



Università Ca' Foscari - Venezia

Relazione del Progetto di Basi di Dati

Piattaforma di E-commerce

Massimo Costantini - 877336

Samuel Obeng Takyi - 881431

Filippo Tiozzo - 887683

Anno Accademico 2023 - 2024

Indice	2
1 Introduzione	3
1.1 Applicazione	3
1.2 Blueprint	3
1.3 Caricamento grant e trigger	3
2 Funzionalità principali	4
2.1 Esecuzione dell'applicazione	5
2.2 Requisiti del progetto	13
3 Progettazione concettuale e logica della basi di dati	14
3.1 Modello Grafico ad Oggetti	14
3.2 Classi (tabelle)	15
3.3 Associazioni	20
3.4 Modello Logico Relazionale	22
4 Query	23
5 Principali scelte progettuali	29
5.1 Autorizzazioni	32
5.2 Trigger	33
6 Ulteriori informazioni	35
6.1 Comandi utili	35
7 Contributo al progetto	37

1 Introduzione

Il progetto scelto riguarda la Piattaforma di E-commerce, una web application sviluppata in linguaggio Python mediante il framework Flask, ed interfacciata con il database relazionale PostgreSQL utilizzando la libreria SQLAlchemy.

In questo documento sono descritte le funzionalità principali dell'applicazione, la progettazione concettuale e logica della base di dati, le query e le scelte progettuali che sono state fatte.

1.1 Applicazione

L'interfaccia web permette di registrarsi e collegarsi come acquirente oppure come venditore fornendo una serie di possibili azioni, diverse a seconda del ruolo, come ad esempio la scelta da parte dell'acquirente dei prodotti da acquistare e la creazione da parte del venditore di nuovi prodotti da vendere.

1.2 Blueprint

L'applicazione è suddivisa in Blueprint Flask, un insieme di route, cioè di percorsi URL, che gestiscono le chiamate HTTP.

In questo modo il codice sorgente è suddiviso in moduli (i file `acquirente.py`, `logica.py`, `modelli.py` e `venditore.py`) che facilitano così la gestione del progetto.

1.3 Caricamento grant e trigger

Per il caricamento dei grant e dei trigger necessari al funzionamento dell'applicazione, sono stati inseriti all'interno del progetto due file chiamati `grant.sql` e `trigger.sql`.

Un ulteriore file chiamato `comandi.sql` contiene i comandi necessari alla creazione del database e dei ruoli.

Alla Sezione 6.1 viene anche elencata una serie di comandi utili per muoversi all'interno del database utilizzato che, per questo progetto, è PostgreSQL.

2 Funzionalità principali

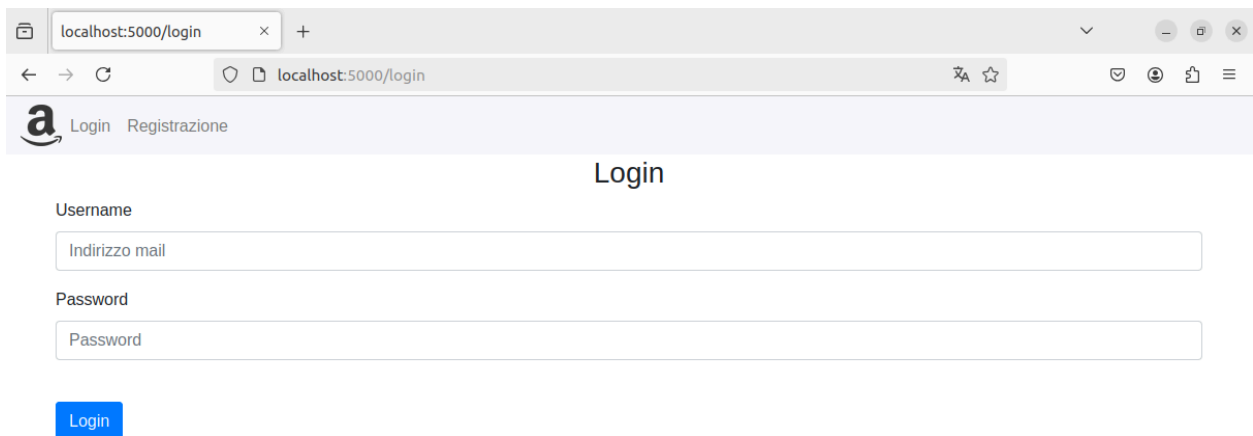
La scelta del progetto Piattaforma di E-commerce ci ha permesso di sviluppare l'applicazione suddividendola opportunamente nelle due parti acquirente e venditore, assegnando un ruolo diverso durante la registrazione dell'utente per poter poi assegnare ad esso le funzionalità corrispondenti.

2.1 Esecuzione dell'applicazione

Di seguito una breve spiegazione di come viene eseguita l'applicazione in base ai requisiti suggeriti.

