

Tecnologie WEB

Relazione Progetto
(a.a. 2020-2021)

Alessio Vannella
Matr. 881164

Contents

1	Presentazione del sito	2
1.1	Tema e struttura del sito	2
2	Funzionalità	3
2.1	Login	3
2.2	Registrazione	3
2.3	Logout	3
2.4	Contenuto generato dall'utente	3
3	Caratteristiche	4
3.1	Usabilità	4
3.2	Interazione e animazione	4
3.3	Sessioni	4
3.4	Interrogazioni del database	4
3.5	Validazione dati input	4
3.6	Sicurezza	5
3.7	Presentazione	5
4	Front-end	6
4.1	Separazione presentazione / contenuto / comportamento	6
4.2	Soluzioni cross-platform	6
4.3	Organizzazione file e cartelle di progetto	6
4.3.1	App	6
4.3.2	Public	6
5	Back-end	7
5.1	Architettura generale classi php	7
5.1.1	Core	7
5.1.2	Controllers	7
5.1.3	Models	8
5.1.4	Routing	8
5.2	Schema del DB	9
5.3	Descrizioni delle funzioni remote	11
5.3.1	Esempio di esecuzione	11

1 Presentazione del sito

1.1 Tema e struttura del sito

Ecommerce è un sito di e-commerce che offre prodotti di diverse categorie. Il sito è ispirato ai colossi dell'e-commerce come Amazon e e-Bay.

La navigazione è divisa in questo modo:

- **Homepage:** contiene un semplice slider in cui sono presenti i prodotti più popolari del sito;
- **I miei ordini:** in questa sezione è presente una lista degli ordini passati;
- **La mia wishlist:** in questa sezione è presente la lista dei prodotti salvati nella wishlist;
- **Carrello:** in questa sezione è presente la lista dei prodotti che si è aggiunti nel carrello durante la navigazione;
- **Prodotto:** questa pagina contiene una descrizione del prodotto in cui è possibile interagire per aggiungere il prodotto al carrello o in wishlist. È possibile anche lasciare una recensione per il prodotto con una valutazione che va da 1 a 5 stelle. Infine, è presente uno slider che contiene una lista di prodotti simili a quello che si sta guardando;
- **Ricerca:** in questa pagina, in seguito all'inserimento di una o più keyword all'interno della barra di ricerca, è possibile visualizzare una serie di prodotti ed è possibile filtrarli in base a dei parametri specifici;

Una volta loggati, è sempre presente una navigation bar in alto in cui è possibile ricercare i prodotti tramite keyword e spostarsi all'interno delle varie sezioni, e un footer che anch'esso permette di spostarsi nelle sezioni principali del sito.

2 Funzionalità

2.1 Login

La logica del login è implementata nel controller login.php. In seguito all'inserimento dei dati nel form da parte dell'utente e all'invio dell'evento tramite il click sul bottone, viene fatta una validazione iniziale da parte del javascript, che poi invia una richiesta Ajax al controller.

Successivamente, vengono puliti eventuali input non ammessi tramite la funzione filter_input e viene richiamata la funzione di login che controlla se la coppia username / password esista all'interno del database.

In caso affermativo, verrà creata una nuova sessione nella quale verrà salvato il valore SESSION['id'] e come valore l'id dell'utente appena loggato e l'utente verrà reindirizzato sulla pagina di home.

In caso negativo, verrà visualizzato un messaggio di errore in una navbar in basso a sinistra.

2.2 Registrazione

La logica della registrazione è ancora una volta implementata nel controller login.php. La procedura di invio dei dati è simile alla precedente, verranno validati in un primo momento i dati inseriti da javascript e verranno filtrati tramite la funzione filter_input. A questo punto, verrà richiamata la funzione doRegister che prova ad inserire all'interno del database il nuovo utente.

Se l'operazione va a buon fine (non esiste nessun altro utente con quello username o email), verrà creata una nuova sessione nella quale verrà salvato il valore SESSION['id'] e come valore l'id dell'utente appena registrato, e l'utente verrà reindirizzato sulla pagina di home.

In caso negativo, verrà visualizzato un messaggio di errore in una navbar in basso a sinistra.

2.3 Logout

Se l'utente ha effettuato il login, può disconnettersi effettuando il logout clickando sulla navbar in alto a destra (sotto la sezione "Il mio profilo"). Questa operazione è controllata dal controller logout, che distrugge la sessione con un session_destroy() e ne genera una nuova con un session_start().

