UNIBAY

ARESTA SIMONE MAT: 179981

Progetto Tecnologie Web

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Indice

1 Introduzione

L'applicazione web in questione vuole gestire un E-commerce in stile Ebay in cui perciò è possibile acquistare e vendere prodotti.
L'obiettivo del sito è di emulare al meglio le principali funzionalità del colosso sito di acquisti Ebay, cercando di prendere però le note positive anche dei suoi rivali e concorrenti cercando di rendere il sito funzionale e intuitivo.

L'applicazione è pensata per essere usata da utenti anonimi, utenti registrati, venditori e chiaramente gli amministratori del sito.

1.1 Requisiti

L'applicazione web quindi deve consentire le seguenti azioni ai vari utenti:

1. Gli utenti anonimi possono:
    - Visitare l'e-commerce andando a vedere anche i prodotti specifici
    - Visualizzare per ogni prodotto dei simili
    - Cercare i prodotti tramite un apposita barra di ricerca e/o filtri
    - Visualizzare le recensioni
2. Gli utenti registrati possono:
    - Acquistare un prodotto
    - Salvare un prodotto in una lista preferiti
    - Scrivere una recensione al prodotto e/o al venditore
    - Segnalare un venditore
3. I venditori possono:
    - Mettere prodotti in vendita
    - Visualizzare informazioni sui prodotti messi in vendita e su quelli venduti
    - Visualizzare tutte le recensioni fatte su di essi
4. Gli amministratori hanno il compito di:
    - Gestire le segnalazioni inviate dagli utenti e nel caso avvisare/bannare il venditore

Altre funzionalità:

- Sotto ad un prodotto specifico gli utenti registrati hanno la possibilità di fare domande e il fornitore può rispondere oppure autorizzare le risposte di altri utenti. Ogni qualvolta c'è una risposta, colui che ha posto la domanda riceverà una notifica
- Dotare l'applicazione di un meccanismo di "carrello", quindi un utente registrato può inserire da 1 a N prodotti nel carrello e acquistarli tutti insieme. Per acquistarli è necessario un bonifico e dopo averlo fatto i venditori riceveranno una notifica e potranno far spedire i prodotti. Quando il prodotto viene spedito, l'acquirente riceve una notifica.

2 Descrizione del progetto

2.1 Tipologie di utente

All'interno di UniBay gli utenti sono divisi in:

- Utente anonimo: questa è una tipologia di utente che rappresenta il visitatore casuale della pagina. Le possibilità di questi utenti sono disponibili anche a tutti gli altri utenti. Questa tipologia di utenti può:
  - Registrarsi o effettuare un Log In
  - Visitare l'E-commerce, quindi in generale cercare prodotti tramite appositi filtri
  - Visualizzare prodotti simili al prodotto che si sta visualizzando (ovvero semplicemente prodotti della stessa categoria del prodotto che si sta visualizzando)
  - Visualizzare recensioni e domande sotto ad un prodotto
- Utente Registrato: questa è la tipologia di utente più semplice. Questa tipologia di utenti può:
  - Acquistare prodotti in una quantità arbitraria (chiaramente inferiore alla quantità di prodotto disponibile)
  - Inserire uno o più unità del prodotto in un carrello per poi poterli acquistare tutti insieme
  - Visualizzare informazioni sui prodotti acquistati
  - Salvare i prodotti in una lista di preferiti per poi visualizzare questa lista dal proprio profilo
  - Scrivere, modificare ed eliminare una recensione riguardo ad un prodotto con apposita votazione in stelle da 1 a 5.
  - Scrivere, modificare ed eliminare una recensione riguardo ad un venditore
  - Segnalare un venditore per diverse ragioni (è possibile eseguire una sola segnalazione per venditore), starà poi all'admin capire se punire o meno il venditore in questione
  - Porre una domanda riguardo al prodotto e/o rispondere a domande poste da altri utenti. Se la risposta verrà approvata dal proprietario di quel prodotto, allora la risposta sarà visibile a tutti.
- Venditore: questa tipologia di utente rappresenta un utente che vuole vendere qualcosa sull'E-commerce. Questa tipologia di utenti può:
  - Mettere prodotti in vendita e poterli poi modificare
  - Visualizzare informazioni sui prodotti messi in vendita
  - Visualizzare informazioni sui prodotti venduti indicando quando un prodotto è stato spedito in modo che l'utente che ha acquistato il prodotto ne venga a conoscenza tramite una mail
  - Visualizzare tutte le recensioni fatte dagli utenti su di essi
  - Approvare le risposte fatte dagli utenti sui prodotti di loro proprietà, oppure direttamente rispondere loro
  - Come gli utenti registrati possono acquistare prodotti (non di loro proprietà), metterli in un carrello e avere una lista di preferiti
- Admin: questa è la tipologia di utente che è abilitata ad accedere al pannello amministrativo di Django. Questa tipologia di utenti può:
  - Inviare degli strike ai venditori, dopo 3 strike l'account del venditore è in automatico cancellato, con mail in automatico inviata alla sua posta elettronica
  - Come utenti registrati e venditori possono acquistare prodotti (non di loro proprietà), metterli in un carrello e avere una lista di preferiti

Quanto specificato in questo paragrafo lo si può vedere riassunto nello schema UML che segue.

