Metodologie di programmazione del Web 2018/2019

Documento di progetto

20024195 Gabor Galazzo

28/06/2019

## Plagio (non rimuovere questa sezione)

##### **Consegnando questa relazione al docente, si dichiara implicitamente che il lavoro effettuato (codice del progetto e relativo documento) è frutto di un lavoro strettamente individuale.**

##### **Il plagio (cioè spacciare per proprio un lavoro fatto in realtà da altri) comporta l’annullamento immediato dell’esame, con obbligo di saltare un appello.**

## analisi dei requisiti

Tabella 1 Analisi dei requisiti

|  |  |  |  |
| --- | --- | --- | --- |
| Id. | Requisito obbligatorio (O) oppure “di più” (D) | Descrizione formale e non a | Eventuale ipotesi |
| RU001 | O | Il gestore può mettere on-line un articolo specificando, attraverso una pagina dedicata, i seguenti attributi:  Un identificativo  La sua categoria commerciale (libri, fumetti, videogiochi, CD, vinile e Dvd),  la foto del prodotto e una sua descrizione,  il costo,  il costo di spedizione | Dato che il gestore è unico e si suppone che la fonte sia unica il costo di spedizione non sarà sul singolo prodotto ma sull’ordine. Le categorie di prodotto sono equivalenti dal punto di vista delle spedizioni |
| RU002 | O | Il prodotto è descritto da attributi diversi a seconda del tipo. Per esempio, un libro avrà un autore, un titolo, un editore, un numero ISBN, il numero di pagine e l’anno di uscita. Un CD e il corrispondente vinile avranno l’autore e il titolo, i titoli delle canzoni e la durata di ciascuna, l’anno di uscita. Il Dvd avrà un titolo, un direttore e i nomi di almeno 3 attori (ma non più di 6), l’anno di uscita e il genere (Dramma, sentimentale, fantascienza, giallo, documentario, ecc.) | Si assume che il videogioco sia descritto dai seguenti attributi specifici: Piattaforma, Azienda produttrice, anno di pubblicazione |
| RU003 | O | Qualora la giacenza di un prodotto scenda sotto le X unità, la pagina deve visualizzare un messaggio del tipo “Solo X istanze di questo prodotto sono disponibili” | X = 5 |
| RU004 | O | Nel caso in cui il prodotto sia esaurito, la pagina deve mostrare un messaggio corrispondente, ma deve offrire all’utente la possibilità di prenotarlo (questo vuol dire che il gestore può ricevere o un ordine o una prenotazione). | Si è scelto che il gestore non riceva una “prenotazione” ma possa vedere quali sono gli articoli esauriti e quante prenotazioni vi sono associate (watchers list) |
| RU005 | O | Quando la giacenza del prodotto sia nuovamente maggiore di zero, l’utente che si era prenotato deve ricevere un’e-mail, che lo avvisa che ora è possibile effettuare l’acquisto. | Se il prodotto è tornato disponibile viene semplicemente inviata una e-mail che avvisa l’utente che un prodotto è tornato disponibile. |
| RU006 | O | L’acquisto di un prodotto avviene esclusivamente per mezzo di carta di credito. Quando il gestore riceve un ordine d’acquisto, invia gli estremi della carta di credito (proprietario, numero di carta, scadenza e codice sicurezza) a un gestore di carte di credito separato, che approva o rifiuta il pagamento. Le ragioni di un rifiuto sono: attributi errati o superamento del tetto mensile. | Il gestore di terze parti viene emulato da una serie di funzioni che: valutano la correttezza dei dati della carta inseriti e nel caso di un pagamento il 70% dei casi genera un errore tra “Not enough money on your account", "We cannot contact your bank now", "Service temporary unavailable, please retry after 5 minutes", "Transaction negated from your bank account".  Nella carta di credito è stato aggiunto l’intestatario. (RU008) |
