Progetto di Basi di Dati anno 2023/2024

Membri del gruppo:
KRISTI NEZHAJ 887560
FEDERICO GAUDENZI 894069
AVORNIC CRISTINA-ECATERINA 883711

Nome dell'applicazione: TradeTrove

Indice

1. Introduzione

Obiettivo del progetto: TradeTrove è una piattaforma di e-commerce progettata per l'acquisto e la vendita di prodotti online. L'obiettivo principale è fornire un'interfaccia user-friendly che permetta agli utenti di navigare, cercare e gestire i loro acquisti in modo semplice e intuitivo. Inoltre, offre ai venditori gli strumenti necessari per gestire il proprio inventario e interagire con i clienti.

2. Funzionalità principali

Gestione degli utenti: TradeTrove offre un sistema di registrazione e gestione dei profili progettato per garantire un'esperienza utente semplice e sicura. Durante il processo di registrazione, agli utenti viene richiesto di fornire informazioni fondamentali, come nome e cognome. Successivamente, nella sezione del profilo, è necessario inserire l'indirizzo completo per permettere la corretta spedizione dei prodotti acquistati. I venditori, oltre a questi dati, devono fornire il nome dell'azienda e la partita IVA, essenziali per la gestione dei guadagni e delle transazioni aziendali. Per garantire l'unicità degli utenti e prevenire conflitti, è obbligatorio fornire un indirizzo email valido.
Una volta registrati, gli acquirenti possono facilmente visualizzare e gestire i prodotti nel proprio carrello, procedere al pagamento e monitorare lo stato dei propri ordini. Hanno anche l'opportunità di lasciare recensioni sui prodotti acquistati. D'altra parte, i venditori gestiscono i propri prodotti e le ordinazioni direttamente dal loro profilo utente, mantenendo separati i dati di vendita da quelli degli acquirenti.
Entrambi i ruoli, sia l'acquirente che il venditore, hanno la possibilità di cambiare la propria password direttamente dal profilo, garantendo così un ulteriore livello di sicurezza.

Gestione dei prodotti: I venditori possono aggiungere, eliminare e aggiornare i propri prodotti attraverso un form accessibile dalle impostazioni del profilo venditore, per eliminare il prodotto sarà necessario reinserire i dati del prodotto per eliminarlo dalla vendita.
Il prodotto verrà ritirato nel caso la sua quantità scenda a 0.
Il cliente può aggiungere il proprio prodotto al carrello, recensire il prodotto selezionando le stelle che vuole assegnare e cliccare il pulsante, e può visualizzare le informazioni utili del prodotto, cioè nome, descrizione e prezzo.

Ricerca e filtri: La ricerca dei prodotti è presente nel sito attraverso una search-bar disponibile nell’header del sito.
I filtri vengono successivamente mostrati alla ricerca e comprendono categoria prezzo e valutazioni in ordine crescente oppure decrescente.

Carrello della spesa: Il carrello è facilmente accessibile all'utente tramite un'apposita sezione nell'header del sito, rendendolo disponibile in ogni momento durante la navigazione. All'interno del carrello, l'utente ha la possibilità di gestire i prodotti selezionati, potendo aggiungere nuovi articoli, rimuovere quelli non desiderati e modificare la quantità di ciascun prodotto.

Gestione degli ordini: Gli acquirenti possono visualizzare i loro ordini attraverso il loro profilo personale, con informazioni utili sulla spedizione dei prodotti e con tutte le informazioni fondamentali riguardo i diversi prodotti inclusi nell'ordine effettuato.
I venditori attraverso il loro profilo possono visualizzare gli ordini riguardanti esclusivamente i prodotti da loro venduti a differenti utenti finali.