Si avvia andando direttamente alla pagina di login, dove si ha la possibilità di effettuare il login o la registrazione di un nuovo utente.

In Figura 1 la schermata relativa al login di un utente:

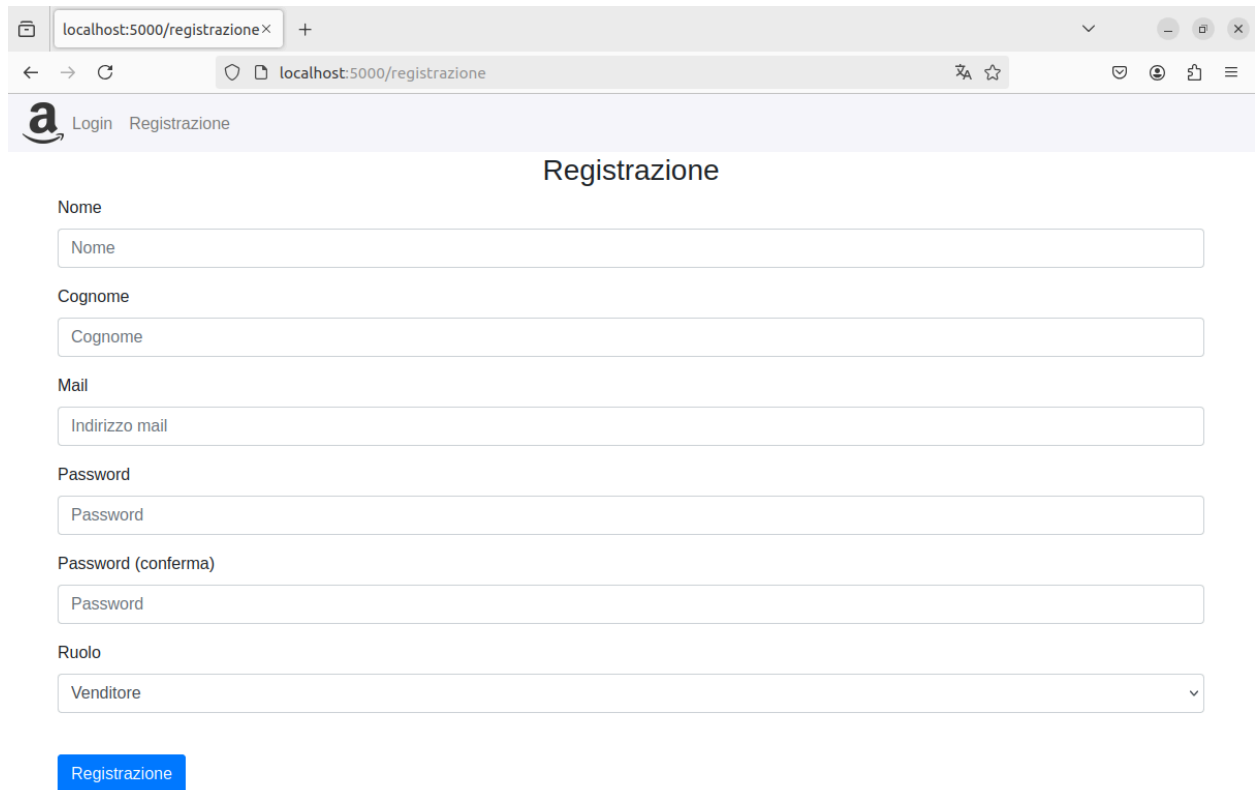


The screenshot shows a web browser window with the address bar displaying 'localhost:5000/login'. The page features a header with an Amazon logo and navigation links for 'Login' and 'Registrazione'. The main section is titled 'Login' and contains two input fields: 'Username' with a placeholder 'Indirizzo mail' and 'Password' with a placeholder 'Password'. Below these fields is a blue 'Login' button.

Figura 1: Login utente

Effettuando la registrazione di un nuovo utente si dovrà scegliere il ruolo per il quale ci si registra, acquirente o venditore, andando poi direttamente alla homepage dedicata in base al ruolo scelto.

In Figura 2 la schermata relativa alla registrazione di un nuovo utente:



The screenshot shows a web browser window with the address bar displaying 'localhost:5000/registrazione'. The page has a header with the Amazon logo and links for 'Login' and 'Registrazione'. The main heading is 'Registrazione'. Below this, there are several input fields: 'Nome', 'Cognome', 'Mail' (with placeholder 'Indirizzo mail'), 'Password', and 'Password (conferma)' (both with placeholder 'Password'). At the bottom, there is a 'Ruolo' dropdown menu with 'Venditore' selected. A blue button labeled 'Registrazione' is positioned below the form fields.

Figura 2: Registrazione utente

Gli acquirenti possono quindi visualizzare i prodotti da acquistare, oppure cercare prodotti da acquistare.

