Immagine che contiene logo, Carattere, design

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente

**Dahi Fatima 152770**

Sito E-Commerce per la vendita e l’acquisto di Libri.

Tecnologie Web 2023/2024

1. Traccia Progetto ………………………………………………………………….. 3
2. Descrizione del Progetto ………………………………………………………. 4
   1. Diagramma UML ………………………………………………………… 4
   2. Diagramma delle classi ……………………………………………….. 5
3. Tecnologie usate …………………………………………………………………. 5
4. Organizzazione logica dell'applicazione …………………………………. 6
5. Scelte implementative …………………………………………………………. 7
   1. Sistema di Ricerca ………………………………………………………. 7
   2. Reccomandation System …………………………………………….. 7
   3. Profilo Utente …………………………………………………………….. 8
   4. Categorie ………………………………………………………………….. 8
   5. Carrello …………………………………………………………………….. 9
   6. Preferiti …………………………………………………………………….. 9
   7. Tipi …………………………………………………………………………… 9
   8. Pagina Home ……………………………………………………………… 9
   9. Messaggi …………………………………………………………………… 9
   10. Recensioni e votazioni ……………………………………………….. 10
6. Test
   1. Views ……………………………………………………………………… 10
   2. Models ……………………………………………………………………. 10
7. Interfaccia Utente …………………………………………………………….. 11

Traccia Progetto

Il progetto sviluppato consiste in un portale di e-commerce per la vendita di libri. Lo scopo è quello di creare un sistema completo e funzionale che consenta agli utenti di acquistare, vendere e interagire con i prodotti disponibili, offrendo un’esperienza integrata per fornitori e acquirenti.

Gli utenti possono registrarsi scegliendo uno dei seguenti ruoli:

1. Fornitore:
   * Può caricare nuovi libri indicando dettagli come titolo, categoria, scheda tecnica, prezzo e immagine di copertina.
   * Riceve notifiche relative agli acquisti dei propri libri e può confermare il pagamento e spedizione effettuata.
2. Acquirente
   * Può visualizzare le recensioni e le votazioni dei libri scritte da altri utenti.
   * Ha la possibilità di confrontare libri simili basandosi su schede tecniche utilizzando tecniche di recommandation system.
   * Può votare e recensire libri, e assegnare una valutazione al fornitore.

Mentre un utente non registrato può navigare sul sito e visualizzare i libri disponibili con il relativo prezzo e le votazioni degli altri utenti, ma non può leggerne le recensioni.

Il progetto è dotato di un sistema di ricerca che permette di selezionare i prodotti in base a diverse caratteristiche:

* Prezzo crescente
* Prezzo decrescente
* Libri più votati

La ricerca dei libri è effettuata in base a tre criteri: nome libro, nome autore e nome categoria.

BookShelf include inoltre un sistema di recoomandation sistem in grado di mostrare ad ogni utente libri che potrebbero interessarli.

Tutti gli utenti, inclusi quelli anonimi, hanno la possibilità di aggiungere i libri nel Carrello, ma solo gli utenti registrati possono confermare l’acquisto durante l’operazione di Checkout. Il Checkout prevede il pagamento tramite bonifico.

**Descrizione del progetto**

Diagramma UML

Come si può vedere dal grafico UML, oltre gli utenti anonimi, gli acquirenti e i fornitori, è presente anche l’admin che è l’amministratore e ha il potere su tutta l’applicazione.

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

**Immagine che contiene testo, diagramma, Parallelo, Carattere

Descrizione generata automaticamente**Diagramma delle Classi UML

**Tecnologie usate**

Django: Django è un framework web open source per il linguaggio di programmazione Python. Viene utilizzato per costruire applicazioni web in modo rapido e semplice.

Squlite: In Django, è il database predefinito per i progetti appena creati. Django è configurato per utilizzare SQLite senza la necessità di installare alcun server di database separato, rendendolo molto facile da utilizzare durante lo sviluppo

JavaScript e Ajax: Invece di ricaricare l'intera pagina ogni volta che un utente aggiunge un prodotto al carrello, il codice che ho scritto JavaScript, invia una richiesta al server (tramite AJAX) e aggiorna solo la parte della pagina che mostra la quantità del carrello, consentendo all'utente di vedere immediatamente l'aggiornamento del carrello senza dover aspettare che la pagina si ricarichi.