2.4 Contenuto generato dall'utente

L'utente ha la possibilità di svolgere diverse azioni:

- aggiungere / rimuovere / aggiornare un prodotto dal carrello;
- aggiungere / rimuovere un prodotto dalla wishlist;
- lasciare una recensione con un commento per un prodotto;
- filtrare la ricerca per un prodotto sulla base di diversi filtri;

Tutte queste azioni sono fatte attraverso l'utilizzo di chiamate Ajax di tipo POST. Dopo aver processato, il risultato verrà restituito a javascript che lo visualizzerà sotto forma di messaggio di testo.

3 Caratteristiche

3.1 Usabilità

Per assicurare un'ottima leggibilità, durante la progettazione è stato deciso di usare testo nero su sfondo bianco. Il sito è dotato di un design minimal in modo da aiutare l'utente a compiere il suo obiettivo nel modo più veloce ed efficiente possibile.

3.2 Interazione e animazione

- **Infinite scrolling:** per ottimizzare la pagina di ricerca, non verrà restituito l'intero pool di risultati per quella ricerca ma solo una sua parte (è presente un valore parametrico che al momento è settato per 10 elementi per volta). Appena l'utente arriverà alla fine della pagina di ricerca, se sono presenti altri risultati per quella ricerca, verranno caricati all'interno del div dei risultati;
- **Filtro:** durante la ricerca, è possibile filtrare i risultati attraverso un menu laterale stick che seguirà l'utente durante lo scroll. È possibile filtrare una ricerca in base ad un range di prezzo, in base alla disponibilità di un prodotto o meno e in base alle recensioni ricevute;
- **Votazione:** per ogni prodotto, è possibile lasciare un voto da 1 a 5 con un commento. Se l'utente non ha ancora votato, nella parte inferiore della sezione delle recensioni è possibile clickare sulla valutazione che si vuole lasciare e in seguito scrivere un commento;
- **Slider:** all'interno della home è presente uno slider in cui verranno visualizzati i prodotti più popolari tra tutti gli utenti (quelli con più visite). All'interno della pagina di ogni prodotto, invece, è presente uno slider di prodotti in cui ci sono i prodotti simili a quello che si sta guardando. Questi slider sono implementati tramite il plugin slicker;

3.3 Sessioni

Nella creazione della pagina, nel file `init.php` viene fatto un `session_start()` per creare una nuova sessione o recuperare una precedente già creata. La session verrà utilizzata principalmente per mantenere l'utente loggato e per evitare agli utenti non loggati di accedere alla piattaforma. Ogni tentativo di connessione ad una pagina diversa da login mentre non si è loggati, risulterà in un reindirizzamento sulla pagina di login.

In seguito al login, viene salvato nella variabile `SESSION['id']`, l'id dell'utente loggato e verrà utilizzato nella web app per identificare l'utente che sta cercando di eseguire determinate azioni (aggiungi a wishlist, aggiungi al carrello, completa ordine, ...).

In seguito al logout di un utente loggato, verrà eseguito un `session_destroy()` e un nuovo `session_start()`

3.4 Interrogazioni del database

Le interrogazioni del database sono tutte presenti all'interno dei file *Model*. Sono stati definiti dei metodi generici *selectQuery*, *insertUpdateQuery* e *deleteQuery* per ogni macro tipo di operazione. Tutte le funzioni fanno il catch di possibili `PDOException` e le inviano al chiamante.

3.5 Validazione dati input

La validazione degli input è stata implementata su due livelli:

- **Livello client:** javascript si assicura che l'input sia coerente con i dati che ci aspettiamo e controlla che eventuali campi richiesti non siano vuoti. Sono stati usati anche gli attributi di HTML5 (required, min, max) per gli input con determinate caratteristiche;
- **Livello server:** Per estrarre i parametri da richieste HTTP (POST), si è utilizzata la funzione `filter_input` di PHP con il flag `FILTER_SANITIZE_FULL_SPECIAL_CHARS`, che filtra l'input in modo robusto. Inoltre, è stata definita una funzione `isValidInput` che controlla che l'input non sia vuoto o null;

3.6 Sicurezza

Come conseguenza della validazione degli input, gli attacchi XSS e SQL injection sono resi vani. Inoltre, le password degli utenti sono salvate sul database con crittografia MD5.

3.7 Presentazione

Il sito è composto da un menu di navigazione e un footer. Il contenuto delle pagine è nella sezione centrale ed è diviso in colonne responsive che puntano sulla semplicità e sulla chiarezza.

4 Front-end

4.1 Separazione presentazione / contenuto / comportamento

È stato utilizzato lo stile unobtrusive per separare il più possibile la presentazione dallo stile e dal comportamento. Tutti gli handler di JS, sono stati aggiunti in maniera totalmente staccata dalla presentazione e caricati durante l'evento ready di jquery. Allo stesso modo, tutti gli stili sono stati aggiunti tramite fogli di stile in file separati.