| RU007 | D | Il sistema può memorizzare i metodi di pagamento utilizzati di recente dall’utente. | L’implementazione viene fatta in modo sicuro delegando al sistema di 3e parti la memorizzazione dei dati della carta e mantenendo nel nostro db il riferimento (token\_id) alla carta del servizio di terze parti e i dati “descrittivi” (tipo, ultime cifre, scadenza, intestatario) |
| RU008 | O | Un utente può registrarsi memorizzando i propri dati anagrafici, l’indirizzo di spedizione e la carta di credito (numero, data scadenza, codice di tre cifre e intestatario). L’utente può gestire i propri metodi di pagamento (cancellarli) | Si è scelto di non aggiungere di dati per la spedizione e il pagamento in fase di registrazione ma in fase di checkout dell’ordine come in tutti i principali siti di e-commerce. |
| RU009 | D | Il sistema può memorizzare gli indirizzi di spedizione utilizzati di recente dall’utente. L’utente può gestire i propri indirizzi (cancellarli) |  |
| RU010 | O | Un utente anonimo può visionare il catalogo offerto dal gestore, effettuando un semplice browsing per categoria, oppure specificando un prodotto (anche parzialmente specificato): per esempio il nome di un autore (“Umberto Eco”, “Eco”) o di un attore (“Robert De Niro”, “De Niro”) o una canzone (“Lucy in the sky with diamonds”, “Lucy in the sky”), un videogioco (“Civilization IV” o “Civilization” per considerare le versioni I, II, III, ecc.) o il personaggio di un fumetto (“Spiderman” oppure “Spider”). La ricerca può essere raffinata per “prezzo: crescente”, “prezzo: decrescente”, “media recensioni clienti” e “Ultimi arrivi”.  L’utente può ordinare (crescente/decrescente) gli articoli risultanti dalla ricerca per prezzo o nome. | “media recensioni cliente” ovvero “Only Best” e “Ultimi arrivi” ovvero “Latest” non sono criteri di ordinamento ma filtri di ricerca combinabili con il testo ricercato e la categoria prodotto. Anche la categoria prodotto è un filtro.  I risultati della ricerca verranno paginati. |
| RU011 | O | Una volta che l’utente accede a un articolo, deve poter visionare le recensioni di altri clienti (una recensione ha da 1 a 4 stelle e un testo obbligatorio di lunghezza finita, minimo 80 caratteri e massimo 512). Se lo desidera, deve poter ordinare le recensioni per stelle sia in ordine crescente che decrescente. |  |
| RU012 | O | Per effettuare l’acquisto di un prodotto, l’utente deve essere registrato. Ovviamente, è possibile acquistare più prodotti per volta editando un apposito carrello. |  |
| RU013 | O | Un utente registrato può lasciare una recensione su di un prodotto se lo desidera (non si controlla se l’utente ha acquistato l’articolo o meno). | Il commento finale viene lasciato VOLUTAMENTE in quanto essa è una politica adottata anche da Amazon: chiunque può lasciare recensioni a patto che sia registrato. |
| RU014 | D | Le recensioni possono essere ordinate per “data: crescente”, “data: discrescente” |  |
| RU015 | D | È possibile acquistare più quantità dello stesso prodotto |  |
| RU016 | D | È possibile seguire lo stato degli ordini in corso e passati |  |
| RU017 | D | L’utente può inviare delle segnalazioni riguardo l’ordine al sistema |  |
| RU018 | D | Il gestore gestisce l’avanzamento dello stato degli ordini e le segnalazioni |  |