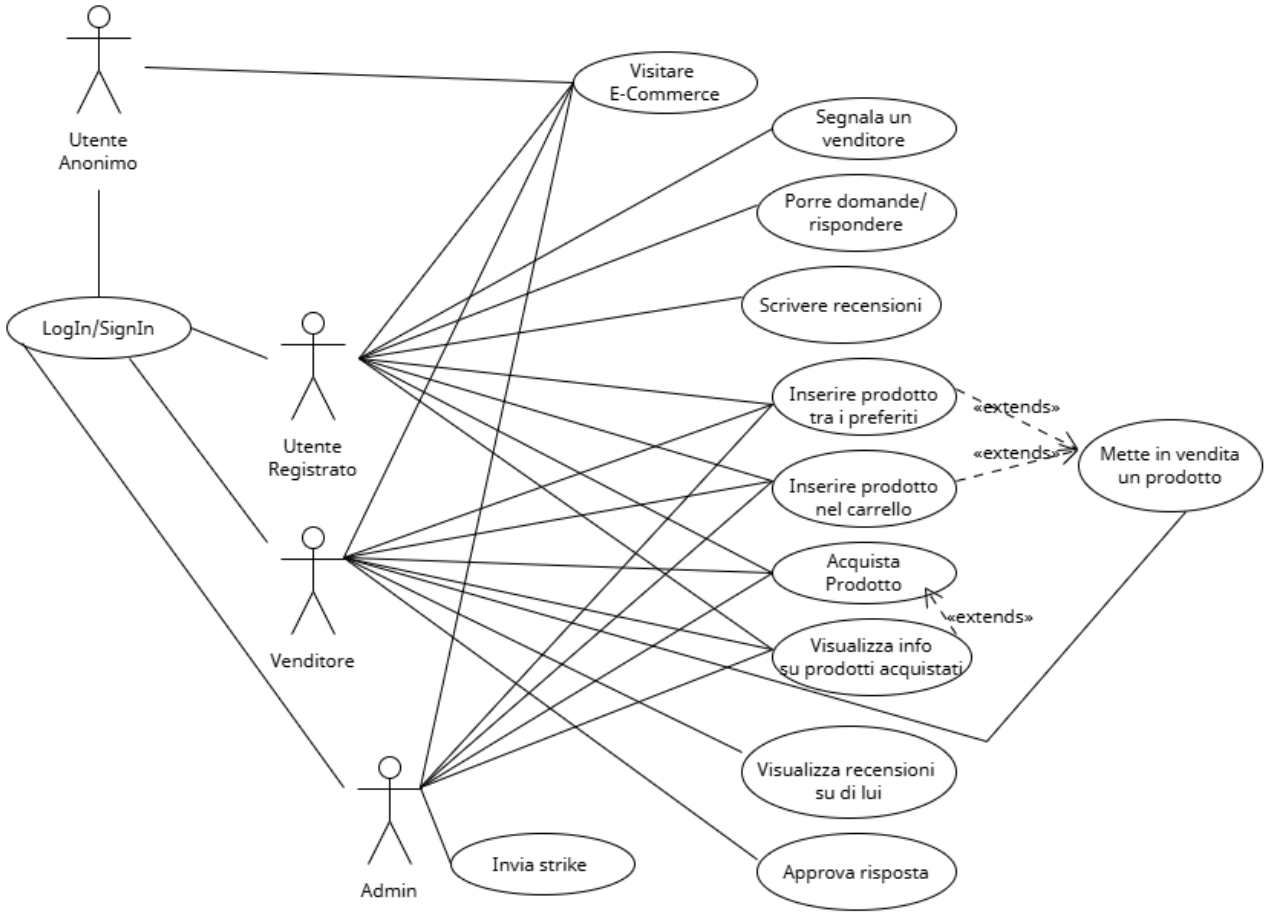

figura 1: UML funzionalità utenti

2.2 Database

Il DBMS scelto per il progetto è stato SQLite per varie ragioni:

- Non richiede l'utilizzo di un server poichè tutto è salvato in un file
- Già integrato nel linguaggio Python
- Semplicità di utilizzo al fine di sviluppare il progetto

Chiaramente comunque nulla vieta in futuro di spostarsi all'utilizzo di altri DBMS.

2.3 Diagramma delle classi

Nelle prossime figure sono mostrati i diagrammi delle classi che cercano di spiegare al meglio le varie scelte effettuate. Si è preferito dividere i diagrammi di classe sulla base degli utenti presenti nell'applicazione web, questo per una maggiore comprensione del funzionamento.

2.3.1 Gestione utenti

Si parte dal diagramma che mostra come sono stati gestiti gli utenti.

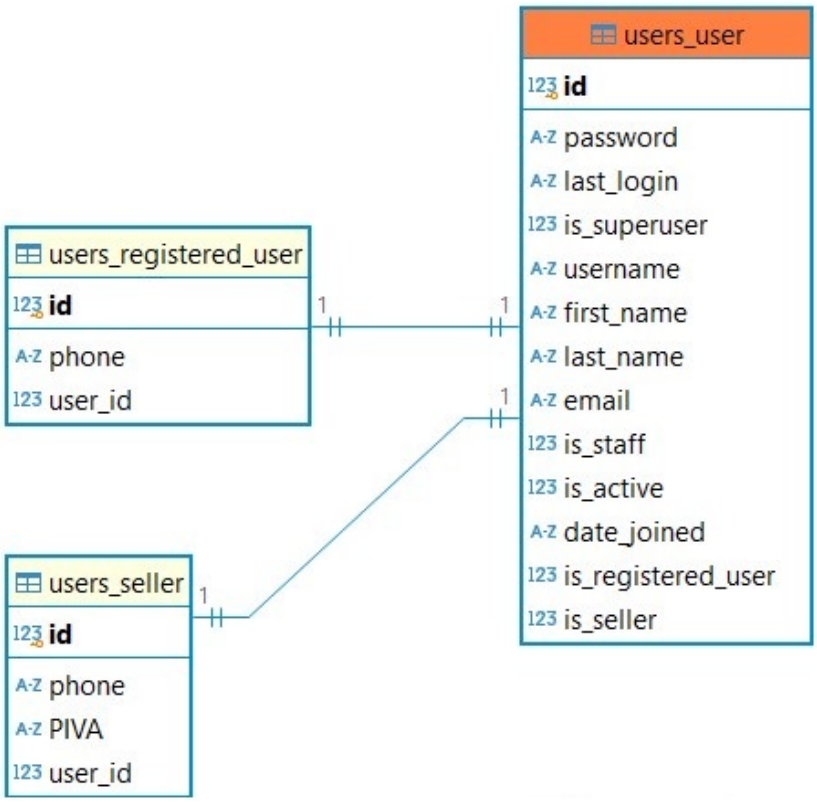

figura 2: UML Diagramma utenti

- User: Il modello user estende il modello *AbstractUser* presente in Django, questo per includere i campi come password, is_superuser o email. Si aggiungono poi due campi aggiuntivi che sono is_registered_user e is_seller, che di default sono impostati a false, ma se collegati a registered_user o seller allora saranno impostati a true.
- Registered_user: Questo modello ha una relazione 1 a 1 con User, e rappresenta il classico utente registrato che naviga sul sito. Avrà il campo is_registered_user impostato a true e poi ha un campo aggiuntivo che è il numero di telefono
- Seller: Questo modello ha una relazione 1 a 1 con User, e rappresenta invece un venditore del sito. Avrà il campo is_seller impostato a true e poi ha due campi aggiuntivi che sono il numeo di telefono e la Partita IVA

2.3.2 User

Verrà mostrato di seguito il diagramma delle classi che rappresenta i modelli collegati a User.