3. Progettazione concettuale e logica della base di dati

La progettazione del database per l'applicazione TradeTrove è stata realizzata utilizzando il framework SQLAlchemy per Flask. Il modello prevede le seguenti entità principali:

3.1. Modelli e Relazioni

1. **User**
   - **Descrizione**: Rappresenta gli utenti che possono registrarsi, accedere e gestire i propri profili.
   - **Attributi**:
     - `id`: Identificatore univoco dell'utente.
     - `username`: Nome utente unico.
     - `email`: Indirizzo email unico.
     - `indirizzo`, `città`, `nazione`, `cap`: Informazioni di contatto.
     - `password_hash`: Hash della password per la sicurezza.
   - **Metodi**:
     - `set_password(password)`: Imposta la password dell'utente in forma hashata.
     - `check_password(password)`: Verifica la password fornita con l'hash salvato.
2. **Vendor**
   - **Descrizione**: Rappresenta i venditori registrati sulla piattaforma.
   - **Attributi**: Simili a quelli di `User`, ma include anche `piva` e `nome_azienda` per identificare l'attività commerciale.
   - **Metodi**: Stessi metodi di `User` per la gestione della password.
3. **Category**
   - **Descrizione**: Rappresenta le categorie di prodotto.
   - **Attributi**:
     - `id`: Identificatore univoco della categoria.
     - `name`: Nome della categoria.
     - `image_url`: URL di un'immagine rappresentativa della categoria.
4. **Product**
   - **Descrizione**: Rappresenta i prodotti in vendita.
   - **Attributi**:
     - `id`: Identificatore univoco del prodotto.

        - `name`, `description`, `price`, `quantity`: Informazioni dettagliate sul prodotto.
        - `seller_id`: Riferimento al venditore del prodotto.
        - `category_id`: Riferimento alla categoria del prodotto.
        - `created_at`, `updated_at`: Timestamp di creazione e aggiornamento.
    - **Relazioni**: Può avere recensioni (`Review`), può essere collegato a più carrelli (`CartItem`), appartiene ad una categoria (`Category`) ed è venduto da un venditore (`Vendor`).
5. **Cart**
    - **Descrizione**: Rappresenta il carrello della spesa di un utente.
    - **Attributi**: Include l'ID dell'utente e i timestamp di creazione e aggiornamento.
    - **Relazioni**: Può contenere più `CartItem`, appartiene ad un utente (`User`).
6. **CartItem**
    - **Descrizione**: Rappresenta gli articoli presenti nel carrello.
    - **Attributi**: Include riferimenti a `cart_id` e `product_id` con la quantità del prodotto.
    - **Relazioni**: Collega i prodotti (`Product`) al carrello (`Cart`) .
7. **Order**
    - **Descrizione**: Rappresenta un ordine effettuato da un utente.
    - **Attributi**: Include l'ID dell'utente, lo stato dell'ordine, il prezzo totale e timestamp.
    - **Relazioni**: Contiene più `OrderItem` e viene effettuato da un acquirente (`User`).
8. **OrderItem**
    - **Descrizione**: Rappresenta gli articoli all'interno di un ordine.
    - **Attributi**: Include riferimenti a `order_id` e `product_id`, insieme alla quantità e al prezzo unitario.
    - **Relazioni:** collega i prodotti (`Product`) che fanno parte dello stesso ordine (`Order`).
9. **Review**
    - **Descrizione**: Permette agli utenti di lasciare recensioni sui prodotti acquistati.
    - **Attributi**: Include valutazione, commento e timestamp.
    - **Relazioni:** permette ad un utente (`User`) di recensire un prodotto (`Product`).