Json: Utilizzato per serializzare i dati del carrello (prodotti e quantità) e salvarli nella sessione dell'utente. Questo permette di caricare il carrello precedentemente salvato anche dopo un login.

Bootstrap: una libreria front-end che semplifica la creazione di interfacce utente responsive e moderne. Viene utilizzata per migliorare l'aspetto e la funzionalità dell'applicazione, come nel caso di moduli di registrazione, carrelli e pagine di pagamento.

**Organizzazione del codice**

Per il mio progetto ho creato tre applicazioni separate:

1. **Store**: L’applicazione base. Si occupa della gestione degli utenti (registrazione, login e logout), della ricerca dei prodotti e delle categorie, e della possibilità di aggiungere o modificare recensioni e valutazioni, e per i fornitori la possibilità di aggiungere un prodotto.
2. **Cart**: Gestisce il carrello per l'acquisto dei prodotti, consentendo agli utenti di aggiungere, rimuovere o modificare gli articoli selezionati.
3. **Payment**: Gestisce tutta la logica relativa ai pagamenti.
4. **Notifications**: è una libreria di terze parti per Django che permette di creare un sistema di notifiche avanzato per gli utenti. Utilizza un **modello** per gestire le notifiche. L’ho utilizzata per inviare notifiche personalizzate tra agli utenti dell’applicazione, in particolare tra i Fornitori e gli Acquirenti.

Inizialmente, non avevo previsto una divisione tra **Cart** e **Payment**, ma a metà del lavoro ho deciso di separare le due funzionalità in applicazioni diverse, per avere una struttura più organizzata. Questa suddivisione mi ha permesso di mantenere il codice più chiaro.

All’interno dell’applicazione Cart, si trovano le views che gestiscono il carrello dell’utente: il Cart\_summary, ovvero il carrello con tutti i prodotti che si decide di acquistare, il cart\_add che gestisce la logica di aggiunta del prodotto nel carrello, cart\_delete rimuove il prodotto dal carrello, cart\_update che dà la possibilità di modificare la quantità che si vuole acquistare (1 a 5).

Nella views del Payment abbiamo: il checkout in cui abbiamo il totale da pagare e le informazioni dell’utente, se quest’ultime non sono state aggiunte in precedenza lo si può fare direttamente lì, poi il billing\_info dove è possibile aggiungere i dati bancari per proseguire il pagamento. A seguito del pagamento i dati sono stati salvati nel DataBase e il fornitore, una volta che ha spedito il prodotto, deve notificare l’acquirente, attraverso la vista Messages, in cui appare la data della spedizione quando il fornitore clicca il bottone della spedizione, oltre la data in cui è stato fatto l’ordine.

Per rendere il sistema più ordinato e organizzato, ho deciso di collegare il modello **OrderItem** al modello **Order\_P**.

Il modello **Order\_P** si occupa di salvare gli ordini effettuati da ogni utente, registrando dettagli come la data dell’ordine, la data della spedizione, il totale e l’user associato.

Il modello **OrderItem**, invece, tiene traccia di tutti i prodotti acquistati dagli acquirenti all'interno di ciascun ordine.

In questo modo, per ogni ordine è possibile visualizzare facilmente tutti gli articoli acquistati.

Per implementare questa struttura, ho personalizzato la gestione di questi modelli nel file admin.py

1. Ho creato una classe **OrderItemInline** per rappresentare i dettagli degli articoli dell'ordine in un layout più chiaro, usando StackedInline.
2. Ho esteso il modello Order\_P con una classe personalizzata **OrderAdmin**, che include i campi di sola lettura come "date\_ordered" e integra il collegamento con gli articoli dell'ordine tramite la lista inlines.

Perciò per ogni ordine, ho l'elenco completo di tutti gli articoli acquistati.

**Scelte implementative**

**Sistema di ricerca**

In alto al centro, sopra la navbar, si trova una barra di ricerca. Se non si sceglie nessun tipo di filtro attraverso il menu a tendina, la ricerca si basa sul nome dei libri, il nome degli autori e il nome delle categorie presenti nel sistema.

