Progetto di Introduzione alla programmazione per il Web

Matteo Battilana - 180209
Massimo Girondi - 178114
Daniele Isoni - 181839

Versione del January 20, 2018

Contents

1 Tecnologie utilizzate

Il progetto è stato realizzato con Java in versione 7, coadiuvato dalle librerie per lo sviluppo Web tra cui JSP e JSTL.

L'interfaccia Web è stata sviluppata attraverso Twitter Bootstrap 3, con JQuery e HTML5.

Il database scelto è stato MySQL, data la grande affidabilità. Rappresenta il giusto compromesso tra flessibilità e velocità.

All'interno dell'applicazione sono integrate altre librerie JavaScript, tutte liberamente scaricabili, e alcune chiamate alle API di Google Maps, in particolare per l'autocompletamento dei campi e per la visualizzazione della mappa.

Il sito è responsive e supporta la localizzazione mediante le tecnologie offerte da Java Web (resourceBundle e Format Library).

La maggior parte delle pagine sono realizzate mediante una coppia servlet-JSP, soluzione che permette di separare il codice dalla struttura visiva e ottimizza l'uso delle risorse. Sono inoltre presenti alcuni filtri per poter controllare l'accesso a determinate pagine.

2 Installazione

Per poter installare l'applicazione è necessario seguire i seguenti passi:

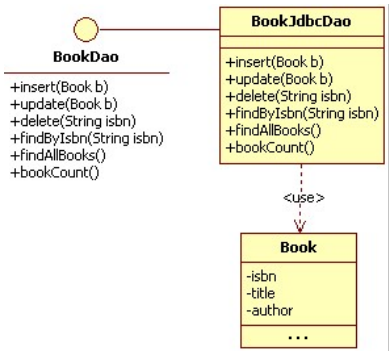
1. Ripristinare il database e i dati presenti nella cartella `DB/DBUtils/`.
2. Copiare la cartella `DB/DBUtils/UploadedContent da copiare` nella cartella **`catalina.base`**`/UploadedContent`. La posizione della cartella **`catalina.base`** varia da installazione a installazione e, solitamente, si trova nel percorso `/opt/tomcat/bin/` (Linux e Mac) o analoghi.
3. Definire quindi le proprie impostazioni nel file `PROJECT/BuyHub/src/main/resources/config.properties`.
4. Modificare le impostazioni per il proprio server di posta elettronica nel file `PROJECT/BuyHub/src/main/java/it/unitn/buyhub/utils/Mailer.java`. In particolare, togliere il commento dalla riga 67 per abilitare l'invio e immetere le proprie credenziali nelle righe 109-126. Per avere i dettagli su come configurare il client email, contattare il provider del servizio.
5. È quindi possibile eserguire l'applicazione all'interno di Apache TomCat o GlassFish.

3 Database

Il database scelto è stato MariaDB, una variante di MySQL. Abbiamo scelto questa distribuzione grazie alla semplice installazione realizzata mediante XAMPP, con il quale viene installato anche PHPMyAdmin, una comoda interfaccia di gestione del database. Un altro punto a favore di questo DBMS è la totale compatibilità con MySQL (tutte le query funzionano anche su un normale database MySQL) e la velocità, oltre alla stabilità e la diffusione pressochè globale.

3.1 Entities DAO - Data Access Object

Per ogni entità presente che necessita di interazioni con il Database è stato creato il relativo DAO per permetterne una corretta interazione. Il DAO (Data Access Object) è un pattern architetturale che ha come scopo quello di separare le logiche di business da quelle di accesso ai dati. L'idea alla base di questo pattern è quello di descrivere le operazioni necessarie per la persistenza del modello in un'interfaccia e di implementare la logica specifica di accesso ai dati in apposite classi. (Fonte: html.it)



3.2 Database ER schema

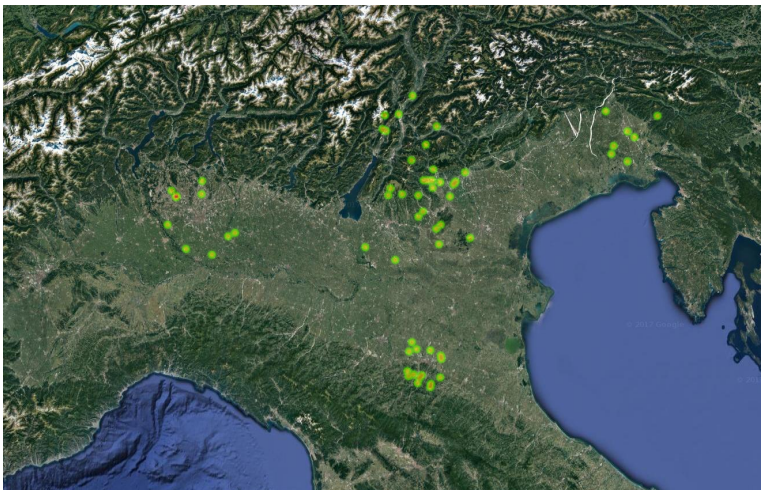
3.3 Popolazione di test

Il DataBase di test (`DB/DBUtils/data.sql`) è così formato:

- 62 punti vendita associati a 31 negozi (almeno un punto vendita a negozio)
- 336 prodotti totali inseriti, distribuiti equamente in tutti i negozi (almeno un prodotto per negozio)
- 327 fotografie (di cui 302 associate a prodotti)
- 122 utenti
- 2298 recensioni, almeno una per prodotto e 7 in media per prodotto

Tutti i campi utilizzati nella nostra applicazione sono stati riempiti con valori verosimili, prelevati dal Web mediante degli script Python, alcuni dei quali presenti nella cartella `DB/DBUtils/scripts`, pertanto non rispettano assolutamente il nostro pensiero e non ci assumiamo nessuna responsabilità del contenuto degli stessi.

I punti vendita sono stati distribuiti casualmente attorno a 6 città (Vicenza, Trento, Bologna, Milano, Udine, Verona). La loro distribuzione è riassunta nell'immagine seguente:



4 Autocompletamento

4.1 Barra di ricerca

L'autocompletamento presente nella barra di ricerca è stato realizzato mediante la libreria JavaScript **bootstrap-ajax-typeahead**, la quale recupera i suggerimenti per l'autocompletamento dalla servlet AutoCompleteServlet mediante richieste Ajax, a cui la servlet rispone in formato JSON.

La servlet effettua una ricerca per similarità (secondo l'algoritmo di Jaro-Winkler) all'interno di una lista, la quale viene caricata all'avvio dell'applicazione analizzando i nomi dei prodotti presenti nel database e aggiornata a intervalli prestabiliti (ogni 30 minuti) mediante le classi Executors e ScheduledExecutorService presenti all'interno di Java stesso.

Abbiamo scelto questa implementazione per evitare di sovraccaricare il database eseguendo una query a ogni richiesta di autocompletamento (e quindi a ogni tasto premuto). Un'altra opzione, poi scartata, era aggiornare la lista dell'autocompletamento a ogni modifica della tabella dei prodotti, ma questo avrebbe rallentato ogni operazione sulla tabella stessa.

L'algoritmo di Jaro-Winkler è utilizzato anche durante la scansione del database, per evitare di inserire nella lista dei suggerimenti titoli doppi o troppo simili (ad esempio "frigo" e "frigorifero") dato che questi verrebbero comunque selezionati al momento della ricerca, dove è applicata ancora una volta la similarità.

4.2 Indirizzo

L'autocompletamento presente nel form di creazione di un nuovo negozio per l'indirizzo è stato utilizzato mediante la libreria JavaScript **Google Maps API**. Queste API permettono tramite chiamate asincrone di ricevere i suggerimenti i quali vengono mostrati mediante un dropdown. Una volta fatta la scelta vengono riempiti i campi del bean con indirizzo completo, e coordinate (latitudine e longitudine).

5 Risultati della ricerca

la ricerca tra i prodotti è realizzata mediante una servlet che elabora la richiesta e restituisce la lista dei prodotti sottoforma di JSON. L'unico parametro obbligatorio è `q` che rappresenta la query di ricerca. I parametri opzionali sono:

- `p`: la pagina desiderata (nel caso di una ricerca multipagina)
- `s`: il metodo di ordinamento dei risultati, di default alfabetico (0), alfagetico inverso(1), prezzo crescente (2), prezzo decrescente(3), valutazione decrescente(4), numero recensioni decrescente(5).
- `c`: la categoria
- `min` e `max`: gli estremi dei prezzi.
- `minRev`: il minimo del valore delle recensioni.

Il controllo sul nome del prodotto e sul range di prezzo è effettuato all'interno del DAO del prodotto (mediante SQL per il prezzo e algoritmo di Jaro-Winkler per il nome), mentre il controllo sulla categoria e sul valore minimo della media delle recensioni è effettuato mediante un predicato invocato dal metodo `removeIf` della lista prodotti. Oltre alla lista prodotti nel risultato sono restituiti anche il numero di pagine, il tipo di ordinamento usato e la pagina corrente.

6 Logging

Il logging, fondamentale per un'applicazione di questo tipo, è stato implementato mediante la libreria Apache Log4j, che consente di gestire in modo molto semplice i vari livelli di log e il salvataggio automatico sul disco.

Abbiamo poi creato un wrapper attorno alla libreria per limitare il più possibile il numero di istruzioni richieste per scrivere un messaggio nel log.

I log vengono salvati nella cartella `$CATALINA_HOME/logs` in file testuali che iniziano con il nome `BuyHub`. La cartella `$CATALINA_HOME` è in genere la cartella di installazione di TomCat o GlassFish, ma potrebbe variare in base alla configurazione del sistema.