Inoltre, grazie all'utilizzo del pattern MVC che vedremo in dettaglio successivamente, è stato possibile implementare una divisione chiara delle componenti.

4.2 Soluzioni cross-platform

Il sito è reso completamente responsive grazie all'utilizzo delle classi built-in di bootstrap, che crea un container diviso in colonne responsive (fatto attraverso le media query di CSS3). Il menu di navigazione diventa un hamburger menu quando visualizzato da mobile.

Sono stati fatti ulteriori aggiustamenti manuali per dispositivi di piccoli dimensioni (min-width: 600px) con le media query.

4.3 Organizzazione file e cartelle di progetto

Il progetto è diviso in due sottocartelle principali:

4.3.1 App

All'interno di questa cartella sono presenti i componenti principali logici della web app. Nell'analisi del front-end, possiamo fare il focus sulla cartella *views*, al cui interno sono presenti tutte le viste (HTML + PHP) che verranno poi caricate dai relativi controller. Analizzeremo meglio l'architettura nella prossima sezione.

4.3.2 Public

All'interno della cartella public sono presenti tutti i componenti *client-side* della nostra web-app. Guardiamo nel dettaglio le varie cartelle:

- **css:** contiene tutti i fogli di stile, divisi per controller, della piattaforma;
- **img:** contiene tutte le immagini della piattaforma. In particolare, è presente una cartella *products* al cui interno ci sono, divise per id, le varie immagini di ogni prodotto;
- **js:** contiene tutti gli script javascript, divisi per controller, della piattaforma;

All'interno di public, inoltre, sono presenti dei componenti della pagina che verranno utilizzati in tutte le pagine o quasi (top, banner, bottom).

5 Back-end

5.1 Architettura generale classi php

Il back-end è stato implementato seguendo il pattern MVC. In particolare, è stato utilizzato il paradigma object oriented di PHP per implementare le varie componenti in maniera ottimale. La struttura del codice è la seguente:

5.1.1 Core

Classi per il funzionamento generale.

- **App:** è un wrapper di tutte le componenti che servono a far funzionare la web app;
- **Controller:** modello astratto di un controller, ogni controller concreto dovrà estendere questa classe;
- **Database / Model:** la classe database fornisce un'implementazione del database e fornisce delle funzioni per eseguire le operazioni in sul database in maniera molto generale (select, insert, update e delete);
- **Response:** classe di utility per rendere uniformi le risposte tra i vari layer. Fornisce un codice di errore, una risposta testuale e un array di dati che può essere null nel caso non sia necessario;

5.1.2 Controllers

Ogni controller gestisce una porzione di sito web, visualizzazione o comunicazione con i model corrispondenti alle operazioni richieste. Possono essere chiamati direttamente da ajax attraverso un parametro *op* inviato in POST.

- **Cart:** gestisce la visualizzazione della pagina del carrello e fornisce una serie di funzioni richiamabili da ajax sul carrello (*get, count, add, remove, update, order*);
- **Home:** gestisce la visualizzazione della pagina della home e permette di richiamare *popular*, che restituisce una serie di prodotti popolari, attraverso ajax;
- **Login:** gestisce la visualizzazione della pagina del login e fornisce le funzioni *login* e *register* richiamabili da ajax;
- **Logout:** un controller di comodità che gestisce la disconnessione da parte dell'utente dal sito;
- **My:** gestisce la visualizzazione della pagina "il mio profilo" nella quale, attraverso delle selezioni da parte dell'utente, si può visualizzare sia la lista degli ordini che la lista degli oggetti in wishlist. Inoltre, fornisce una serie di funzioni richiamabili da ajax (*get, order, wish*);
- **Prod:** gestisce la visualizzazione della pagina del prodotto. Prende come parametro GET l'id del prodotto da visualizzare. Inoltre, fornisce una serie di funzioni richiamabili da ajax (*getProd, getReviews, addReview*);
- **Search:** gestisce la visualizzazione della pagina di ricerca e fornisce una serie di funzioni richiamabili da ajax. Sono utilizzate per filtrare i dati in base alla selezione dell'utente e per ritornare i valori utili per l'autocomplete (*getProd, getSearch*);
- **Wish:** fornisce una serie di funzioni chiamabili da ajax per la gestione della wishlist (*add, remove, exists*);

5.1.3 Models

I models forniscono delle funzioni che manipolano il database. Vengono utilizzati i metodi generici di select, insert / update e delete ereditati dalla classe Database.

