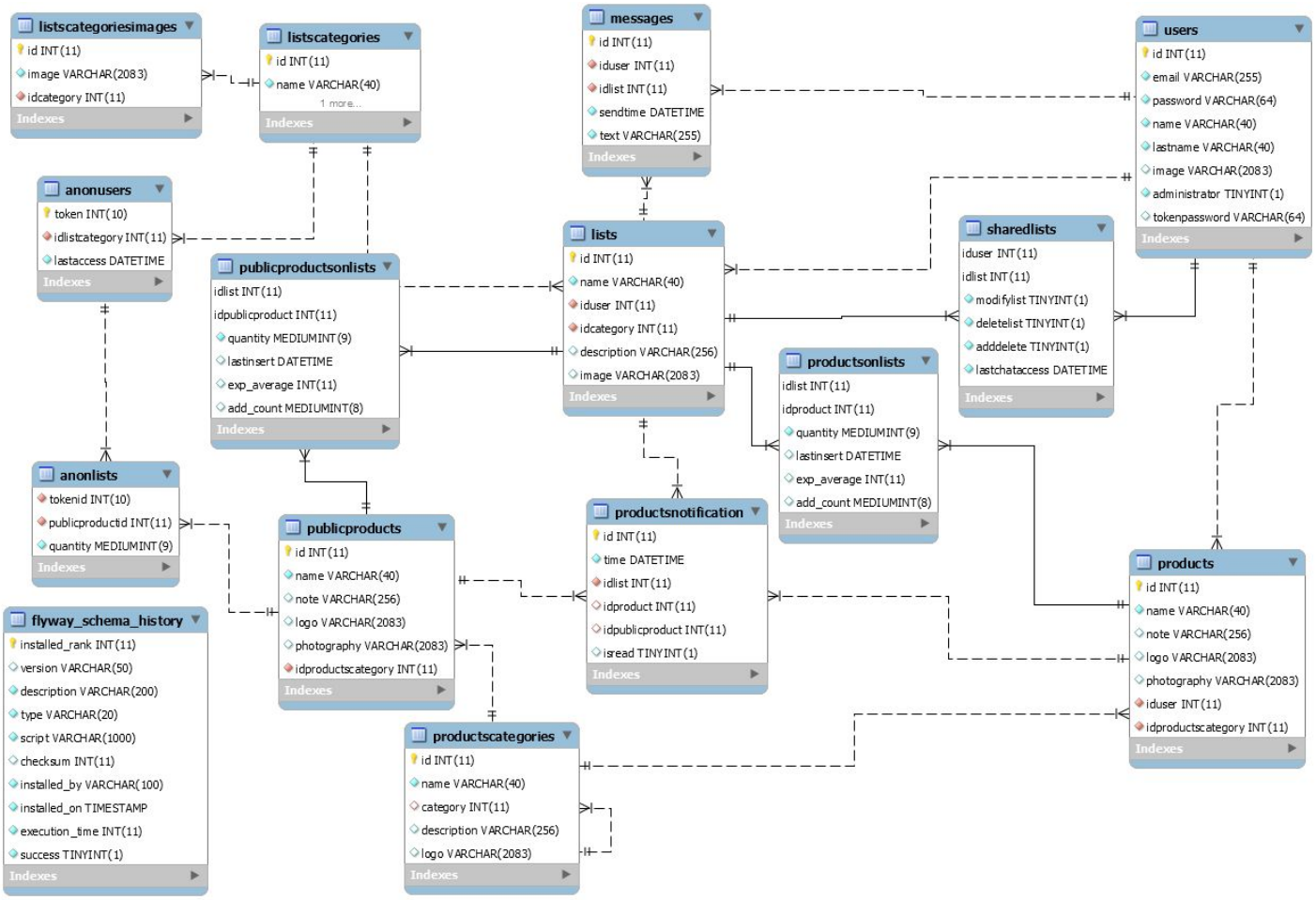
Notifiche sui prodotti frequenti

Se un utente inserisce regolarmente i prodotti in una determinata lista, per almeno 3 volte, allora cerchiamo di prevedere quando sarà la prossima volta che avrà bisogno di quel prodotto tramite una media esponenziale dei tempi che intercorrono tra un inserimento e l'altro. Se allo scadere della previsione il prodotto non è ancora stato inserito in lista (e quindi la data del prossimo inserimento non è ancora stata aggiornata) l'utente riceverà un'email che gli suggerisce che potrebbe aver bisogno di tale prodotto a breve. Sul server è implementata la possibilità di comunicare tali notifiche all'utente anche sul portale, ma siccome non è stata implementata lato client, la lasciamo nel progetto come idea di implementazione per la versione 2.0 di Shopping List.

Model

Abbiamo scelto di utilizzare MySQL (versione 8.0.5), database già utilizzato nel corso di Basi di Dati e che si presenta come un server stand-alone capace di sostenere più connessioni contemporanee a differenza di Apache Derby che nella versione embedded viene integrato nell'applicazione Java che lo utilizza. MySQL è inoltre supportato da *Flyway* come database su cui eseguire le migrazioni.

Presentiamo di seguito il diagramma ER della versione finale del nostro database.



