Progetto di Introduzione alla programmazione per il Web

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

Versione del January 27, 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.

Lo sviluppo dell'applicazione è stato gestito facendo uso di Git e GitHub, i quali ci hanno concesso di sviluppare autonomamente diverse parti del progetto, er poi unirle insieme, oltre a mantenere la "storia" delle modifiche e fornire un repository centralizzato per il codice. Abbiamo utilizzato un workflow a branch: ogni componente del gruppo sviluppava su un branch diverso e, al termine delle modifiche, questo veniva unito al master mediante `merge`.

2 Installazione

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

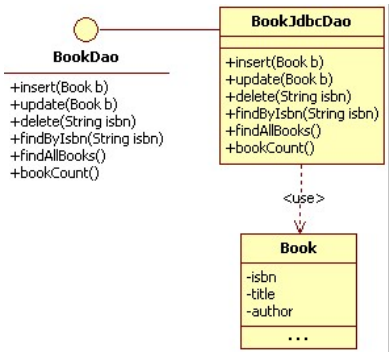
1. Ripristinare il database e i dati presenti nella cartella `DB/DBUtils/`.
2. Copiare la cartella `DB/DBUtils/UploadedContent da copiare` nella cartella **`catalina.base`**`/UploadedContent`. La posizione della cartella **`catalina.base`** varia da installazione a installazione e, solitamente, si trova 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)



Questa archietttura architettura di classi permette di poter estendere l'applicazione su qualsiasi tipo di database, semplicemente riscrivendone le classi di interfaccia con il database.

3.2 Struttura del database

Il database è stato strutturato per essere il più semplice possibile nell'utilizzo. È stato strutturato secondo le principali regole di design relazionale, utilizzando chiavi esterne e indici. Sono state inoltre incluse alcune tabelle e alcuni campi superflui, utili per un'implementazione futura. Di seguito una rapida descrizione delle tabelle:

- **users**: questa è la tabella principale del database, attorno alla quale è costruito tutto il resto. Descrive ogni singolo utente e ne determina anche i privilegi (con un campo intero: 0 se l'utente è disabilitato, 1 se è un utente normale, 2 se è un utente abilitato alla gestione di un negozio, 3 se è un amministratore). Per scelta implementativa un amministratore non può avere un negozio e un negozio può avere solo un utente ad esso collegato.
- **shops**: questa tabella contiene i dati dei singoli negozi. Il campo `validity` descrive l'abilitazione o meno del negozio, mentre i campi `shipment` e `shipment_costs` determinano se la spedizione è disponibile (se il primo campo è vuoto significa che non lo è) e il costo della stessa.
- **coordinates**: in questa tabella sono inseriti i singoli punti vendita. Ogni negozio può infatti avere più punti vendita sul territorio, ognuno con un proprio indirizzo e un poprio orario di

apertura

- **products**: questa tabella contiene tutti i dati dei prodotti in vendita. La categoria è determinata con un ID numerico, che viene poi convertito in testo attraverso il codice Java. Una descrizione delle categorie si trova nei file della lingua e come appendice a questo documento.

- **pictures**: questa tabella contiene i dati delle foto caricate attraverso il sito. Nell'implementazione attuale solo i campi `id`, `name` e `path` sono utilizzati, gli altri sono pensati per una futura implementazione.

- **pictures_products** e **pictures_shops** : mediante questa relazione possiamo collegare le foto ai prodotti e ai negozi.

- **reviews**: questa tabella contiene i dati delle recensioni dei prodotti. Per scelta implementativa è presente un indice sulla coppia `(id_product, id_creator)` che impedisce ad un utente la creazione di più di una recensione sullo stesso oggetto.

- **pictures_reviews**: questa tabella non è al momento utilizzata. Servirebbe per implementare la possibilità di allegare foto alle recensioni.