3.2. Schema della Base di Dati

Diagramma ad Oggetti

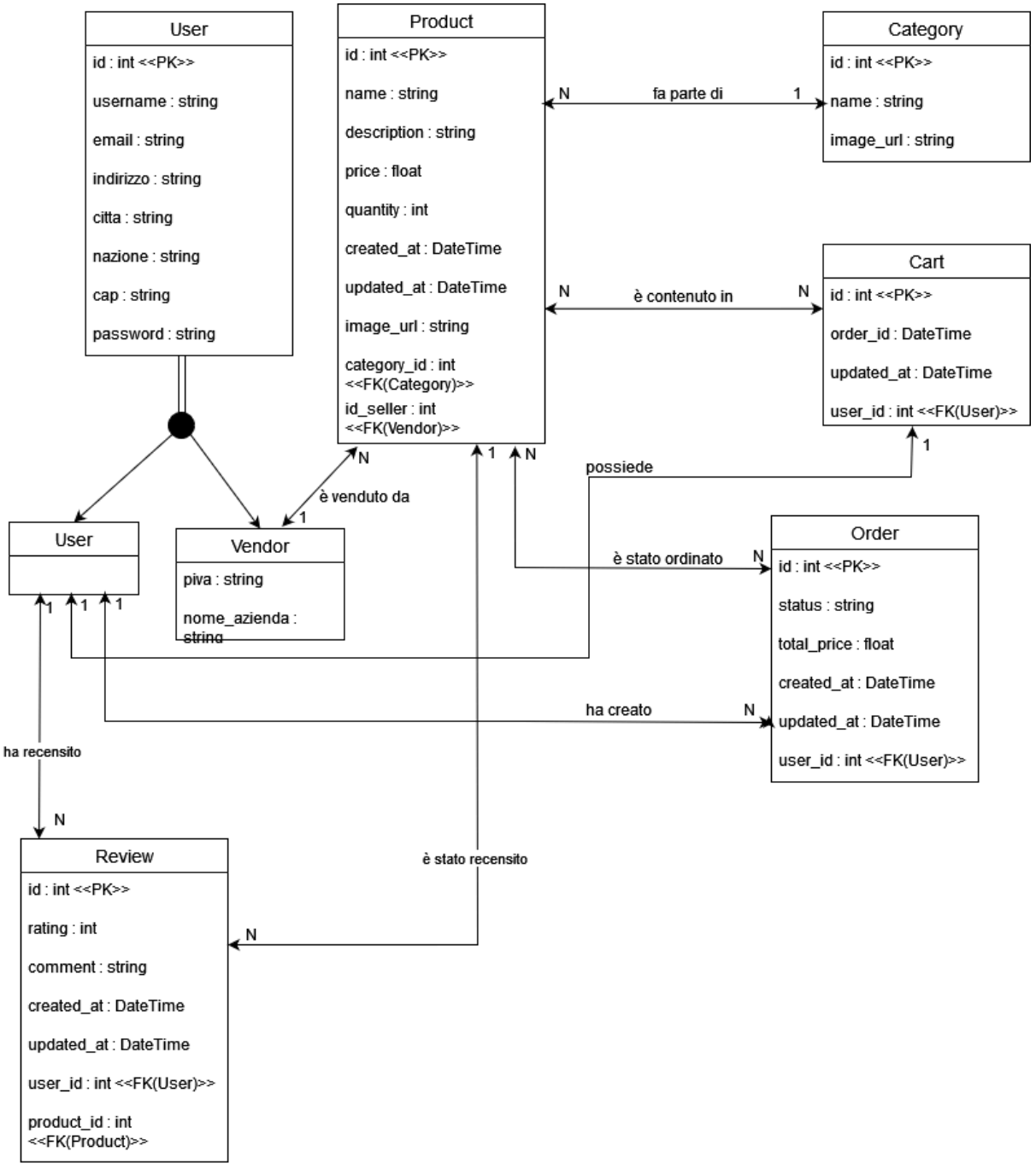
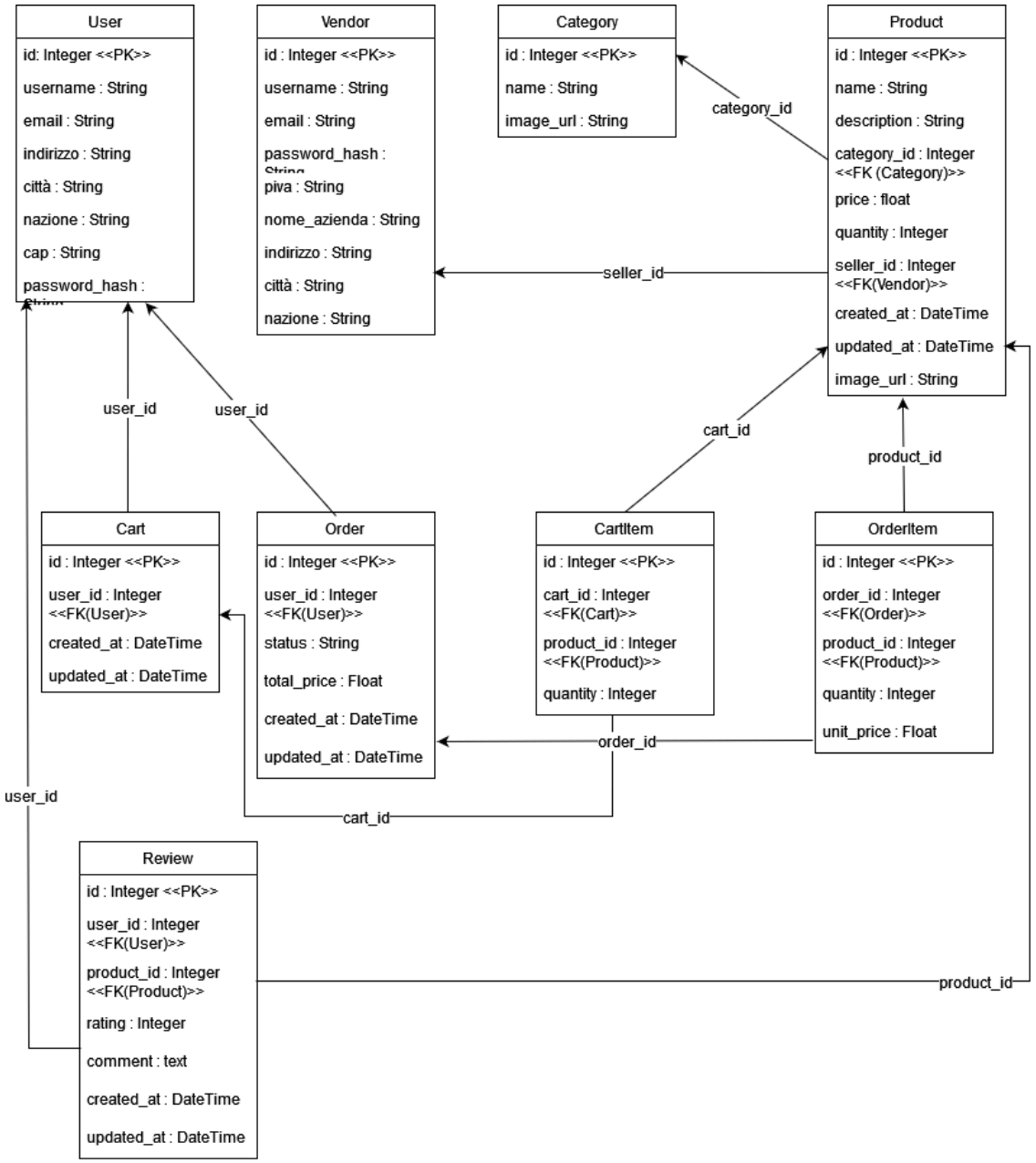