## analisi funzionale

### Il Data Base

#### Modello ER

##### 

#### Modello EER

##### 

### Corrispondenza dei requisiti Utente

### Tabella 2 Requisiti funzionali

|  |  |  |  |
| --- | --- | --- | --- |
| Requisito utente | Requisito funzionale | Descrizione | Eventuali ipotesi |
| RU001 | RF001 | Deve essere presente un form in cui inserire i dati specifici per un nuovo prodotto, tale form deve avere una parte generica (Con i dati comuni dei prodotti) e una parte specifica per ogni di prodotto. | NA |
| RU002 | RF002 | Le informazioni riguardanti la categoria prodotto devono essere salvate in un’apposita struttura che ne permetta la ricerca |  |
| RU003 | RF003 | La giacenza dei prodotti deve essere memorizzata tramite una tabella che rappresenta le singole copie di quel prodotto (tb stock) in modo tale da poterne tracciare lo stato e gestire la concorrenza. La lista dei prodotti e la pagina di dettaglio di prodotto deve mostrare un “etichetta” con lo stato della giacenza. | … |
| RU004 | RF004 | Nel caso un prodotto non sia più disponibile nella pagina di dettaglio deve essere presente un bottone per essere avvisati nel caso in cui ritorni disponibile. Dovrà esserci anche una pagina in cui poter vedere tutti i prodotti “in osservazione” |  |
| RU005 | RF005 | Quando l’admin aggiunge copie ad un prodotto che non era più presente in stock, il sistema dovrà inviare una mail a tutti gli utenti che hanno sotto osservazione quel prodotto, informandoli che è tornato disponibile, poi verrà rimossa la voce riguardante quell’articolo dalla lista di osservazione di ogni cliente. |  |
| RU006 | RF006 | Nella pagina del checkout dell’ordine bisogna poter inserire la carta di credito desiderata, tramite un menu a tendina si può scegliere una carta già usata oppure aprire un form per inserire gli estremi. |  |
| RU007 | RF007 | Quando viene pagato un ordine con una carta “nuova” i suoi dati vengono salvati nel profilo utente, dove potranno essere visionati. |  |
| RU008 | RF008 | Il profilo utente dovrà avere una sezione per gestire gli indirizzi e le carte. Esse potranno essere aggiunte sono in fase di checkout. | Come la maggioranza dei siti di e-commerce fa |
| RU009 | RF008 | // |  |
| RU010 | RF009 | Dovrà essere presente una pagina di ricerca con un campo di testo in cui inserire delle keyword per trovare il prodotto desiderato, nella pagina dovranno essere presenti anche i filtri per categoria e rilevanza sotto forma di menu a tendina. |  |
| RU011 | RF010 | La pagina di ricerca dovrà presentare i risultati sotto forma di lista, la quale dovrà poter essere ordinata tramite un menu a tendina in cui si specifica il criterio e l’ordine. |  |
| RU12 | RF011 | La piattaforma dovrà avere un form di login per far accedere l’utente (e-mail, password) e un form per la registrazione in cui inserire tutti i suoi dati anagrafici. |  |
| RU12 | RF012 | Nell’header di ogni pagina dovrà essere presente un riferimento al carrello. Anche gli utenti anonimi dovranno poter avere un carrello. Gli articoli si potranno inserire nel carrello tramite un apposito pulsante della pagina di dettaglio del prodotto, potrà anche esserne specificata la quantità. |  |
| RU013 | RF013 | Tramite un apposito form raggiungibile dalla pagina di dettaglio del prodotto, gli utenti potranno lasciare le recensioni sul prodotto. |  |
| RU014 | RF014 | Nella pagina di dettaglio di prodotto dovrà essere possibile consultare le recensioni del prodotto corrente. Nel caso in cui l’utente abbia recensito il prodotto la sua recensione sarà sempre la prima. In questa sezione della pagina dovrà essere presente un menu a tendina per ordinare le recensioni per criterio e direzione. |  |
| RU015 | RF015 | Nella pagina di dettaglio del prodotto vicino al pulsante “aggiungi al carrello” dovrà essere presente un menu a tendina per indicarne le quantità. |  |
| RU016 | RF016 | Dalla pagina di profilo utente dovrà essere possibile visionare gli ordini effettuati e da lì poter accedere alla pagina di dettaglio dell’ordine in cui sarò presente lo stato dell’ordine. |  |
| RU017 | RF017 | Dalla pagina di dettaglio dell’ordine dovrà esserci un bottone per andare nella sezione reclami dell’ordine in cui intraprendere una chat con un referente che possa aiutarlo nel gestire la segnalazione. |  |
| RU018 | RF018 | L’admin avrà a disposizione una  pagina di gestione dell’ordine in cui gestire i reclami e modificare/aggiornare lo stato dell’ordine. |  |

Tutte le validazioni vengono fatte Server-Side, tutti i criteri di lunghezza e required vengono implementati con le funzioni length() delle stringhe e che non siano nulle.