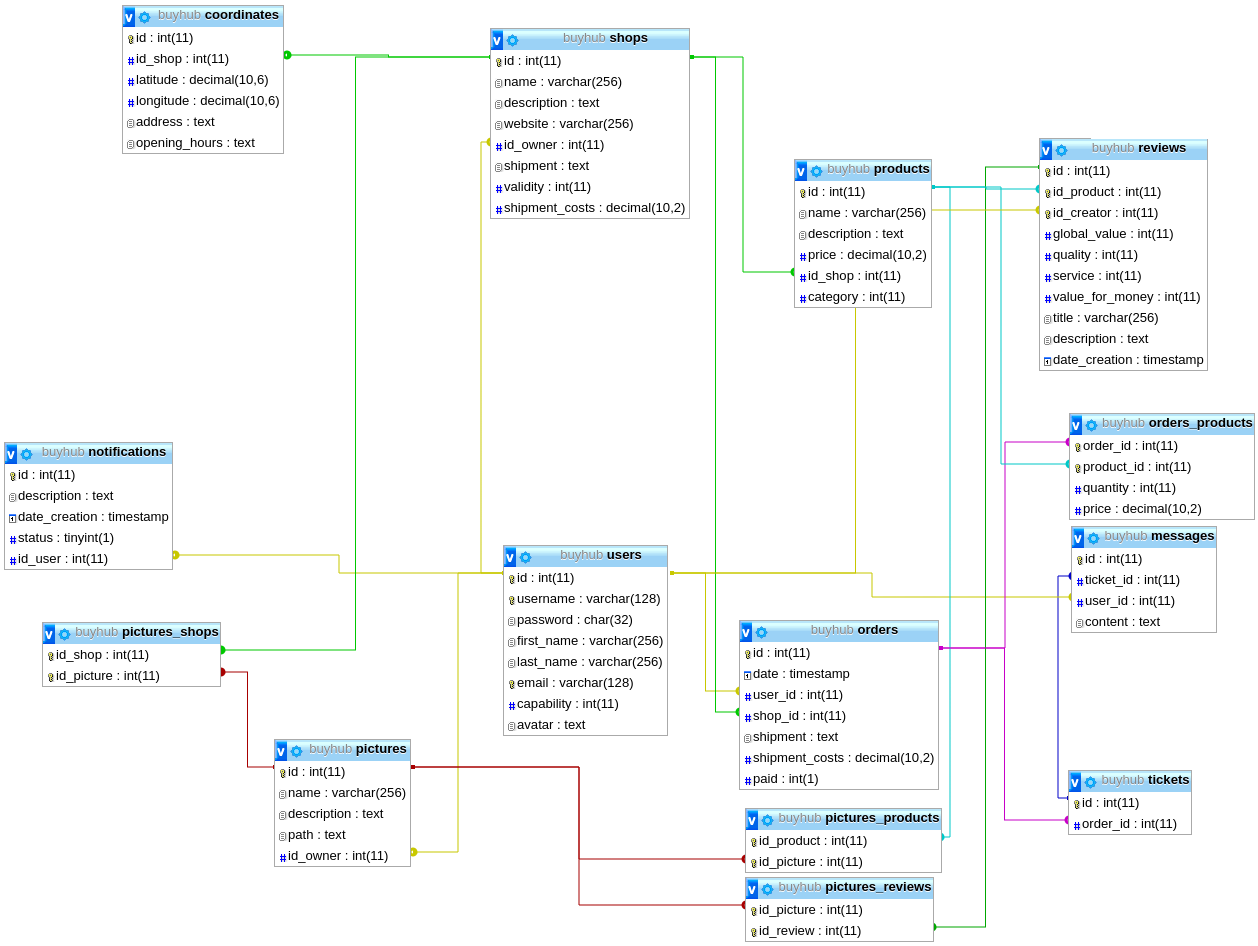
- **orders**: in questa tabella vengono salvati i dati di tutti gli ordini. Viene salvato anche il metodo e il costo di spedizione, dato che nel tempo potrebbero variare (ma un ordine fatto nel passato non deve essere influenzato da questi cambiamenti).

- **orders_products**: mediante questa tabella possiamo salvare i prodotti associati ad un ordine. Oltre alla quantità viene salvato anche il prezzo perchè, come per i costi di spedizione, potrebbero variare nel tempo.

- **tickets**: un ticket è una segnalazione ad un negozio relativa ad un ordine. In questa tabella vengono salvati i ticket aperti in modo tale da sapere se a un dato ordine è associato un ticket.

- **messages**: all'interno di questa relazione sono salvati tutti i messagi associati a un ticket.