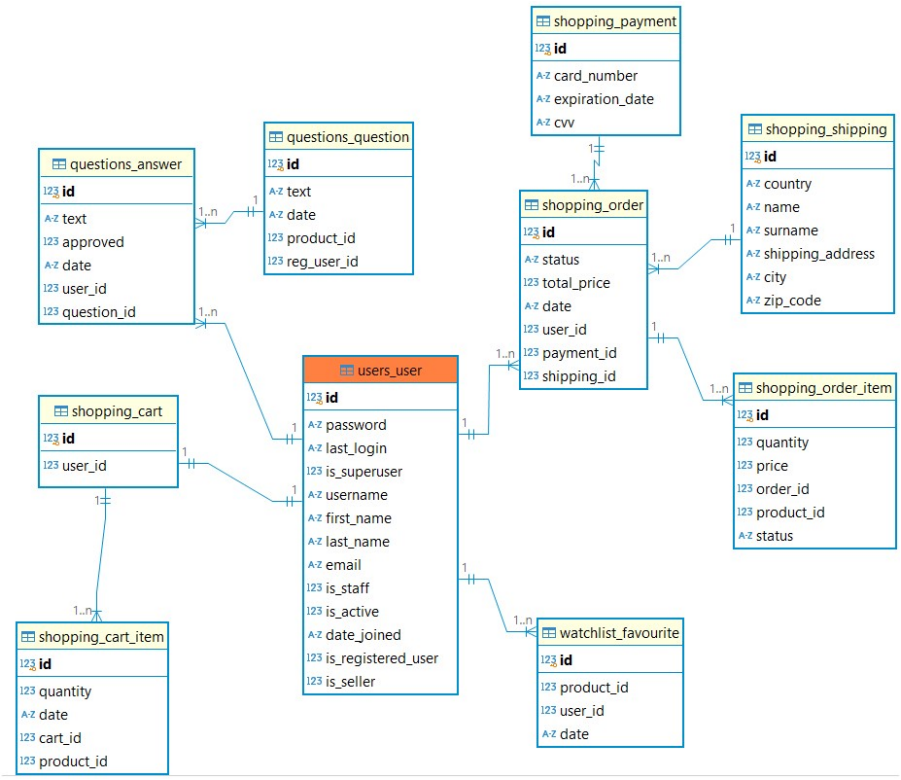

figura 3: UML Diagramma utenti-links

- User: Il modello user rappresenta un qualsiasi utente registrato (utente, venditore, admin). Le azioni possibili da questo utente sono possibili da tutti gli utenti registrati
- Cart: Il modello cart rappresenta il carrello, ha una relazione 1 a 1 con User e rappresenta appunto il carrello che ogni utente ha, dove può inserire i prodotti da comprare. Il cart è a sua volta composto da item
  - Cart_Item: Il modello cart_item rappresenta il prodotto che è inserito all'interno di un carrello. Ha una relazione 1 a n con il cart, quindi chiaramente un carrello può avere più item, e un item appartiene ad un solo carrello. cart_item può presentare anche più unità dello stesso prodotto, mantenuto nell'attributo quantity.
- Favourite: Il modello favourite rappresenta il singolo prodotto inserito in una lista preferiti. Ha una relazione 1 a n con user, quindi chiaramente ogni utente può avere tanti prodotti nella lista preferiti.
- Order: Il modello order rappresenta l'ordine già effettuato dall'utente. Ha una relazione 1 a n con user. In questo modello è stato deciso di aggiungere l'attributo status, che indica con un messaggio testuale lo stato dell'ordine, mente l'attributo total_price indica il prezzo totale dell'ordine.
  - Order_Item: Il modello order_item rappresenta il il prodotto che si trova all'interno dell'ordine. Ha una relazione 1 a n con order. Ha un attributo quantity che rappresenta le unità del singolo prodotto ordinate, un attributo price che indica il prezzo totale di quel prodotto (calcolato come quantity * product.price) e infine un attributo status che indica lo stato di quell'item. Se tutti gli stati degli item di un ordine si aggiornano, allora si aggiorna anche lo status dell'ordine.
  - Payment: Il modello payment rappresenta la carta di credito usata per il pagamento dell'ordine a cui è collegato (o degli ordini a cui è collegato). Come attributi ha le caratteristiche di una carta di credito. Nel database questi dati non sono cifrati, scelta da cambiare nel caso il sito venga messo online.
  - Shipping: Il modello shipping rappresenta l'indirizzo di spedizione di un ordine. Ha come attributi le varie caratteristiche di un indirizzo di spedizione, quali paese, indirizzo, ... Si è deciso di inserire anche Nome e Cognome tra gli attributi perchè può capitare che il destinatario sia diverso dall'acquirente.
- Answer: Il modello answer rappresenta una risposta data da un utente ad una domanda. Ha una relazione 1 a n con user, c'è però da fare una precisazione. Un utente può scrivere tante risposte, ma è stato inserito come vincolo non esprimibile che un utente può inserire una sola risposta ad ogni domanda. Ha come attributi text, che rappresenta il testo della risposta, approved che è un booleano che vale true se è stata approvata dal venditore del prodotto sotto cui è stata posta la domanda, false altrimenti.

- Il modello question rappresenta una domanda posta sotto ad un prodotto. Ha una relazione 1 a n con answer, quindi ogni domanda può avere tante risposte, ma il sito è stato programmato per fare in modo che ci possa essere solo una risposta approvata per ogni domanda.

2.3.3 Registered_User

Verrà mostrato di seguito il diagramma delle classi che rappresenta i modelli collegati a Registered_User