Validazioni usate:

|  |  |
| --- | --- |
| User name | [a-zA-Z]+ |
| User surname | [a-zA-Z]+ |
| User email | filter\_var($this->email, FILTER\_VALIDATE\_EMAIL) |
| User password | (?=^.{8,}$)((?=.\*\d)|(?=.\*\W+))(?![.\n])(?=.\*[A-Z])(?=.\*[a-z]).\*$ |
| Review vote | $this->vote <= 1 && $this->vote <= 4 |
| Review content | strlen($this->content) >= 80 && strlen($this->content) <= 512 |
| PaymentMethod card (VISA) | ^4[0-9]{12}(?:[0-9]{3})?$ |
| PaymentMethod card (Mastercard) | ^5[1-5][0-9]{14}$ |
| PaymentMethod full name | [a-zA-Z]+ |
| PaymentMethod cvc | ^[0-9]{3}$ |
| Address full name | ^[\w\s]{3,}$ |
| Address Street | ^[\w\s-]{5,}$ |
| Address City | ^[\w\s-]{2,}$ |
| Address Region | ^[\w\s-]{2,}$ |
| Address Zip | ^[\w]{4,}$ |
| Address Country | ^[\w\s-]{3,}$ |
| Product code | ^[\w]{4,}$ |
| Product name | [a-zA-Z]+ |
| Product Photo Url | MalformedURLException |
| Product Unit price | >=0.0.1 |
| Product Small description | smallDescription.length() >= 10 && smallDescription.length() <= 300 |
| Product Description | description.length() >= 10 |
| Product Category | category >= 1 && category <= 6 |
| Product Author | ^[\w\s]{3,}$ |
| Product Editor | ^[\w\s]{3,}$ |
| Product ISBN | ^[\w]{10,}$ |
| Product Print Length | >= 1 |
| Product Pubblication Year | ^[0-9]{4,}$ |
| Product Director | ^[\w\s]{3,}$ |
| Product Platform | ^[\w\s]{4,}$ |
| Product Publisher | ^[\w\s]{3,}$ |

## analisi tecnica

### Tecnologia utilizzata

*DBMS:* MySQL 5.7.18

*Client-Side:* Bootstrap v4.1.0, Bootstrap-e-commerce UI kit v4, jQuery 2.0.0, Font Awesome Free 5.0.8 (icone), List.js v1.5.0 (libreria javascript per la gestione di liste)

*Server-Side (Customer)*: Linguaggio: PHP 7.1.7, Deploy: Apache httpd-2.4.27

*Server-Side (Admin):* Linguaggio: Java8, Deploy: Tomcat 9.0.20, Librerie: simple-java-mail-5.1.7.jar, mysql-connector-java-5.1.47.jar, jstl-1.2.jar

### Mappatura dei requisiti funzionali Tecnologia utilizzata

Sia la parte Customer che Admin sono state sviluppate tramite i pattern MVC, 3-Tier, DAO/Repository. Per rappresentare ogni livello si sono usate tipologie di classe apposite: le classi Controller rispondono alle richieste http, le classi Repository/DAO accedono ai dati e fanno le interrogazioni, la parte di view sfrutta il concetto di Template, ovvero, le pagine web vengono costruite tramite un template (php o jsp+jstl) che viene renderizzato dal controller. I Controller java sono delle servlet che invocano i vari metodi tramite il metodo service() che analizza la richiesta e fa la chiamata corrispondente. I controller php vengono invocati tramite degli script nella root directory che creano un istanza del controller e eseguono il metodo doAction() che ispeziona la richesta e invoca i metodi.

#### Requisito funzionale RF001

Sul server Admin è presente un ProductController.java (servlet) che risponde alla richiesta GET alla rotta “/products/add” renderizzando il template “page/product/form.jsp” contenente un form per l’inserimento del prodotto. Questa rotta si aspetta anche un parametro “category” per far visualizzare il form per categoria prodotto corretto: la pagina jsp presenta un controllo che include l’html corretto a seconda della categoria prodotto.