- **notifications**: questa entità contiene i dati delle notifiche che vengono inviate agli utenti. Queste sono generate da degli eventi (segnalazione ticket, apertura negozio, ...) e sono mostrate agli utenti nella barra principale.



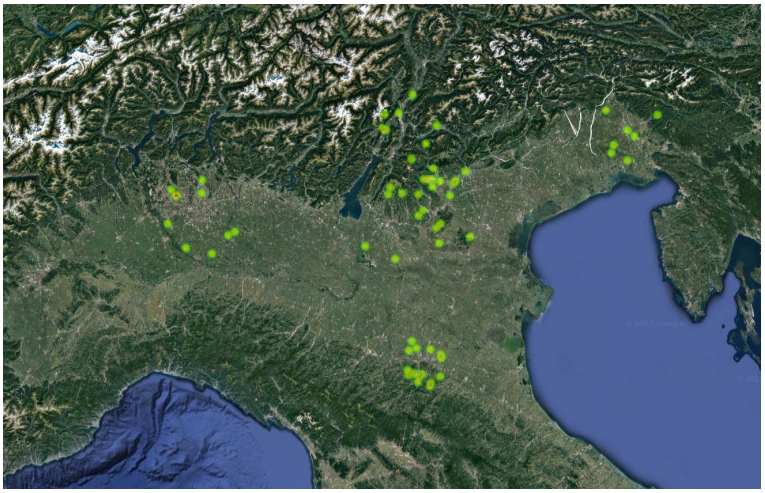
3.3 Popolazione di test

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, ognuno con il proprio avatar
- 2298 recensioni, almeno una per prodotto e 7 in media per prodotto

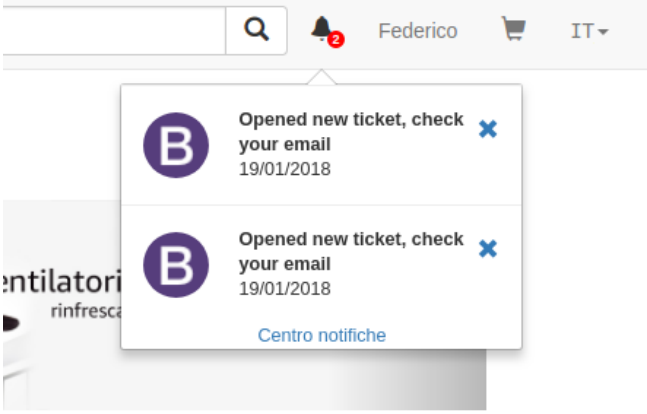
Tutti i campi utilizzati nella nostra applicazione sono stati riempiti con valori verosimili, prelevati dal Web mediante degli script Python, alcuni dei quali presenti nella cartella `DB/DBUtils/scripts`, pertanto non rispettano assolutamente il nostro pensiero e non ci assumiamo nessuna responsabilità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

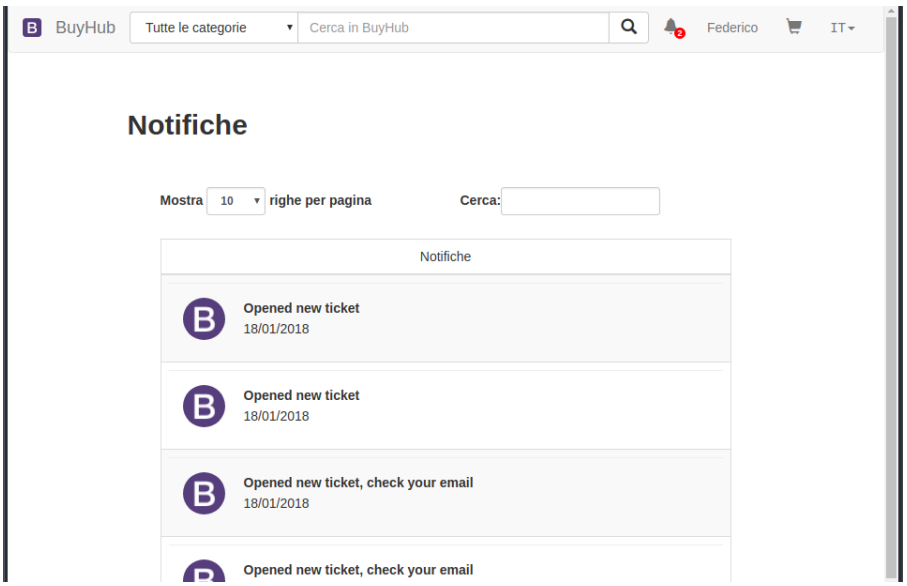
I punti vendita sono stati distribuiti casualmente attorno a 6 città (Vicenza, Trento, Bologna, Milano, Udine, Verona). La loro distribuzione è riassunta nell'immagine seguente:



4 Notifiche

Nella barra di ricerca è stata aggiunta aggiunta inoltre l'icona per le notifiche; cliccando su di essa verrà aperto un pop-up con le notifiche non lette. Nella parte bassa del pop-up è presente un link per visualizzare tutte le notifiche, sia lette che non; inoltre è stata applicata una funzione di ricerca.





5 Anomalie

Dopo aver effettuato un ordine, ossia un insieme di prodotti acquistati da un cliente, è necessario fornire all'utente una protezione sugli acquisti.

5.1 Ticket

Per la gestione delle anomalie e la segnalazione di esse al proprietario del negozio e agli admin è stato implementato il ticket.

Il ticket, si apre recandosi nella sezione ordini dell'utente e cliccando sull'apposito bottone. L'utente verrà quindi reindirizzato verso quella che a tutti gli effetti è una chat. L'utente provvederà quindi a spiegare il problema attraverso questa piattaforma. Ogni qualvolta venga aperto un nuovo ticket, sia il proprietario dello shop che gli admin verrano notificati con una email e un link per accedervi, oltre che alla semplice notifica. L'utente quindi illustrerà il problema e lo shop provvederà a rispondere per cercare un accordo. L'admin può anch'esso rispondere ai messaggi in modo da, se non vi è soluzione, di porre rimedio dando ragione ad una delle parti.

6 Autocompletamento

6.1 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.

6.2 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).

7 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. Al momento della ricezione dei risultati dal server vengono creati anche i pulsanti di navigazione tra le pagine. Questi sono realizzati dinamicamente e presentano sempre almeno 5 pulsanti (se il numero di pagine è superiore a 5). Detto n il numero di pagine e p la pagina corrente, i pulsanti visualizzati (in genere) sono:

- 1
- $p - 1$
- p - la pagina corrente
- $p + 1$
- n

8 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.

9 Dettagli implementativi

9.1 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.

9.2 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. Questa tecnica ci consente inoltre di limitare il più possibile l'interazione con il database, che in caso di grande traffico potrebbe essere rallentato dalla continua aggiunta e rimozione di prodotti dal carrello dei vari utenti.

9.3 Privilegi degli utenti

Il sito prevede 3 categorie di utenti, identificati da un intero:

- 0: utente disabilitato
- 1: utente normale
- 2: utente abilitato alla gestione di un negozio
- 3:amministratore

Un utente amministratore non può avere negozi. Questo è, oltre che un vincolo implementativo, anche un principio di sicurezza: se un'operazione può coinvolgere 2 utenti, questi non possono essere lo stesso utente (*Segregation of duties*).

9.4 Filtri

Sono stati realizzati tre filtri, che permettono di regolare l'accesso all'area riservata e alle pagine dei negozi e amminsitrative, implementando così l'autorizzazione.

- `it.unitn.buyhub.filter.AuthenticationFilter`: controlla l'accesso alle pagine riservate agli utenti abilitati (ovvero la cartella `/restricted/`).
- `it.unitn.buyhub.filter.AuthorizationFilter`: controlla che un utente non possa modificare risorse che appartengono o sono legate ad altri utenti.
- `it.unitn.buyhub.filter.AdminFilter`: limita l'accesso alla sezione amministrativa (ovvero la cartella `/restricted/admin/`) ai soli amministratori.

L'uso dei filtri permette di implementare in maniera più approfondita il pattern MVC, oltre a consentire di escludere la logica di sicurezza dalle singole pagine o servlet.