Diagramma Entità Relazione



3.3. Motivazioni progettuali

La progettazione del database è stata realizzata per garantire l'integrità e la coerenza dei dati, utilizzando vincoli di unicità per gli attributi chiave e relazioni ben definite tra le entità. È stata posta particolare attenzione alla sicurezza, con l'hashing delle password e l'uso di Foreign Key per mantenere l'integrità referenziale.

4. Query principali

Login: Verifica delle credenziali e distinzione tra utente normale e venditore.

```
user = User.query.filter_by(username=username).first()
vendor = Vendor.query.filter_by(username=username).first()
```

Ricerca prodotti per categorie e filtri: Ricerca avanzata di prodotti filtrata per categoria, prezzo.

```
query = Product.query
query = query.filter(Product.category_id == category_filter)
query = query.order_by(Product.price.asc() or Product.price.desc())
```

Visualizzazione ordini: Recupera gli ordini e articoli associati di un utente o venditore.

```
orders = Order.query.filter_by(user_id=current_user.id).all()
order_items = OrderItem.query.filter_by(order_id=order_id).all()
```

Elimina prodotto: Rimozione di un prodotto, articoli associati nel carrello e ordini.

```
# Elimina manualmente gli articoli di carrello associati al prodotto
cart_items = CartItem.query.filter_by(product_id=product_id).all()
for item in cart_items:
    db.session.delete(item)

# Elimina manualmente gli articoli di ordine associati al prodotto
order_items = OrderItem.query.filter_by(product_id=product_id).all()
for item in order_items:
    db.session.delete(item)
```

Visualizzazione ordini: Recupera gli ordini e articoli associati di un utente o venditore.

```
orders = Order.query.filter_by(user_id=current_user.id).all()
order_items = OrderItem.query.filter_by(order_id=order_id).all()
```