Il ProductController.java sulla rotta “/products/add” ma con metodo POST legge i dati del form generato dalla “add (GET)” per aggiungere un nuovo prodotto. I parametri della request vengono utilizzati per popolare un bean di tipo Product che poi viene validato; in caso di errore viene renderizzato lo stesso form di inserimento precompilato con i dati appena inviati e anche con i relativi errori attribuiti ai campi. Gli errori vengono gestiti tramite un array associativo “error” che per ogni campo contiene i suoi errori. Questo array viene passato dal template che se nota che per un determinato campo sono presenti errori sfrutta la classe .is-invalid di bootstrap per farli visualizzare correttamente.

Nel caso in cui non vi siano errori di validazione del bean allora viene passato ad un ProductDAO.java che inserisce nel DB il bean, sfruttando i dati validati del bean per popolare i parametri della query di inserimento nel db.

Le query vengono generate come PreparedStatment.

#### Requisito funzionale RF002

I dati relativi alla categoria prodotto vengono serializzati come array associativo (Key=>Value) nella colonna “category\_info” della tabella product.

#### Requisito funzionale RF003

Il magazzino viene gestito tramite la tabella stock che contiene le singole compie di un prodotto. La tabella ha una colonna status che ne indica la disponibilità: 0 disponibile, non 0 => non disponibile. Tramite delle query aggregate o dei count si può risalire al conteggio delle unità disponibili in magazzino. La parte model tiene conto di questa informazione aggiuntiva.

Nella lista dei prodotti e nel dettaglio dei prodotti, se il prodotto non è più disponibile viene mostrata sull’immagine del prodotto tramite la classe css. badge-danger un messaggio che dice “not available”; mentre se la disponibilità x è compresa tra 1 e 5 viene mostrata sull’immagine del prodotto tramite la classe css .badge-warning un messaggio che dice “only x copy available”.

L’aggiornamento concorrente della giacenza è gestito tramite un lock sulla tabella stock ogni vota che vengono inseriti prodotti nei carrelli degli utenti.

#### Requisito funzionale RF004

Se il prodotto non è più disponibile il template del prodotto anziché visualizzare il bottone “add to cart” farà visualizzare il bottone “add to watcher list”. Tale bottone è un submit di un form che punta al ProductWatcherController.php con metodo POST inviandogli il product\_id in questione. Questo Controller quindi passera questo product id al ProductWatcherRepository che salverà una nuova row sulla tabella product\_watchers usando come user\_id quello dell’utente in sessione. Nel caso non vi sia nessun utente in sessione, il controller renderizzerà la pagina di login.

Il ProductWatcherController.php al metodo GET se è presente un utente in sessione, tramite la ProductWatcherRepository.php otterrà la lista dei prodotti sotto osservazione dell’utente e li passerà al template page-product-watcher-list.html.php che genererà una lita dei suddetti prodotti.

#### Requisito funzionale RF005

L’admin tramite il ProdcuController.java può aggiungere unità al magazzino di un determinato prodotto. Nel form di aggiunta/modifica è presente anche un input numerico per specificare la quantità di unità da aggiungere allo stock. Il controller verifica quante erano le unità prima dell’incremento; se le unità erano 0 allora tramite il ProductWatcherDAO.java trova tutti gli utenti che hanno quel prodotto nella loro watchlist e per ognuno di essi remove quel prodotto tramite il ProductWatcherDAO.java e invia loro una mail, tramite SMTP server gmail.com con account dell’università, che li informa che il dato prodotto è tornato disponibile.