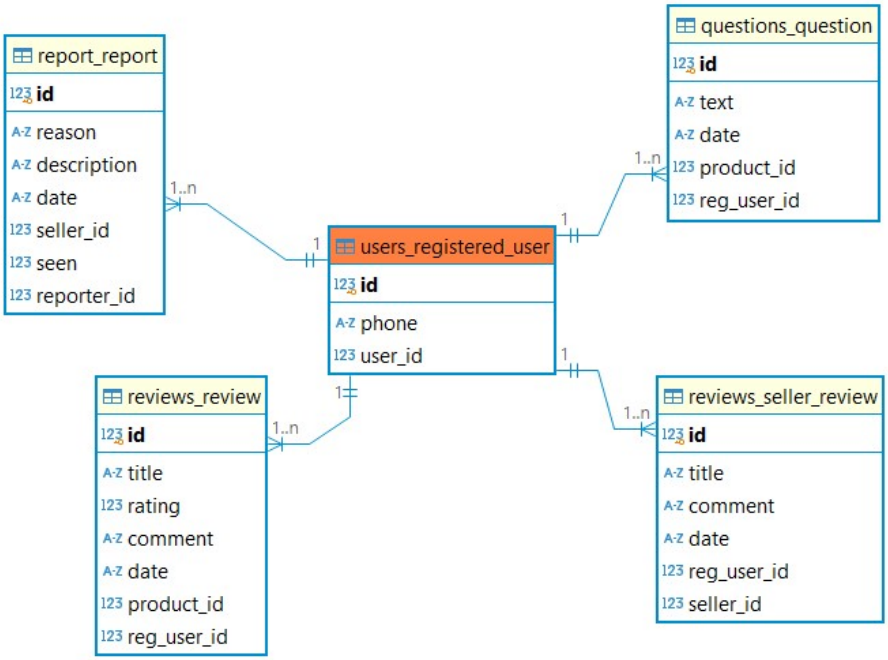

figura 4: UML Diagramma utenti registrati-links

- Registered_User: Il modello registered_user rappresenta un classico utente registrato del sito, che vuole comprare prodotti.
- Report: Il modello report report rappresenta la segnalazione che un utente può inviare riguardo ad un venditore. C'è una relazione 1 a n con registered_user, ma è stato inserito come vincolo non esprimibile che un utente può inviare una sola segnalazione ad ogni venditore. Come attributi sono stati inseriti la ragione della segnalazione, una descrizione e un flag seen, che vale true se un admin ha già visionato il report, false altrimenti
- Review: Il modello review rappresenta la recensione che un utente può lasciare sotto un prodotto. C'è una relazione 1 a n con registered_user ma anche qui è stato inserito come vincolo non esprimibile che un utente può lasciare una sola recensione ad ogni prodotto. Come attributi sono stati inseriti titolo della recensione, valutazione esprimibile come numero intero da 1 a 5 e il commento della recensione.
- Seller_Review: Il modello seller_review rappresenta la recensione che un utente può lasciare per un venditore. C'è una relazione 1 a n con registered_user ma anche qui è stato inserito come vincolo non esprimibile che un utente può lasciare una sola recensione ad ogni venditore. Come attributi sono stati inseriti titolo della recensione e un commento alla recensione.
- Question: Il modello question rappresenta la domanda che un utente può porre riguardo un prodotto. C'è una relazione 1 a n con registered_user quindi ogni utente può porre tante domande anche sotto lo stesso prodotto.

2.3.4 Seller

Verrà mostrato di seguito il diagramma delle classi che rappresenta i modelli collegati a Seller

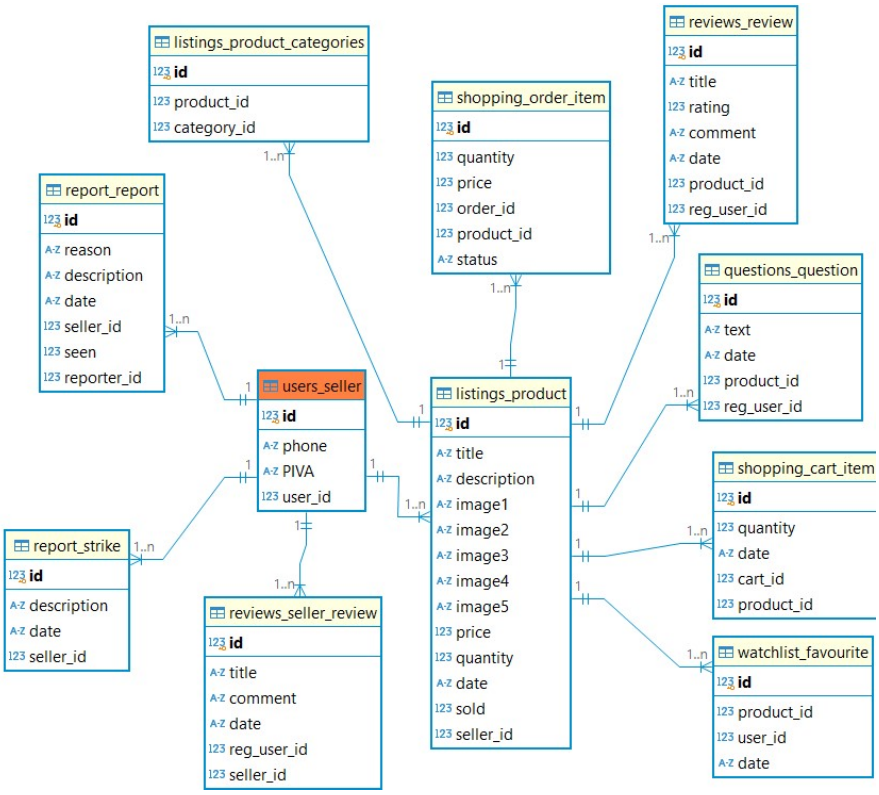

figura 5: UML Diagramma seller-links

- Seller: Il modello seller rappresenta il venditore del sito
- Report: Il modello report rappresenta una segnalazione che un venditore può ricevere da un utente item Strike: Il modello strike rappresenta uno strike che un venditore può ricevere da un admin. Ha una relazione 1 a n con seller, anche all'interno del sito un seller può ricevere fino a 3 strike prima di vedersi il proprio profilo cancellato. Come attributo è stato inserito la descrizione per lo strike
- Seller_Review: Il modello seller_review rappresenta una recensione che un venditore può ricevere da parte di un utente.
- Product: Il modello product rappresenta il prodotto che un venditore può mettere in vendita. Ha una relazione 1 a n con seller, quindi un seller può avere tanti prodotti in vendita. Come attributi sono stati inseriti titolo del prodotto, una descrizione, un prezzo, quantityt che rappresenta la quantità disponibile di quel prodotto, sold che rappresenta la quantità di unità vendute di quel prodotto e infine 5 immagini di cui solo la prima è obbligatoria. Inoltre è presente un metodo *average_rating* che permette di calcolare la media di votazione che quel prodotto ha ricevuto.
    - Category: Il modello category rappresenta la categoria di cui un prodotto fa parte. Ha una relazione 1 a n con product, quindi ogni prodotto può avere tante categorie di cui fa parte. Nel sito sono state inserite delle categorie, e un prodotto può avere come categorie solo le categorie già esistenti.

3 Tecnologie utilizzate

3.1 Frontend

Il frontend è stato sviluppato utilizzando JavaScript, HTML e CSS, in particolare:

- **Bootstrap 5**: Boostrap è stato largamente utilizzato per ampliare il CSS, fornendo una grande quantità di stili e componenti predefiniti. L'utilizzo di Bootstrap ha infatti reso più facile la creazione di layout reattivi e ben progettati, senza doversi preoccupare di scrivere molto codice CSS personalizzato.
- **Crispy forms**: Django-crispy-forms è un'applicazione di Django che permette di controllare il rendering dei Django Form in modo elegante.
- **Font Awesome**: La libreria Font Awesome è una libreria che amplia ulteriormente le possibilità di Django. Esso permette infatti di inserire nella propria applicazione web delle icone (es: stelle per le recensioni, simbolo del cart per il carrello,...) per poter migliorare in generale l'estetica del sito.
- **JavaScript**: JavaScript è stato impiegato anche per il frontend al fine di rendere le pagine più reattive e dinamiche e quindi evitare di caricare ulteriori pagine ad ogni azione.