Trigger per Cambio Password dell'Utente: controlla e cambia la vecchia password con una nuova.

```
CREATE OR REPLACE FUNCTION password_change() RETURNS TRIGGER AS $$
BEGIN
    IF OLD.password_hash != NEW.password_hash THEN
        INSERT INTO user_password_log (user_id, old_password_hash)
        VALUES (NEW.id, OLD.password_hash);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER password_change_trigger AFTER UPDATE ON "user"
FOR EACH ROW EXECUTE FUNCTION password_change();
```

Trigger per Prezzo Totale dell'Ordine: aggiorna il nuovo totale dell'ordine.

```
CREATE OR REPLACE FUNCTION total_price() RETURNS TRIGGER AS $$
DECLARE calculated_total FLOAT := 0;
BEGIN
    SELECT SUM(order_item.unit_price * order_item.quantity)
    INTO calculated_total FROM order_item
    WHERE order_item.order_id = NEW.id;
    IF NEW.total_price != calculated_total THEN
        RAISE EXCEPTION 'Mismatch prezzo totale: % (dovrebbe essere %)', NEW.total_price, calculated_total;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER order_total_price BEFORE INSERT OR UPDATE ON "order"
FOR EACH ROW EXECUTE FUNCTION total_price();
```

Trigger per Cancellazione dell'Ordine: cancella un ordine effettuato dal cliente.

```
CREATE OR REPLACE FUNCTION cancellation() RETURNS TRIGGER AS $$
BEGIN
    RAISE NOTICE 'Order % has been cancelled', OLD.id;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER cancellation_trigger AFTER DELETE ON "order"
FOR EACH ROW EXECUTE FUNCTION cancellation();
```

5. Principali scelte progettuali

Nell'ambito del progetto descritto, sono state effettuate determinate scelte progettuali per garantire il corretto funzionamento dell'applicazione. Le principali scelte riguardano le politiche di integrità, la gestione dei ruoli e delle autorizzazioni, l'uso di indici e altre decisioni di design rilevanti. Queste scelte progettuali mirano a garantire un'applicazione web sicura, scalabile e facile da mantenere.

1. **Politiche di integrità**

Le politiche di integrità sono state garantite attraverso l'uso di relazioni forti tra le entità e l'integrità referenziale.

- **Relazioni tra modelli:** Il codice utilizza relazioni di tipo foreign key tra i modelli, come `User`, `Vendor`, `Product`, `Order`, e `CartItem`. Queste relazioni assicurano che ogni entità abbia una relazione coerente con le altre entità, come ad esempio `Order` che appartiene a un `User` e include `OrderItem` che fa riferimento a Product. In questo modo, si evitano stati incoerenti come ordini senza utenti o prodotti.
- **Controllo delle dipendenze tra le tabelle:** Quando si elimina un prodotto, il sistema elimina manualmente le dipendenze per evitare violazioni di integrità. Queste dipendenze vengono gestite in cascata utilizzando logiche customizzate per mantenere la coerenza.
- **Gestione delle transizioni:** Durante le operazioni critiche, come la registrazione di un nuovo utente o la creazione di un ordine, il database viene coinvolto in una transazione. Se un errore si verifica durante la commit, viene eseguito un rollback per mantenere il sistema in uno stato consistente, come mostrato nei vari blocchi `try/except`.

2. **Definizione di ruoli e politiche di autorizzazione**

L'applicazione gestisce due ruoli principali: User e Vendor. Questi ruoli determinano l'accesso a risorse e funzionalità specifiche:

- **Gestione tramite** `Flask-login`: Il framework `Flask-login` viene utilizzato per la gestione dell'autenticazione e della sessione tra utenti. Funzioni come `login_user()`, `logout_user()` e `login_required` garantiscono che solo gli utenti autenticati possano accedere a determinate pagine e risorse.
- **Login differenziato per** `User` **e** `Vendor`**:** Il sistema verifica se un account appartiene a un normale utente (`User`) o a un venditore (`Vendor`), e reindirizza gli utenti autenticati alle pagine appropriate. Questo garantisce che le funzionalità siano adeguatamente separate tra i diversi ruoli. I venditori hanno un extranet a loro dedicato.
- **Controlli specifici nei moduli:** Vengono eseguiti controlli per impedire la registrazione di duplicati (es. username o email già registrato). Questo evita conflitti di identità e garantisce che ogni utente o venditore sia unico nel sistema.

3. **Gestione dei permessi e validazione**

Le politiche di autorizzazione e validazione sono fondamentali per mantenere il sistema sicuro e affidabile.