#### Requisito funzionale RF006

Il CheckoutController.php renderizza la pagina ‘page-checkout.html.php’ che al suo interno contiene una select da cui poter selezionare un metodo di pagamento precedentemente utilizzato. Questi dati sono immagazzinati nella tabella ‘payment\_methods’. Di default viene selezionato l’ultimo utilizzato dall’utente ed è salvato nella colonna ‘default\_payment\_method’ della tabella ‘users’. Per accedere a questi dati si usano l’oggetto utente in sessione e la classe ‘ PaymentMethodRepository.php’. All’interno del menu di selezione è presente una voce “Create New”. Tramite del codice javascript registrato sull’evento onchange della select, se viene selezionata la voce “Create New” (val=”new”) allora viene reso visibile un form di inserimento dei dati della nuova carta di credito da utilizzare. Una volta fatto il submit del form di conferma del Checkout sempre verso il CheckoutController.php, viene controllato il parametro “payment\_method” (nome della select) nel caso in cui sia uguale a “new” viene creata una nuova istanza di oggetto PaymentMethod.php che successivamente viene validata tramite la classe “PaymentService.php” che emula il comportamento di un provider di pagamenti esterno. Se i dati di pagamento non sono corretti (formato del numero, expiry, cvc, full name) allora vengono inviati dei messaggi di errore che poi verranno visualizzati nello stesso form tramite la gestione della visualizzazione degli errori di bootstrap. Nel caso in cui la validazione vada a buon fine, allora il Provider restituirà una stringa che racchiude le info della carta ed un token che servirà per effettuare i pagamenti successivamente; in questo modo non si devono salvare le informazioni sensibili della carta sul nostro DB.

#### Requisito funzionale RF007

Nella pagina del profilo utente generata dal ProfileController.php è presente un collapse contenente le carte di credito registrate dall’utente. Questi dati si ottengono dalla PaymentMethodRepository.php.

Le carte sono rappresentate sotto forma di lista (la lista non mostrerà dati sensibili della carta dato che nel db non sono stati memorizzati), al termine di ogni riga è presente un bottone remove. L’evento onclick del bottone è associato ad una funzione javascript che effettua una chiamata Ajax al ProfileApiController.php con parametro “payment\_method”={id}. L’id è specificato come parametro del metodo. Il parametro corretto viene settato in php nella fase di costruzione della pagina.

Se la richiesta va a buon fine allora quella voce scompare dalla lista ma non dal db: per l’eliminazione di una voce viene utilizzato un soft delete per mantenere la relazione con un ordine passato.

#### Requisito funzionale RF008

La gestione degli indirizzi di spedizione è analoga a quello riportato sopra per i metodi di pagamento. La differenza e che i dati di spedizione vengono immagazzinati sul DB nella tabella “shipping\_addresses” e le validazioni vengono fatte internamente.

#### Requisito funzionale RF009

Nell’header di ogni pagina è presente un form con una barra di ricerca e due select una per la categoria di prodotto e un filtro per nuovi e migliori. La selezione di una voce da questi ultimi due select farà triggerare immediatamente il submit del form. Quest’ultimo punta al ProductController.

Questo form è salvato nel file ‘head-search-form.html.php’ che viene incluso in quasi tutte le pagine.

#### Requisito funzionale RF010

Il ProdiuctListController.php genera una pagina html contenente la lista dei prodotti. La lista dei prodotti viene ottenuta tramite la ProductRepository.php che può filtrare i prodotti tramite un’interrogazione full-text sui campi name, description, category\_info con parametro le keyword del form sopra menzionato, per categoria, controllando la data di inserimento e la media dei voti. Inoltre, si possono effettuare degli ordinamenti per nome, prezzo e voti. Al fine di una migliore presentazione viene gestita anche la paginazione. L’ordinamento si effettua tramite un form nella pagina ‘page-product-list.html.php’ situato all’inizio della lista, che consiste in una select in cui si seleziona il campo con il suo ordine, il form contiene anche tutti i campi hidden necessari per mantenere la ricerca corrente. Allo stesso modo in fondo alla pagina è presente un pager con due bottoni di tipo submit e una serie di bottoni per indicare la pagina.

I prodotti sono disposti a griglia e rappresentati in una card contenente l’immagine, il nome, la small\_description, la media ed il numero di review ed il prezzo. In alcuni casi è presente un’etichetta che indica lo stato nel magazzino. Per vedere i dettagli del prodotto e quindi aggiungerlo al carrello bisogna cliccare sul nome un tag <a> con href verso il ProductController.php.