3.2 Backend

Il backend è stato realizzato utilizzando principalmente le Django Views , le quali gestiscono la logica delle richieste HTTP e restituiscono le risposte tramite il Django Template Engine (DTL) sottoforma di HTML.
Spesso poi si associa Django a JavaScript in particolare tramite l'uso delle **Fetch API** di JavaScript. Per eseguire infatti operazioni asincrone, evitando il caricamento completa della pagina, JavaScript invia richieste HTTP al Server Django, utilizzando il metodo *fetch()*. In questo caso dal lato di Django, ci sono delle views apposite che gestitscono queste richieste *fetch()*, elaborano i dati e restituiscono una risposta in formato JSON.

Per la gestione dei permessi sull'applicazione web è stato utilizzata la classe *LoginRequiredMixin* offerta da *django.contrib.auth.mixins*. Questo però non era sufficiente per l'applicazione, essendo questa dotata di vari utenti, per questa ragione è stato creato il file *mixins.py* all'interno dell'app *users* che crea delle classi ad hoc per gestire l'accesso alle pagine del sito web e soprattutto per evitare la creazione di codice duplicato.
Inoltre, alle views create allo scopo di essere utilizzate dai codici JavaScript, si è deciso di utilizzare il decoratore *require_POST* offerto da *django.views.decorators.http*, questo per evitare che determinate views vengano usate in modo improprio.

Un'ultima nota, l'applicazione fa uso di invio di mail, per garantire l'invio delle mail in determinate situazioni si usa la classe *send_mail* offerta da *django.core.mail*. Questa permette di inviare mail dovendo però precisare un indirizzo mail mittente. I casi in cui le mail sono inviate dal sito, si è deciso di usare la mail del creatore del sito, scelta poi da cambiare nel caso in cui il sito adotti un dominio reale.

4 Organizzazione del codice

Il progetto è stato diviso in 8 applicazioni, ognuna per gestire un diverso aspetto del sito.
Prima di visionarli meglio una precisazione.
All'interno di templates si possono trovare elementi di default delle varie pagine. Nello specifico *base.html* è una base per le pagine che verranno create, che è stata messa in combinazione poi con *head.html* che invece contiene solo il logo della pagina (il quale è un link all'homepage, mentre *base_pages.html* è anche una base per le pagine che verranno create, ma oltre a ciò contiene tutta l'intestazione con la barra di ricerca. Si è deciso di crearne due diverse per non avere duplicazioni di codice e perchè non in tutte le pagine, avere la barra di ricerca è utile.

```
ecommerce/
|- ecommerce/
|    |- ...
|    |- settings.py
|- templates/
|- static/
|    |- css/
|    |- imgs/
|    |- js/
|- listings/
|- pages/
|- question/
|- report/
|- reviews/
|- shopping/
|- users/
|- watchlist/
|- manage.py
```

- **Pages**: Questa applicazione ha più un utilità di "container". Non ha modelli, ha bensì solo l'homepage e la pagina di ricerca. Nello specifico si è deciso di creare una pagina parziale a parte che contenesse gli elementi cercati, per fare in modo che se un utente decidesse di ordinare gli elementi secondo un certo ordine, non sia necessario ricaricare tutta la pagina, ma solo quella parziale. Per una questione di debug in questo momento se si decide di effettuare una ricerca senza chiavi di ricerca, verranno cercati tutti i prodotti. Nel caso il sito venga messo online, questa cosa è da modificare per evitare che il sito debba caricare molti prodotti
- **Users**: Questa applicazione contiene la definizione dei modelli degli utenti che possono interagire con l'applicazione. Come già visionato, si hanno 3 classi di utenti ovvero *User*, *Registered_User* e *Seller*. Gli ultimi due hanno una relazione 1 a 1 con User. Nel caso un User sia un admin, allora avrà impostati a False i due flag *is_registered_user* e *is_seller*, impostando a True i flag *is_superuser* e *is_staff*. La classe Users poi mantiene un duplice scopo:
  1. Gestire signin/login/logout degli utenti sfruttando i meccanismi già presenti in Django
  2. Tutte le azioni che potranno compiere tutti gli utenti registrati, saranno collegati a users

  L'applicazione in questione contiene poi al suo interno i template relativi alla gestione di un account.
- **Listings**: Questa applicazione contiene la definizione dei modelli *Product* e *Category*. Qui si è deciso che ogni prodotto possa avere da un minimo di 1 ad un massimo di 5 immagini e inoltre queste saranno contenute in una directory creata apposta all'interno dell'app listings. Ha al suo interno i template per gestire le operazioni CRUD relativi ai prodotti.
- **Reviews**: Questa applicazione contiene la definizione dei modelli *Review* e *Seller_Review*. Si è deciso qui di dare la possibilità solo ai *registered_user* di poter recensire prodotti e venditori, questo per evitare che un venditore, con quell'account, possa effetuare azioni da utente incentivando maggiormente l'uso dell'account a solo scopo di vendita di prodotti. Nonostante ciò può visualizzare (così come un utente anonimo) tutte le recensioni. Da questa applicazione si gestiscono le operazioni CRUD relativi alle recensioni.
- **Watchlist**: Questa applicazione contiene la definizione del modello *Favourite*. Da questa applicazione si gestisce la visualizzazione dei preferiti, ma parte della logica si trova nel file javascript *favourite.js*
- **Report**: Questa applicazione contiene la definizione dei modelli *Report* e *Strike*. Da questa applicazione si gestiscono le operazioni CRUD relativi ai report (tranne l'aggiornamento, si è deciso di non poter modificare le segnalazioni effettuate). Solo i *registered_user* possono inviare segnalazioni a venditori e un admin può visionare tutte le segnalazioni. Può decidere se inviare quindi uno (o più) strike ad un venditore a causa di una determinata segnalazione e/o non fare niente e spuntarla come visionata.
- **Shopping**: Questa applicazione contiene la definizione dei modelli *Cart*, *Cart_Item*, *Order*, *Order_Item*, *Payment* e *Shipping*. Da questa applicazione si gestisce in generale l'acquisto dei prodotti, la gestione del carrello e la visualizzazione dei prodotti acquistati.
  Si è deciso qui di dividere i modelli Cart e Order sia perchè logicamente sono due cose diverse ma anche perchè in questa maniera, anche avendo un carrello con dei prodotti, è possibile comunque acquistare un

prodotto singolarmente. Si è deciso poi di rendere Payment e Shipping due modelli a parte rispetto all'order e questo per permettere una maggiore scalabilità del sito e dunque per non dover salvare in database duplicazioni di carte di credito o indirizzi di spedizione.
Inoltre essendo che un utente può acquistare prodotti anche da venditori diversi, ogni venditore può segnare quel prodotto come spedito/consegnato, ma ciò non significa che tutto l'ordine è spedito/consegnato.