- **Autorizzazione per funzioni sensibili:** Funzioni come le gestione del carrello, la modifica del profilo e l'accesso ai dettagli degli ordini sono protette dal decoratore `@login_required`, che garantisce che solo gli utenti autenticati possano accedere a questa funzionalità.
- **Validazione del modulo:** Le operazioni di login e registrazione fanno uso di moduli come LoginForm, RegistrationForm, VendorForm, che includono valutazioni preliminari per evitare errori (es. campi obbligatori mancanti o errati). Questo riduce il rischio di immissione di dati non validi.

4. Uso di indici

Gli indici svolgono un ruolo importante per ottimizzare le operazioni di lettura sul database, specialmente nelle query frequenti e nei filtri:

- **Indici per username e email:** Campi come username e email degli utenti (User, Vendor) sono comunemente utilizzati per le ricerche durante il login e la registrazione. Includere indici su questi campi consente al database di eseguire queste query in modo efficiente, migliorando le prestazioni.
- **Indici per le relazioni chiave esterna:** Anche i campi che rappresentano relazioni, come `user_id` in `Cart` o `product_id` in `OrderItem`, beneficiano di indici che accelerano le ricerche di prodotti o ordini associati a un utente.

5. Trigger e gestione delle transazioni

I trigger e le transazioni sono utilizzati per mantenere l'integrità delle informazioni.

- **Esempio di trigger applicativo:** Quando un utente completa il pagamento, il sistema crea automaticamente un nuovo ordine, trasferisce gli articoli dal carrello all'ordine e svuota il carrello. Questa operazione avviene in una singola transazione, il che significa che se un errore si verifica in uno di questi passaggi, l'intera operazione viene annullata.
- **Gestione delle eliminazioni**: In caso di eliminazione di un prodotto, vengono attivati dei "trigger" applicativi per eliminare anche le recensioni e i riferimenti al prodotto nelle tabelle `CartItem` e `OrderItem`. Questo garantisce che non rimangano riferimenti orfani nel database, preservando l'integrità referenziale.

6. Gestione dei file

Per il caricamento di immagini di prodotti, viene implementata una gestione dei file sicura.

- **Sicurezza nei nomi dei file:** L'uso della funzione `secure_filename()` di `Werkzeug` garantisce che i nomi dei file caricati non contengano caratteri non sicuri, proteggendo il sistema da possibili attacchi tramite path traversal.
- **Verifica del formato del file:** Viene implementata una funzione per controllare che i file caricati siano di un tipo permesso (`png`, `jpg`, `jpeg`, `gif`), garantendo così che solo file immagine appropriati vengano accettati.

6. Ulteriori informazioni

Per sviluppare l'applicazione sono state utilizzate le seguenti tecnologie:

- **Python v3**: Linguaggio utilizzato per creare il server, che funge da interfaccia tra il sito web e il database, gestendo la logica dell'applicazione.
- **Postgresql:** Sistema di gestione di database relazionale (DBMS) utilizzato per memorizzare i dati.
- **SQLalchemy**: API che facilita l'associazione di classi Python definite dall'utente con il database. Consente di mantenere sincronizzate le modifiche tra gli oggetti Python e le righe nelle tabelle del database ed esprimere query del database utilizzando le classi e le relazioni definite in Python.
- **Flask:** Framework web scritto in Python che consente una facile interazione tra server e sito web ed altri moduli correlati a Flask.

Le pagine web dell'applicazione sono state create utilizzando il linguaggio di marcatura HTML. Tutte le viste del sito, come la homepage, la pagina di login, la registrazione dei prodotti e altre sezioni, sono contenute nella directory `app\templates`.

7. Contributo al progetto (appendice)

Tutti i membri del gruppo hanno contribuito in modo equilibrato alla realizzazione dell'applicazione, collaborando in maniera reciproca. Ciascun componente ha completato prevalentemente, ma non esclusivamente, ciascuna parte assegnata.
In particolare, **KRISTI NEZHAJ** si è occupato maggiormente della realizzazione dell'applicazione Python, **FEDERICO GAUDENZI** ha curato principalmente lo sviluppo del sito web. La creazione dei diagrammi della base di dati, la documentazione e la produzione del video sono state realizzate da **CRISTINA AVORNIC.**