Il menu a tendina offre 4 tipi di filtri:

* Tutti i tipi = recupera i prodotti che corrispondono a quel termine.
* Prezzo crescente
* Prezzo decrescente
* Libri più votati

**Reccomandation system**

Per i libri consigliati all’utente nella home, ho optato per un sistema di **Collaborative Filtering**. Nella home, per i “Consigliati per te..” ho passato la funziona **recommend\_book\_for\_user()** , che: quando un utente acquista un libro, cerco altri utenti che abbiano acquistato gli stessi libri (o libri simili), e raccomando i libri acquistati da quegli utenti all’utente stesso. Questi libri sono quelli che l'utente non ha ancora acquistato ma che sono stati acquistati da utenti simili.

Mentre nella pagina del prodotto, **product.html**, ho usato un sistema di raccomandazioni ibrido: **Content-Based Filtering**, **Collaborative Filtering** e **Rating-Based Recommendations**:

* Raccomanda libri simili basandosi su attributi come tipo e autore,
* Raccomanda libri che altri utenti, che hanno acquistato lo stesso libro, hanno comprato.
* Suggerisce libri con le migliori recensioni degli utenti.

Alla fine, i libri combinati senza duplicati, vengono passati alla funzione **rank\_books()**

La funzione **rank\_books()** ordina i libri in base alla Popolarità (numero di acquisti per ciascun libro) e Attività recente (numero di acquisti recenti). La combinazione di questi fattori permette di ordinare i libri in modo che quelli più popolari e recenti vengano visualizzati per primi. La funzione assegna dei **pesi** a ciascun criterio per determinare quanto deve influire ciascun fattore nel calcolo finale del ranking.

**Profilo Utente**

Accanto alla barra di ricerca ci sono quattro pulsanti, il primo di essi è il Profilo. Se si clicca il Profilo si apre un menu a tendina, in cui ci sono più opzioni che dipendono se si è autenticati o meno, e se si è l’Acquirente, il Fornitore o l’Admin.

Se si è **l’acquirente** le opzioni sono:

* **Profilo**: che fa accedere alla pagina ‘Profilo’ in cui sono presenti tutte le informazioni dell’utente: Nome Cognome, e-mail, Indirizzo, Città, Codice Postale, Paese, Telefono. Queste opzioni se sono state inserite precedentemente, possono essere modificate cliccando il bottone Modifica, e verranno salvate nel modello **ShippingAddress**, e usate quando si vorrà fare un acquisto, mostrandole nel Checkout.
* **Registrati**: consente la registrazione di un nuovo account. L’utente può scegliere tra due ruoli principali: Acquirente e Fornitore
* Login
* Logout

Se si è il **fornitore** oltre alle opzioni dell’acquirente, è presente l’opzione:

* **Aggiungi Prodotto**: per mettere in vendita i propri libri, aggiungere dettagli nome libro, nome autore, prezzo, immagine, scegliere una categoria esistente o crearne una nuova.

Mentre l’Admin ha accesso a tutte le opzioni.

**Categorie**

Le categorie, come descritte precedentemente, possono essere create durante la fase della registrazione del prodotto e mostrate nel menu a tendina nella navbar.

**Carrello**

L’applicazione ‘Cart’ si occupa di tutta la logica del carrello. Ho utilizzato **AJAX** e **JSON** per migliorare l’esperienza dell'utente. Ogni volta che aggiungo un prodotto nel carrello compare la quantità sopra al bottone ‘Carrello’, riducendo il bisogno di ricaricare l'intera pagina ogni volta che ne si aggiunge uno.

Nel bottone nella pagina del prodotto e nella home ho aggiungo un **value=” {{ product.id }} ”** e **id=”add-cart”**, per poterlo usare nello script di JavaScript. In questo modo, passo l'id del prodotto al mio script, e ciò mi consente di usarlo per gestire le operazioni del carrello.

In quanto i modelli di Django sono pensati per gestire dati che devono essere memorizzati nel database, come Prodotti, User e Ordini, la logica del carrello si occupa di manipolare e gestire dati temporanei, perciò ho creato un file cart.py in cui è presente la classe Cart che gestisce la sessione dell’utente.

**Preferiti**

L’utente ha la possibilità di aggiungere nei Preferiti i libri, e mostrarli in una pagina dedicata. Ho usato la stessa logica del Carrello.

**Tipi**

Ogni volta che i fornitori pubblicano un libro oltre all’aggiunta della categoria, devono specificare il tipo di libro, che si differenziano in: Libri, Bambini e Ragazzi, Manga e Fumetti, Libri Vintage, eBook e Audiobook. Questa divisione l’ho fatta per rendere il sito più organizzato, e dare la possibilità all’utente di cercare i libri in maniera più semplice.