- **Question**: Questa applicazione contiene la definizione dei modelli *Question* e *Answer*. Questa applicazione gestisce il sistema di domande di un prodotto. Infatti è possibile che un *registered_user* inserica un domanda ma che ogni *user* possa rispondere. Il venditore del prodotto in questione poi può, approvare una risposta per poterla quindi rendere visibile oppure scrivere egli stesso una risposta che sarà automaticamente approvata.

5 Scelte di sviluppo

Si è potuto notare un errore di sicurezza, ovvero che un utente potesse andare nella pagina di Checkout anche senza avere realmente un prodotto da inserire nel checkout oppure con una quantità che non rispettasse quella massima del prodotto.
Per poter contenere il problema si deciso di aggiungere un token che viene aggiunto all'URL nel momento in cui viene premuto il tasto *Acquista ora* su un prodotto. Facendo così l'URL generato contiene un token univoco per la sessione. Attenzione che facendo così il problema non è del tutto risolto perchè chiaramente un utente registrato può comunque copiare quel token e riutilizzarlo, ma si disincentiva comunque l'utilizzo di questa tecnica.

6 Test

Per effettuare i test sul backend di UniBay è stata utilizzata la classe TestCase fornita da Django. Sono stati effettuati test generali su tutto il sito, e in particolare per ogni app, ognuna presente nel proprio file *tests.py*.
Per i test è stato largamente usato il metodo tearDown(), infatti per molti test è stato necessario creare all'interno del metodo setUp() dei prodotti, i quali u Si mostreranno solo i test più significativi

6.1 Users

- ProfileViewTests:
  1. deny_profile_detail: verifica che solo gli utenti che hanno effettuato il login, possono accedere alla propria pagina profilo

6.2 Listings

- ProductModelTests:
  1. test_delete_product_folder: verifica che all'eliminazione di un prodotto venga eliminata anche la cartella associata con le immagini all'interno
  2. test_average_rating: verifica che effettivamente con le recensioni, il metodo per calcolare la valutazione media funziona

6.3 Reviews

- ReviewModelTests:
  1. test_review_unique_constraint: verifica che effettivamente un utente possa recensire solo una volta un prodotto

6.4 Watchlist

- FavouriteModelTest:
  1. test_favourite_unique_constraint: verifica che un utente possa aggiungere un prodotto tra i preferiti una sola volta

- ToggleFavoriteTests:
  1. test_add_to_favourites: verifica il funzionamento del javascript che aggiunge un prodotto tra i preferiti
  2. test_remove_to_favourites: verifica il funzionamento del javascript che toglie un prodotto tra i preferiti

6.5 Report

- StrikeCreateTests:
  1. test_strike_create_view: verifica la creazione di uno strike a partire da un report
- MarkReportSeenTests:
  1. test_mark_report_seen: verifica che un report sia riportato come visto in modo corretto

6.6 Shopping

- CartModelTests:
  1. test_cart_total_price: verifica che dati vari prodotti nel carrello, questo abbia il corretto prezzo totale
  2. test_cart_total_items: verifica che dati vari prodotti nel carrello, questo abbia il corretto numero di unità dei prodotti
- OrderModelTests:
  1. test_item_shipped: verifica che un item cambi correttamente lo stato di spedizione
- CheckoutTests:
  1. test_redirect_if_not_logged_in: verifica che un utente anonimo non possa passare alla pagina di checkout
  2. test_checkout_view_invalid_token: verifica che un token invalido utilizzato per andare nella pagina di checkout, non funziona per andare nella pagina di checkout
  3. test_checkout_valid_token: verifica che con l'utilizzo di un token valido, un utente può andare nella pagina di checkout
  4. test_checkout_valid_forms: verifica che inviando correttamente i forms per il pagamento e l'indirizzo di spedizione, venga poi inviata la mail ai venditori dell'ordine effettuato

6.7 Question

- AddAnswerTests
  1. test_approve_answer: verifica che una risposta venga correttamente approvata