#### Requisito funzionale RF011

La gestione del log-in e log-out viene effettuata nel seguente modo: è presente un SignInController.php che se viene invocato tramite GET renderizza il template page-sign-in.html.php che contiene un form di login. Il submit del form punta sempre al SignInController via POST. Quando il controller riceve delle credenziali le verifica cercando l’e-mail specificata nel DB se la trova usa la funzione password\_verify () per controllare se la password corrisponde. Se il form viene validato correttamente all’ora viene aggiunto l’utente in sessione e viene impostato un cookie per riottenere l’ultimo utente che si era autenticato anche in caso di chiusura del browser; nel caso in cui il form non fosse stato corretto avrebbe indicato l’errore "Invalid email or password".

Il SignInController.php gestisce anche il log-out che se invocato invalida la sessione corrente e i cookie relativi all’utente (account e carrello).

Per il Sign-Up è presente il SignUpController.php che tramite GET renderizza il template page-sign-up.html.php contenente il form di sign-up. In caso di POST si aspetta i parametri del form appena citato; con i valori nella request crea un oggetto User.php che viene validato. In caso di errore mostrerà i relativi errori dei campi sul form di sign-up tramite gli errori bootstrap. Se, invece, l’iscrizione va a buon fine salva i dati sul DB ed effettua gli stessi passaggi del login.

#### Requisito funzionale RF012 e *RF015*

All’interno del template ‘head-cart.html.php’ viene incluso nella navbar che mostra un’icona del carrello e un badge che indica la quantità di elementi nel carrello. È presente un link su questa icona verso il CartController.php, esso risponda ad una chiamata GET renderizzando il template ‘page-cart.html.php’ il quale mostra il dettaglio del carrello: la lista dei prodotti (thumb, nome, prezzo\_unitario, quantità e totale) e il subtotale calcolato dell’ordine.

I dati vengono memorizzati su DB nella tabella ‘carts’ e relativa ‘cart\_products’. Per poter usare il carrello anche se l’utente non è registrato si sfruttano i cookie e la sessione. All’interno della tabella ‘carts’ e presente una colonna ‘cookie\_cart’ che serve a memorizzare un codice che viene salvato tramite cookie associato ad un particolare carrello. Nel caso in cui l’utente effettui il login con un carrello anonimo riempito verrà effettuato un merge del carrello che l’utente aveva assegnato con quello anonimo.

Per aggiungere un prodotto all’interno del carrello è presente un bottone e una select per indicarne la quantità nella pagina di dettaglio del prodotto. Questo bottone è un submit di un form verso il CartController.php con metodo POST che invia come parametro l’id del prodotto e la quantità. Quando il CartController.php riceve una POST aggiorna i dati del carrello tramite la ‘CartRepository.php’ che presenta un metodo addProduct. L’aggiunta del prodotto al carrello comporta la modifica della tabella stock (modifica unità disponibili e verifica se disponibili) e la tabella cart\_products. Al fine di gestire gli errori di concorrenza nel caso due utenti effettuino un over commitment sullo stock i prodotti vengono aggiunti nel carrello uno ad uno effettuando un lock sulle tabelle stock e product\_cart. In questo mono non vi sono errori di inconsistenza, inoltre i prodotti vengono distribuiti tra gli utenti che li stanno richiedendo in modo equo: Se un utente a ordina 5 prodotti su 5 disponibili e l’utente B fa lo stesso allora questo metodo distribuirà 3 prodotti ad uno e 2 all’altro. Ovviamente questo “cattivo comportamento” viene segnalato all’utente. Infatti, ogni volta che l’aggiunta viene completata si renderizza la pagina ‘page-cart-product-added.html.php’ che informa se l’aggiunta al carrello è andata a buon fine oppure no (facendo visualizzare quanti prodotti ha effettivamente aggiunto). Un’istanza del carrello è sempre presente in sessione ed è allineata con il DB.

#### Requisito funzionale RF013

Nella pagina di dettaglio del prodotto, dopo le info su di esso sono presenti i commenti. Al termine della lista di commenti è presente un bottone “Write a review” che punta al ReviewController.php che risponde ad una GET renderizzando il template ‘page-review-add.html.php’ contenente un form con le “stelline” e una text-area. Il form punta sempre al ReviewController.php con metodo POST che valida i dati ricevuti e li inserisce nel db con la ‘ReviewRepository.php’. Successivamente renderizza il template ‘page-review-added.html.php’ che informa del corretto salvataggio del commento.