La struttura delle tabelle del database è replicata nei Java Bean (package `ueb.shoppinglist.entities`) che rappresentano le entità corrispondenti nel database.
La connessione con il database viene instaurata tramite driver JDBC per MySQL al momento del deploy dell'applicazione. Ogni azione compiuta sul database avviene tramite questa unica connessione.
Dal punto di vista del codice Java, abbiamo deciso di implementare, per ogni risorsa, un Data Access Object, il quale gestisce l'accesso al database e il passaggio delle informazioni ottenute da esso verso il livello di controllo. Il modulo di Data Access Object è costruito tramite interfacce e rispettive implementazioni, in modo da permettere di cambiare il database mantenendo invariate le modalità di accesso ai dati da parte delle classi che hanno bisogno di accedere al *model*. Questo ha inoltre permesso di utilizzare dei *DAO dummy*, i quali restituiscono informazioni fittizie a chi li invoca e salvano i dati nella memoria dell'applicazione, dando l'impressione di un database già popolato. Grazie a questo abbiamo potuto procedere nello sviluppo dell'applicazione prima che fossero state scritte tutte le queries necessarie.
Nel processo produttivo abbiamo usato *Flyway*, un software da riga di comando che permette di fare migrazioni del database in modo che tutti avessimo sempre la stessa versione del database in locale.

Control

La gestione delle informazioni dal *model* da presentare nella *view* viene eseguita da un layer di controllo in gran parte Java, affiancato dove necessario da codice Javascript lato client. In particolare abbiamo usato le Servlet di JavaEE per per gestire tutte le informazioni scambiate in modo sincrono, i web services di JAX-RS per la gestione della comunicazione asincrona lato-server e i websocket, anch'essi inclusi in JavaEE, quando fosse necessario inviare le informazioni dal server senza una previa richiesta del client. Le classi che fanno parte del *controller* lato server sono le classi che accedono ai dati tramite i DAO e i loro metodi.

Validation

Per controllare lato server l'integrità dei dati inseriti dall'utente e i vincoli del Database sono state utilizzate le validazioni. Queste sono definite nella classe astratta AbstractEntity e poi implementate nelle varie entità. Vengono chiamate come prima istruzione all'interno dei metodi DAO per la comunicazione con il Database in modo che, se i dati non sono corretti, viene settato un errore che verrà mostrato nel form in cui l'utente ha inserito i dati.

Filters

Per quanto riguarda i filtri, questi sono organizzati in modo gerarchico, in base alla struttura degli URL (root filter, restricted, admin…) e svolgono un importante ruolo di sicurezza nell'applicazione. Permettono di prevenire gli accessi alle parti riservate agli utenti registrati senza prima aver effettuato il login (ovvero *esiste un utente associato a questa sessione?*), impediscono a chi non è amministratore di accedere alle *views* relative all'amministrazione (*l'utente associato alla sessione è un amministratore?*).
Inoltre bloccano tutte le richieste dirette a risorse alle quali l'utente stesso non ha accesso, perché non le possiede o non ha i permessi di visualizzazione di una certa lista della spesa. In particolare questa azione viene eseguita direttamente tramite un controllo sugli URL delle richieste, il che permette di bloccare ogni tentativo illecito di manipolazione degli URL stessi, anche nel caso in cui la semplice "crittografia" applicata agli identificativi delle risorse fallisca. Questo avviene per tutte le pagine in cui l'URL della richiesta viene usato per trasmettere informazioni a Vue.js e per tutte le risorse esposte tramite servizi REST.

Listeners

Per quanto riguarda il setup in fase di deploy dell'applicazione, questo viene gestito da una serie di *servlet context listeners* i quali si fanno carico di instaurare la connessione con il database, istanziare gli oggetti responsabili della creazione dei nuovi account, caricare il path della cartella in cui vengono salvati i file caricati da utenti e amministratori. In particolare è interessante spendere due parole sull'upload dei file, i quali vengono salvati nella cartella CATALINA_HOME, per garantire per garantire che essi siano disponibili sotto il dominio del server, senza bisogno di comunicare il loro path completo sul file system tramite URI.

Utilities

Al fine di minimizzare la duplicazione di codice, abbiamo creato alcune classi di utilità, raggruppate nel package `ueb.shoppinglist.utils`, le quali gestiscono l'invio di email (`EmailSender`), la crittografia di password e alcune delle informazioni da esporre al client (`Sha256` e `CookieCipher`), i file e l'accesso al file system (`UploadHandler`) e l'invio dei messaggi di errore all'utente (`HttpErrorHandler`). Infine la classe `Network` si occupa di restituire a chi la invoca nomi e indirizzi di server e database, in modo da dover cambiare un solo punto del codice nel caso in cui si decida di cambiare piattaforma di hosting dell'applicazione.

View

Per la parte di *view* è stato utilizzato Bootstrap come libreria CSS per semplificarne la creazione e garantire alle varie pagine di essere *responsive* così da poter essere utilizzate su qualsiasi dispositivo al meglio. Il sito è diviso in due spazi, uno per l'utente anonimo ed uno per l'utente registrato. La parte per l'utente anonimo è formata da una lista ed un box informativo riguardo le attività nelle vicinanze. Per l'utente registrato abbiamo scelto di fornire ad ogni *view* una sidebar che permette un rapido accesso a tutte le risorse e di una navbar che permette di visualizzare le attività nelle vicinanze con maggiori dettagli rispetto a quelli forniti all'utente anonimo. Per fornire un'esperienza omogenea su ogni pagina abbiamo impiegato file .tag completati con *fragments JSP.* I cambiamenti alla *view* renderizzati attraverso Vue.js vengono effettuati con l'ausilio di componenti per mantenere le varie aree del sito nel loro scope, in questo modo applichiamo la filosofia MVC ad ogni singolo componente dell'interfaccia, facendo così isoliamo dati, template di *view* e logica implementativa nei vari componenti.

Team

Le varie funzionalità sono state sviluppate suddividendo il lavoro tra front-end, back-end Java e database MySQL.

Giulia Peserico

Front-end

Michele Tessari

Database e Back-end

Giulia Carocari

Back-end

Matteo Padovan

Front-end

Simone Lever

Database e Back-end