9.5 Legge sui cookie

Nella giornata del 3 giugno 2015 è entrata in vigore in Italia la cookie law conformemente a quanto stabilito nel provvedimento del Garante per la protezione dei dati personali dell'8 maggio 2014, recante "Individuazione delle modalità semplificate per l'informativa e l'acquisizione del consenso per l'uso dei cookie", adeguandosi, di fatto, alla direttiva comunitaria 2009/136/CE. Questa legge è stata varata con l'intento di arginare la diffusione dei cosiddetti cookie di profilazione e dei connessi rischi per la privacy degli utenti di Internet.

Sebbene la nostra applicazione non faccia uso di cookie di tracciamento e di profilazione, abbiamo ritenuto opportuno inserire una nota a riguardo nell'applicazione. Questa nota viene visualizzata solo alla prima visita e informa l'utente della presenza di cookie tecnici per il buon funzionamento del sito.

10 Upload/Download

L'upload dei file è stato gestito mediante la libreria `org.apache.commons.fileupload`, 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.

La libreria `org.apache.commons.fileupload` è stata scelta in favore della più comune `com.oreilly.servlet` grazie al maggiore supporto, oltre che alla presenza di versioni più recenti rispetto all'altra libreria.

11 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:

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

12 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_images`. 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/`.

13 Mail

Il sito consente l'invio automatico di email. Queste vengono inviate facendo uso della classe `it.unitn.buyhub.utils.Mailer`. All'interno di questa classe sono implemntati diversi componenti:

- La sottoclasse `RunnableMailer`, che implementa l'interfaccia `Runnable`. Questa classe permette di inviare una mail in maniera asincrona e multithread. Questo significa che durante l'invio della mail l'applicazione non deve attendere ma può eseguire altre operazione (ad esempio renderizzare la pagina di avvenuto pagamento).
- Il metodo `mail` che invoca `RunnableMailer`, inviando l'email, e aggiunge al log le informazioni della mail.
- Il metodo `mailToAdmins` che invoca il già noto metodo `mail` per ogni amministratore presente nel database.
- Il metodo `sendMail` che esegue l'invio vero e proprio. Questo metodo è anche chiamato all'interno di `RunnableMailer`.
- `BuildMail`, un metodo che "costruisce" la mail in formato HTML. Viene chiamato all'interno di `sendMail`.

L'invio della mail è effettuato mediante la classe `javax.mail`, che effettua un collegamento `SMTP` e consegna il messaggio al server. Le credenziali di accesso al server sono "hardcoded" all'interno della classe, perciò è richiesto modificarle prima di eseguire l'applicazione.

Abbiamo scelto di salvare le credenziali in questo modo in quanto ogni provider di posta utilizza parametri diversi per l'invio e realizzare un codice funzionante per ogni provider avrebbe causato problemi di sicurezza (ad esempio la connessione cifrata non è supportata da tutti i provider, sarebbe stato quindi necessario disabilitarla).

14 Appendice

14.1 Categorie prodotti

- **0:** Abbigliamento e accessori
- **1:** Alimentari e cura della casa
- **2:** Arte e antiquariato
- **3:** Auto, moto e altri veicoli
- **4:** Bellezza e salute
- **5:** Biglietti ed eventi
- **6:** Cancelleria e prodotti per ufficio
- **7:** Casa, arredamento e bricolage
- **8:** Collezionismo
- **9:** Commercio e Industria
- **10:** Elettrodomestici
- **11:** Elettronica
- **12:** Film e DVD
- **13:** Fotografia e video
- **14:** Francobolli
- **15:** Fumetti
- **16:** Giardino e arredamento esterni
- **17:** Giocattoli e modellismo
- **18:** Gioielli
- **19:** Handmade
- **20:** Illuminazione
- **21:** Infanzia e premaman
- **22:** Informatica
- **23:** Libri e riviste
- **24:** Monete e banconote
- **25:** Musica, CD e vinili
- **26:** Nautica e imbarcazioni
- **27:** Orologi
- **28:** Orologi e gioielli
- **29:** Prima infanzia
- **30:** Prodotti per animali domestici

- **31:** Salute e cura della persona
- **32:** Sport e tempo libero
- **33:** Sport e viaggi
- **34:** Strumenti musicali e DJ
- **35:** Telefonia fissa e mobile
- **36:** TV, audio e video
- **37:** Valigeria
- **38:** Veicoli: ricambi e accessori
- **39:** Videogiochi
- **40:** Videogiochi e console
- **41:** Vini, caffè e gastronomia