7 Dettagli implementativi

7.1 Orari di apertura e negozi solo online

Per semplificare la struttura del DB abbiamo deciso di implementare i vari punti vendita dei negozi come una relazione uno a molti tra `shop` e `coordinate`. All'interno della tabella `coordinate` abbiamo inserito il campo `opening_hours` che contiene gli orari del punto vendita, nel caso in cui per il cliente sia disponibile il ritiro in negozio. Nel caso di negozi che effettuano soltanto vendita online questo campo resta vuoto e viene sostituito all'interno della pagina web con un messaggio che specifica le modalità di vendita solo online del negozio.

7.2 Carrello

Il carrello utente, visto come contenitore temporaneo di prodotti da acquistare è stato implementato a livello di *sessione*; questo significa che i prodotti aggiunti in una sessione non saranno salvati sul database e quindi in un futuro login il carrello sarà vuoto.

8 Upload/Download

L'upload dei file è stato gestito mediante la libreria `com.oreilly.servlet`, mentre il salvataggio dei file è gestito da alcuni metodi all'interno della classe `Utility`.

In particolare, il metodo `saveJPEG` si occupa di salvare un immagine, dandogli un nome casuale basato su un UUID, convertendola a JPEG, mediante la libreria ImageIO, integrata in Java. Purtroppo questa libreria non gestisce correttamente le trasparenze nelle immagini PNG, e al momento, non è ancora stata trovata una soluzione per ovviare a questo problema, in quanto non è possibile dedurre se questo errore di codifica è avvenuto o no.

I file vengono poi salvati in una cartella definita all'interno del file `config.properties`, nel quale è possibile inserire sia un path relativo (in base alla propria cartella `catalina.base`), oppure un percorso assoluto, che verrà automaticamente interpretato e utilizzato per il salvataggio.

Il dowload avviene mediante la servlet `UploadedContentServlet`, che risponde alle richieste che arrivano alle url del tipo `/UploadedContent/*`, in modo tale da simulare a tutti gli effeti una cartella. Il file richiesto viene inviato in modalità "inline", e con il relativo MIME impostato. Nel caso il file non sia presente nel disco, viene restituita un'immagine che indica la non presenza del file.

La scelta di utilizzare questo sistema è stata piuttosto ardua, in quanto, non volendo usare path assoluti, l'unica opzione per caricare un file direttamente con Java, era caricarlo nella cartella di esecuzione (la `contextPath`). Nel momento in cui però sarebbe avvenuto un redeploy tutti i file sarebbero stati cancellati, oltre al fatto che alcuni server Java non permettono la scrittura nella `contextPath`. Altri server, invece, mantengono direttamente tutto il `.war` dell'applicazione in RAM, senza creare una cartella nel disco rigido, e creare cartelle locali alla `contextPath` in questo contesto è abbastanza sconsigliato (aumenterebbe a dismisura l'uso di RAM).

Un'altra soluzione, volendo realizzare un prodotto per il mondo reale, sono i CDN, in particolare quelli specifici per le immagini, come cloudinary.com o imgix.com. Questi servizi, però, sono in genere a pagamento e non permettono il controllo completo delle proprie immagini. Per questi motivi la scelta è ricaduta nel caricare le immagini in un path locale alla cartella di esecuzione, ma esterno al `contextPath`, in modo da preservare i file tra le diverse esecuzioni e deploy dell'applicazione.

9 Verifica dell'account

Quando un utente si registra al sito, è prevista una conferma della registrazione attraverso la mail, onde evitare account fasulli. All'interno della mail viene inserito un link che contiene un codice speciale che abilita l'utente all'accesso.

In particolare, tale codice è composto dall'ID utente e dall'Hash della password, grazie ai quali è possibile identificare l'utente e, data la bassa probabilità di indovinare l'MD5, rende difficile un eventuale attacco.

Questo codice è poi cifrato mediante AES128 secondo una chiave segreta presente nel server, che ci consente quindi di essere *praticamente* certi che un utente smaliziato riesca ad accedere a un account non proprio. Per poter inviare correttamente il risultato mediante una URL il codice è codificato prima in Base64 e poi attraverso la funzione `URLEncode` di Java. Il codice è quindi così formato:

$$URLEncode(Base64(AES(\text{id\$}MD5(\text{password}))))$$

10 Homepage

Nella homepage vengono visualizzati gli ultimi 10 prodotti inseriti nel database, sulla base dell'ID. Sopra agli ultimi prodotti è presente uno slider con alcune immagini, contenute nella cartella `images/slider_image`
Lo slider supporta la localizzazione: carica, mediante un taghandler, tutti i file presenti nella cartella corrispondente alla lingua selezionata. Ad esempio, se la lingua è italiano, caricherà tutte le immagini presenti nella cartella `images/slider_images/it/`.

11 Altri capitoli

12 Conclusioni