#### Requisito funzionale RF014

Nella pagina di dettaglio del prodotto, dopo le info su di esso sono presenti i commenti. La visualizzazione dei commenti viene interamente gestita in javascript. All’interno della pagina viene registrato tramite jQuery il documentLoad. Quando la pagina è caricata quindi viene effettuata una chiamata Ajax verso il ReviewApiController.php che restituisce la lista dei commenti tramite la ReviewRepository.php. La chiamata Ajax invia, oltre il product\_id, altri due parametri l’ordine e il batch. Sopra la sezione dei commenti è presente una select per determinare l’ordine (per voto o per data, asc e desc); anche questa select ha un evento registrato sull’onchange così che quando un utente seleziona un ordinamento la lista viene automaticamente aggiornata. Le review vengono ricevute cinque alla volta dal server e si usa il parametro batch per individuare quante ne ho scaricate. Le review Vengono mostrate sotto forma di lista con autore, voto, data e contenuto.

Per caricare più review è presente un bottone al fondo della lista: “Load more”.

Nel caso in cui un utente abbia inserito una review sull’articolo allora essa viene sempre mostrata come prima della lista. A differenza delle altre, però, presenta due bottoni “Edit review” e Delete che puntano rispettivamente al RevirewController.php con metodo GET (stessa pagina della creazione di una review, solo che prima verifica che la review non sia già presente, nel caso venga trovata allora la mostra all’interno di un form di modifica) e al ReviewApiController.php metodo POST che ricevendo come parametro action=DELETE rimuove dal DB la voce e restituisce la lista aggiornata.

#### Requisito funzionale RF016

L’utente avrà una pagina gestita dal ProfileController.php che quando viene invocato via GET renderizza il template page-profile.html.php. Quest’ultimo file contiene una sezione in cui viene generata la lista degli ordini dell’utente con numero, data, info e status. Cliccando su un numero d’ordine si viene portati all’ OrderController.php che renderizzerà il template ‘page-order.html.php’ contenete il riepilogo dell’ordine (prodotti, costo totale, indirizzo di spedizione, metodo di pagamento e status).

#### Requisito funzionale RF017

All’interno di questa pagina è presente un bottone “OrderReport” che punta al ReportController.php che invocato via GET renderizza il template ‘page-report.html.php’ che mostra una chat con un responsabile. Per scrivere un messaggio di report è presente “fixed” sulla parte bassa della pagina un form per inserire del testo. I messaggi dell’utente vengono posizionati a destra mentre quelli del gestore a sinistra.

#### Requisito funzionale RF018

La parte admin presenta un OrderController.java di base /orders/\*. Questo, alla sotto rotta “/” restituisce la lista degli ordini renderizzando il template “/page/order/list.jsp” passandogli negli attributi la lista degli ordini ottenuta dall’OrderDAO. I risultati sono rappresentati sotto forma di tabella, in cui la ricerca e gli ordinamenti sono implementati via javascript. Ogni voce della tabella presenta il codice, il cliente, lo status, il numero di elementi acquistati, il numero di messaggi di segnalazione non letti, il totale e la data di creazione. Alla fine di ogni riga è presente un bottone “Manage” che punta alla sotto rotta “{id}/manage” dove id è l’order\_id. Nella sezione di manage si presenta il riepilogo dell’ordine più un form con una select per cambiarne lo status. Lo status può essere solo spostato in avanti nel workflow e non in dietro. (To process -> Sent -> Delivered -> Error). All’interno della pagina è anche presente l’input per inserire un eventuale codice di tracciabilità. Gli input si trovano in un form che punta alla stessa sotto rotta della pagina, ma in metodo POST. Anche in questo caso è presente un bottone “Open report (n)” che punta alla sotto trotta “/orders/${order.id}/report/” che mostra una schermata di chat tra un utente e l’admin per la segnalazione di problemi analoga a quella citata per l’utente. Infatti, i template “page/order/report.jsp” e “page-report.html.php” sono pressoché identici.