**Pagina Home**

Nella pagina Home è possibile trovare i libri aggiunti dai fornitori. Per ogni libro è presente il nome, il costo e la categoria, ed è anche possibile cliccare il bottone del carrello o dei preferiti. In alto a destra è presente un filtro per libri: è possibile filtrare in base alla data (dalla più recente), al costo (crescente o decrescente) e alle votazioni.

In fondo alla pagina sono presenti gli Hot Trend, i primi cinque libri con la media delle votazione più alta, e accanto ci sono i Consigliati per te.

**Messaggi**

Per poter implementare i messaggi ho usato il pacchetto django-notification-hq.

Questo pacchetto permette di inviare **notifiche personalizzate** agli acquirenti e ai fornitori, in modo che possano essere informati in tempo reale delle azioni che li riguardano, come l’acquisto di un prodotto da parte dei fornitori.

Per poter ricevere le notifiche in tempo reale senza la necessità di ricaricare la pagina, ho usato AJAX e JavaScript. Il codice che ho scritto aggiornare dinamicamente il conteggio delle notifiche non lette, senza la necessità di ricaricare l'intera pagina.

Vicino al bottone dei messaggi vengono mostrate il numero di notifiche non lette.

**Recensioni e Votazioni**

Nel progetto ho anche implementato una funzionalità di recensione e votazione dei prodotti. Questa funzionalità consente agli acquirenti di scrivere, inviare e aggiornare recensioni per i prodotti.

Inoltre, gli acquirenti possono assegnare un rating attraverso un sistema a stelle, dove il punteggio può variare da 1 a 5, con la possibilità di dare anche mezzi voti (ad esempio 3.5 stelle)

L’acquirente può scrivere una sola recensione e dare una sola valutazione al Prodotto, e ha la possibilità di modificare una già esistente.

Per ogni libro, nella scheda tecnica, è possibile vedere la media delle valutazioni.

Nel modello Review, la relazione tra il prodotto e la recensione è definita con una **Foreign Key,** e la cancellazione di un prodotto comporta automaticamente la cancellazione delle recensioni associate. Questo garantisce che le recensioni siano sempre correlate ai prodotti esistenti.

**Test**

Il test l’ho fatto per il Modello Review, e la Views register\_buyer.

**Class ReviewModelTest(TestCase)**

* Test\_review\_creationg(self) = Verifica che la recensione creata sia corretta
* test\_review\_status\_default(self) = Crea una nuova recensione e verifica che lo status sia True dopo la creazione
* test\_review\_association\_with\_product(self) = verifica che la recensione sia associata al prodotto
* test\_review\_association\_with\_user(self) = verifica che la recensione sia associata all’utente -> testuser
* test\_review\_update\_at\_field(self) = controlla che il campo update\_at sia aggiornato automaticamente

**Class RegisterBuyerTestCase(TestCase)**

* test\_register\_buyer\_get(self) = controlla che la view register\_buyer restituisca il modulo vuote e la pagina corretta, che venga usato il template register\_buyer.html e che la pagina contenga il campo username
* test\_register\_buyer\_post\_valid(self) = simula l’invio di dati corretti.
* test\_register\_buyer\_post\_invalid(self) ì= simula l’invio di dati non corretti, esempio username vuoto o email errate.
* Test\_register\_buyer\_post\_existing\_user(self) = controlla, quando un utente inserisce un username che è già presente nel DataBase, che riceva un messaggio di errore

**Interfaccia Utente**

* Immagine che contiene testo, schermata, software, Software multimediale

  Descrizione generata automaticamenteHome
* Immagine che contiene testo, schermata, software, Sito Web

  Descrizione generata automaticamenteSezione Preferiti
* Immagine che contiene testo, schermata, software, Icona del computer

  Descrizione generata automaticamenteFase registrazione user
* Immagine che contiene testo, schermata, software, Icona del computer

  Descrizione generata automaticamenteSezione Messaggi
* Immagine che contiene testo, schermata, software, Icona del computer

  Descrizione generata automaticamenteCarrello
* Immagine che contiene testo, schermata, software, Icona del computer

  Descrizione generata automaticamenteProdotto
* Immagine che contiene testo, schermata, software, Icona del computer

  Descrizione generata automaticamenteRegistrazione con errori