5.1.4 Routing

Come già accennato in precedenza, la web app è stata sviluppata con un pattern MVC. In particolare, tutte le richieste vengono reindirizzate alla pagina index.html, che genera il contenuto finale attraverso vari passaggi.

- **Import delle librerie:** in un primo momento, viene incluso init.php che andrà a richiamare automaticamente tutte le librerie necessarie per la generazione dei contenuti;
- **Inizializzazione dell'app:** a questo punto, viene creato un oggetto di tipo *App* che andrà a fare il dispatch dei parametri acquisiti tramite GET. In particolare, il primo parametro sarà il controller da richiamare, il secondo parametro sarà il metodo (o action) da invocare e infine ci potrà essere una lista di lunghezza variabile di parametri per l'action specificata;
- **Creazione e invocazione del controller:** l'ultima operazione è quella di invocazione del controller tramite la funzione call_user_func_array, che andrà poi a generare la pagina richiesta dall'utente;

Per rendere più fluido questo meccanismo, si sono definiti dei file .htaccess che modificano la sintassi degli url rendendoli più naturali e leggibili. Ad esempio, se si volesse visualizzare un determinato prodotto (diciamo con id 5), la sintassi di un url sarà

http://127.0.0.1/public/prod/5

invece di

http://127.0.0.1/public/index.php?url=prod/5

5.2 Schema del DB

USERS (ID, name, surname, username, password, email)

PRODUCTS (ID, name, description, category, quantity, price, rating, seller_id)

- PRODUCTS(CATEGORY) referenzia CATEGORY(ID)
- PRODUCTS(SELLER_ID) referenzia SELLERS(ID)

SELLERS (ID, name, surname)

CATEGORIES (ID, category)

VISITED (ID, time, user_id, prod_id)

- VISITED(user_id) referenzia USERS(ID)
- VISITED(prod_id) referenzia PRODUCTS(ID)

REVIEWS (ID, time, user_id, prod_id, vote, review)

- REVIEWS(user_id) referenzia USERS(ID)
- REVIEWS(prod_id) referenzia PRODUCTS(ID)

WISHLIST (ID, user_id, prod_id)

- WISHLIST(user_id) referenzia USERS(ID)
- WISHLIST(prod_id) referenzia PRODUCTS(ID)

CART (ID, user_id, prod_id, cart_quantity)

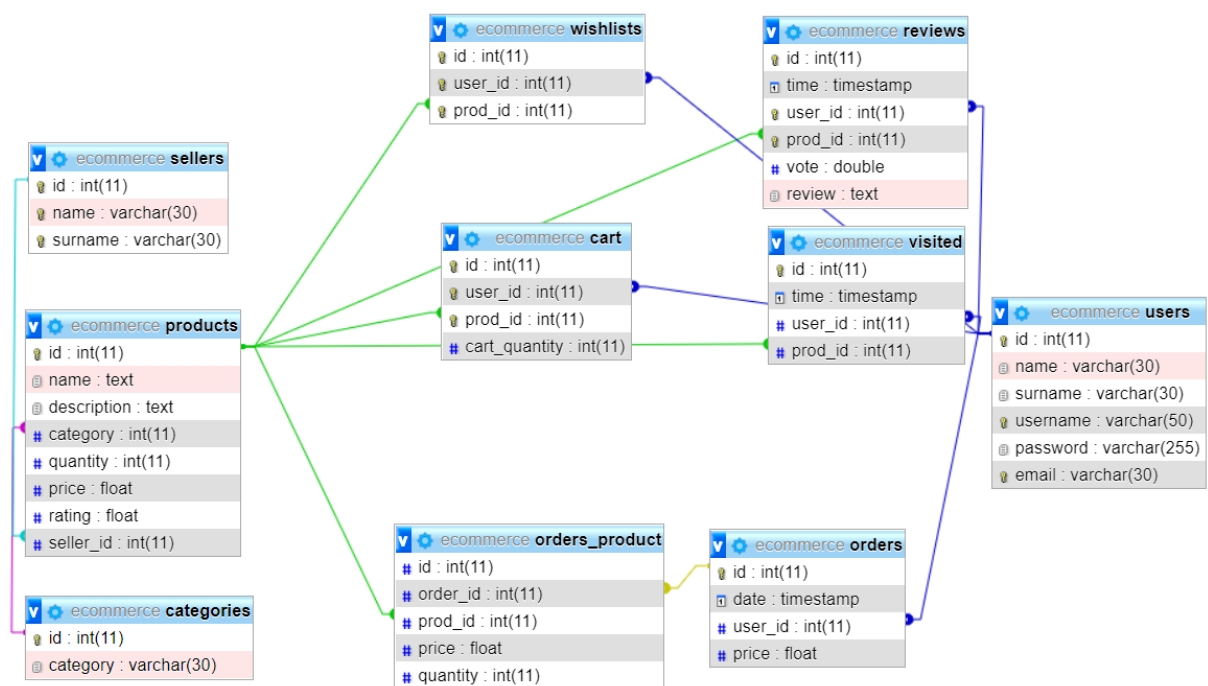
- CART(user_id) referenzia USERS(ID)
- CART(prod_id) referenzia PRODUCTS(ID)