7 Risultati

Homepage

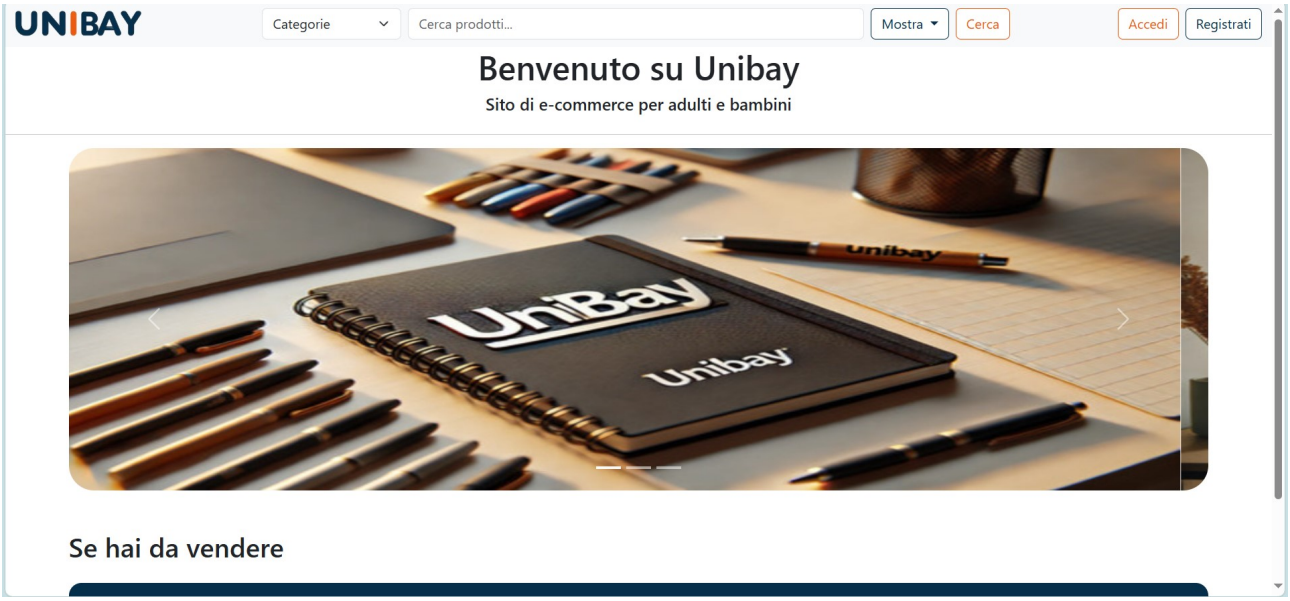

figura 6: Homepage

Profilo utente

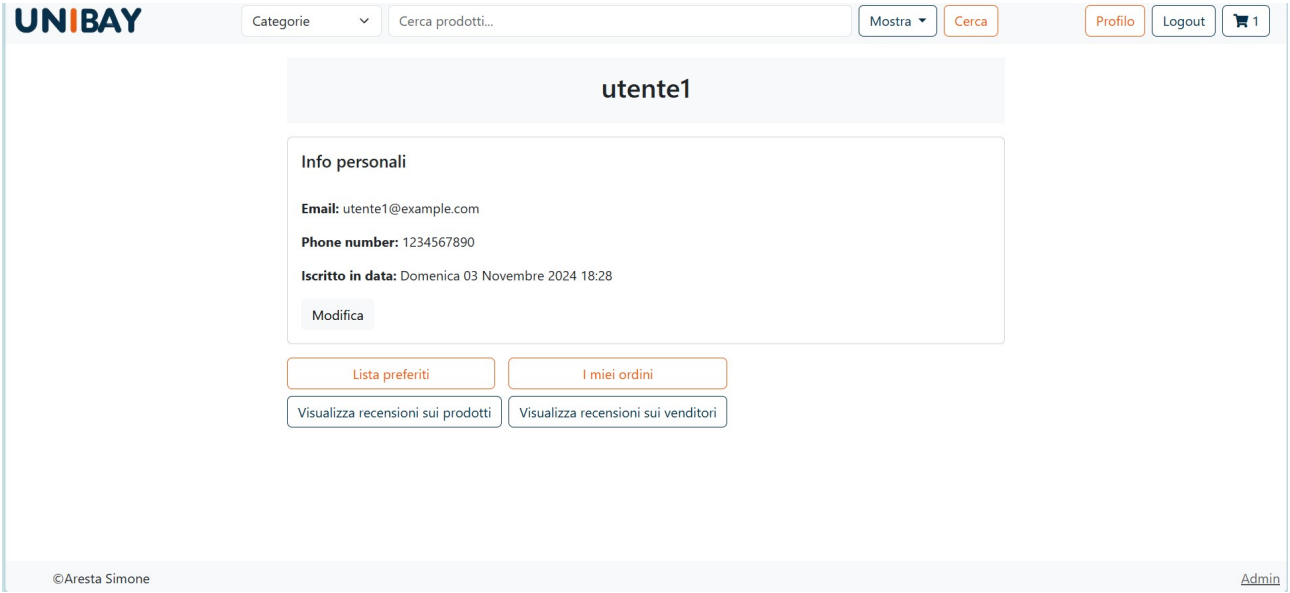

figura 7: Profilo utente

Carrello

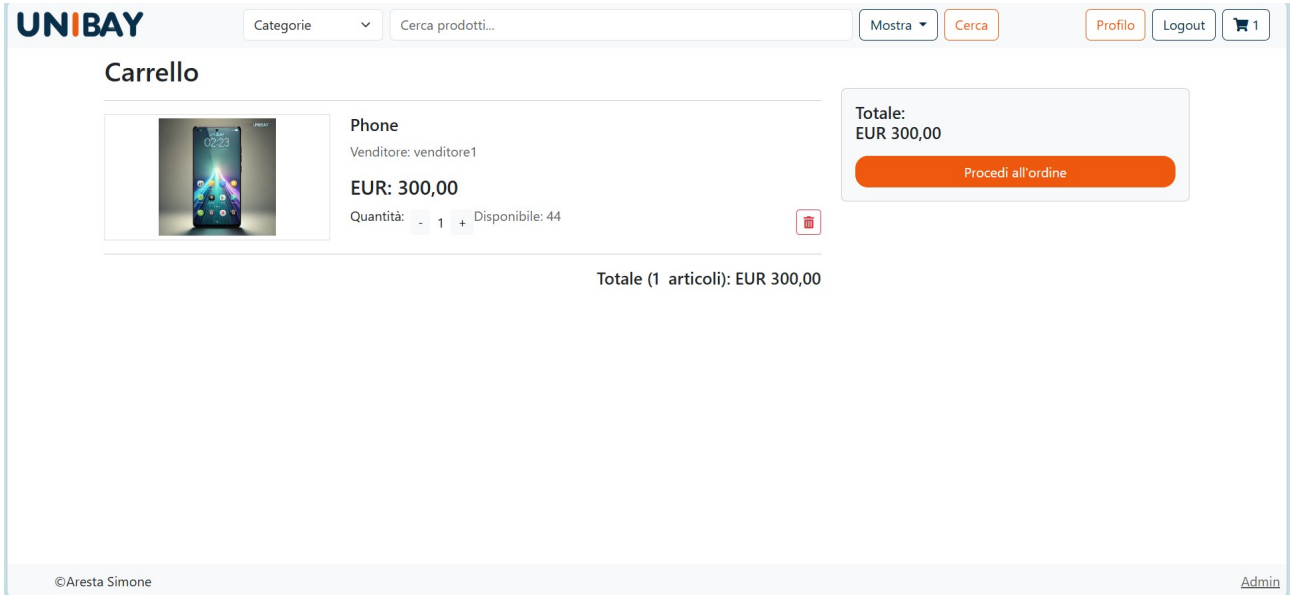

figura 8: Carrello

Pagina di un prodotto

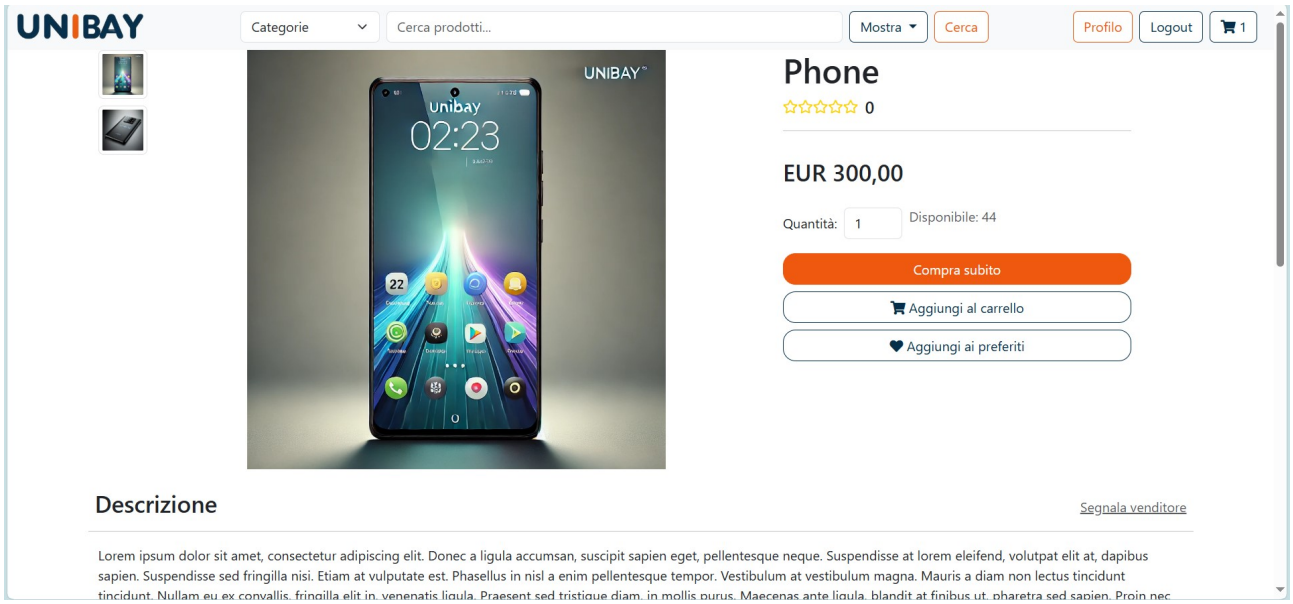

figura 9: Pagina di un prodotto pt. 1

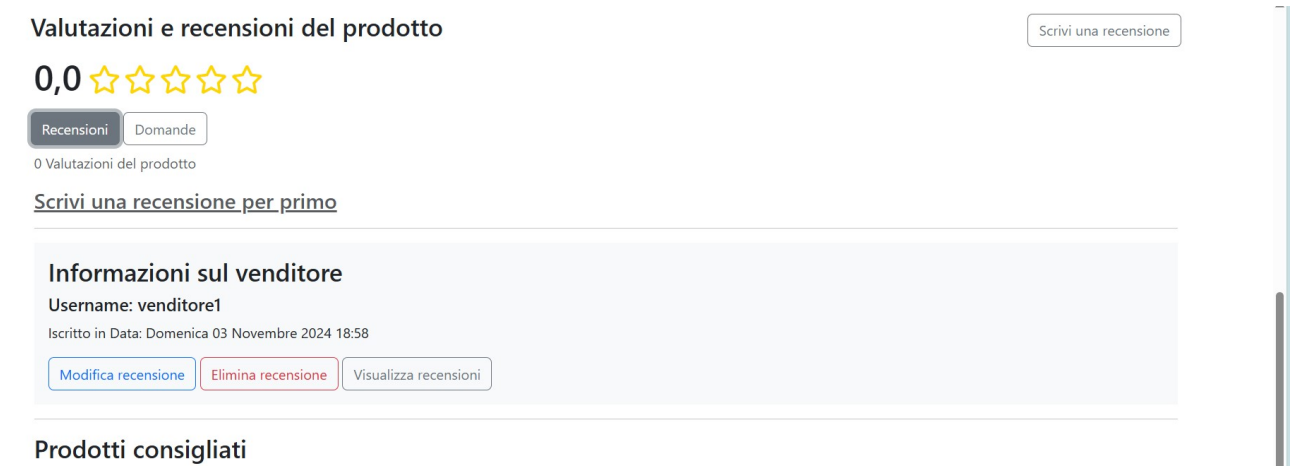

figura 10: Pagina di un prodotto pt. 2

Ordine

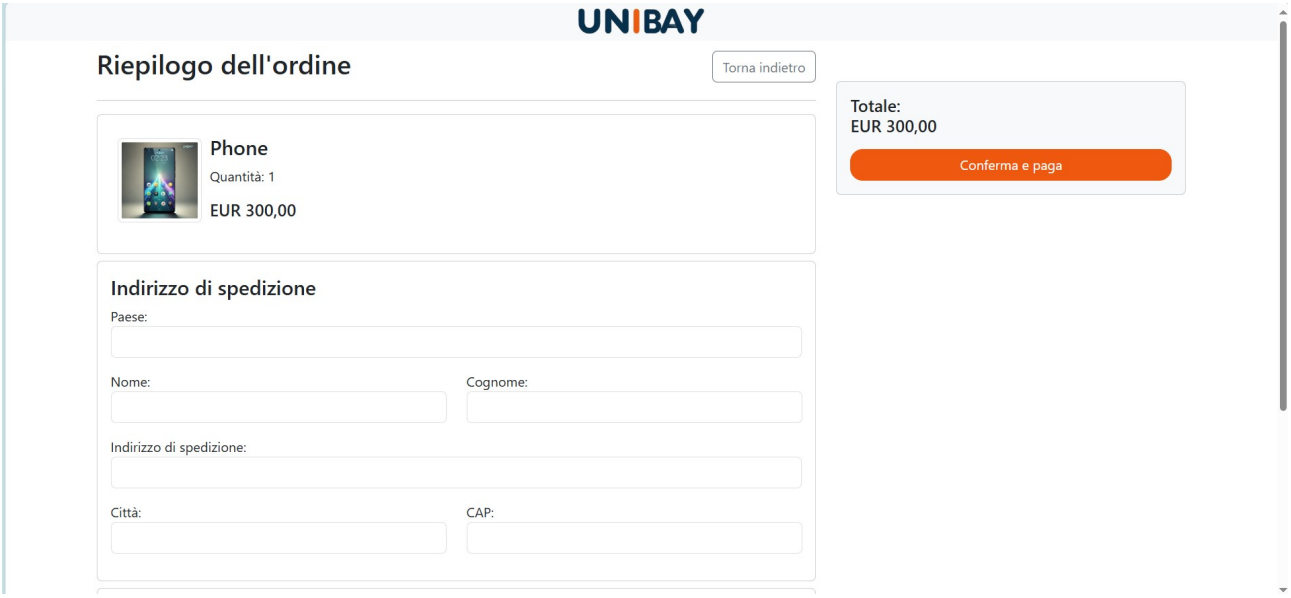

figura 11: Pagina di acquisto di un prodotto

8 Problemi riscontrati

I problemi riscontrati nello svolgimento del progetto sono stati legati principalmente alla mia mancanza di esperienza nell'utilizzo di HTML,CSS ma in particolar modo di JavaScript, per il quale ho dovuto faticare a capire il funzionamento.
Nel pratico del progetto poi la parte più problematica è stata sicuramente la gestione delle immagini, che essendo in generale di grandezze diverse, non era banale cercare di dimensionarle esattamente come volevo.
Nonostante tutto comunque sono soddisfatto del risultato finale sia perché l'applicazione funziona come dovrebbe (anche se dal punto di vista della sicurezza ci sarebbe da migliorare), sia perché ho imparato ad utilizzare Django ed in generale a programmare un sito web.

TUTTE LE IMMAGINI USATE NEL SITO SONO STATE GENERATE DALL'IA