ORDERS (ID, date, user_id, price)

- ORDERS(user_id) referenzia USERS(ID)

ORDER_PRODUCT (ID, order_id, prod_id, price, quantity)

- ORDER_PRODUCT(prod_id) referenzia PRODUCTS(ID)
- ORDER_PRODUCT(user_id) referenzia ORDERS(ID)



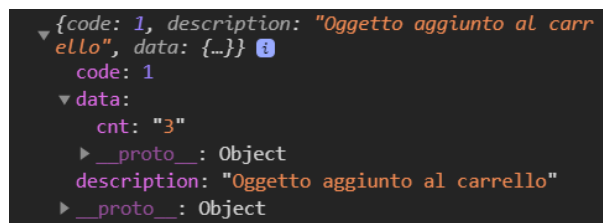
5.3 Descrizioni delle funzioni remote

Per gestire la comunicazione tra Ajax e PHP, viene utilizzata una classe Response che contiene un codice di risposta, una descrizione testuale ed un array eventualmente vuoto di dati che serviranno al chiamante.

5.3.1 Esempio di esecuzione

Immaginiamo di dover aggiungere un prodotto al carrello. Questo verrà fatto tramite una chiamata Ajax al controller cart.

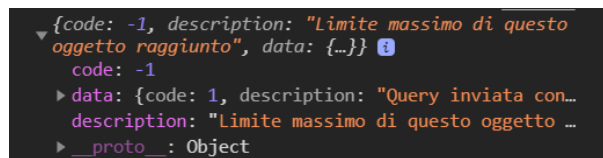
```
1 $.post('cart', {op: 'add', prod_id: 98, quantity: 1}, function(response) {  
2     console.log(response);  
3 }, 'json');
```



```
{code: 1, description: "Oggetto aggiunto al carrello", data: {}}  
  code: 1  
  data: {  
    cnt: "3"  
  }  
    __proto__: Object  
  description: "Oggetto aggiunto al carrello"  
  __proto__: Object
```

Come si può notare, la descrizione dell'output dice "Oggetto aggiunto al carrello" e nei dati è presente il valore "3" che indica il nuovo numero di oggetti presenti nel carrello.

Nel caso in cui provassimo a richiamare la stessa funzione, avremmo un problema perché il prodotto con id 98 ha come campo quantity 1 nel database in questo momento (è disponibile un solo oggetto di quel prodotto). Quindi, il php dopo aver controllato che la disponibilità è minore della quantità richiesta, restituisce un errore significativo



```
{code: -1, description: "Limite massimo di questo oggetto raggiunto", data: {}}  
  code: -1  
  data: {code: 1, description: "Query inviata con..."  
    description: "Limite massimo di questo oggetto ..."  
    __proto__: Object  
  }  
  __proto__: Object
```

Se provassimo, infine ad inviare una richiesta errata (senza specificare la quantità), avremmo un errore diverso

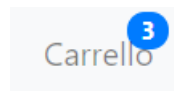
```
1 $.post('cart', {op: 'add', prod_id: 98}, function(response) {  
2     console.log(response);  
3 }, 'json');
```

```
▼ {code: -1, description: "Errore temporaneo", data: Array(0)} ⓘ  
  code: -1  
  data: []  
  description: "Errore temporaneo"  
  __proto__: Object
```

La funzione javascript che gestisce l'aggiunta di un oggetto al carrello utilizza la funzione showNotifyBar per mostrare a video l'output della risposta e la funzione updateBadge per incrementare il badge che indica il numero di prodotti presenti all'interno del carrello

```
/**  
 * add a product to cart  
 * @param {*} prod_id id of the product which will be added  
 * @param {*} quantity quantity of the product  
 */  
function addCart(prod_id, quantity) {  
    $.post('cart', {op: 'add', prod_id : prod_id, quantity: quantity}, function(response) {  
        if(response.code) {  
            updateBadge(response.data['cnt']);  
        }  
        showNotificationBar(response['description']);  
    }, 'json');  
}
```

Codice della funzione di callback



Badge del carrello aggiornato