In Figura 3 la schermata relativa all'elenco dei prodotti acquistati:

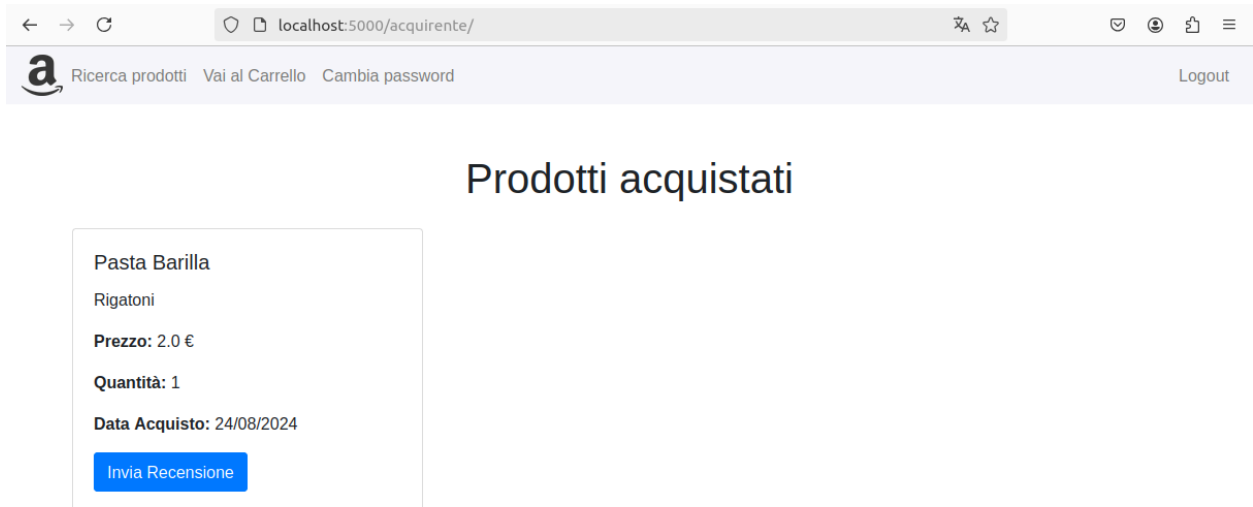
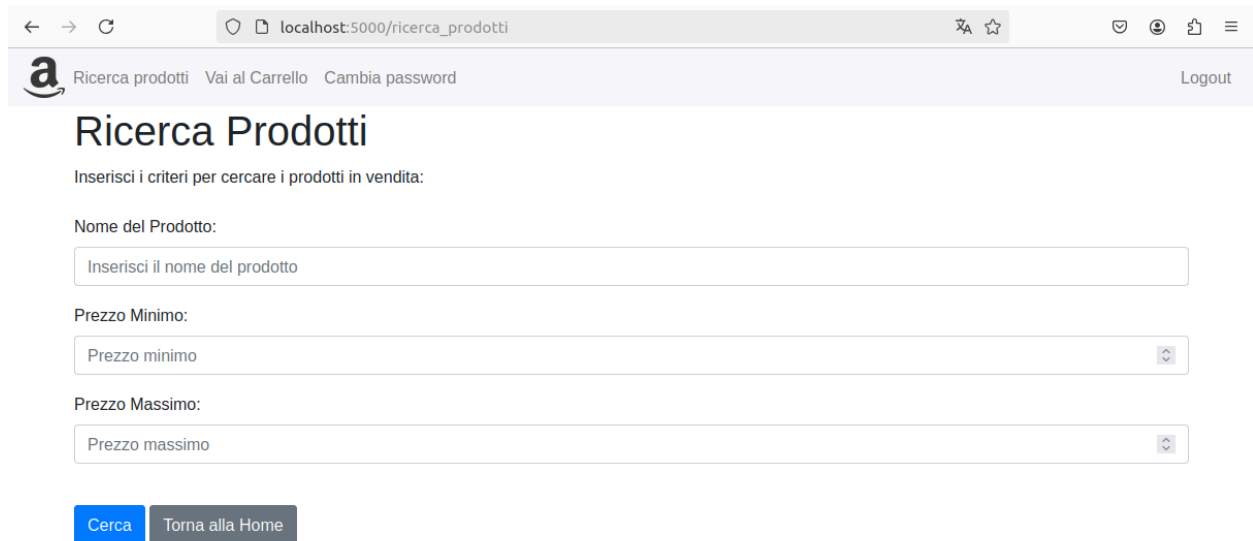


Figura 3: Elenco dei prodotti acquistati

Per cercare un prodotto l'acquirente può selezionare una sottostringa (ad esempio "pasta" oppure "barilla") ma anche un range di prezzo.

In Figura 4 la schermata relativa alla funzionalità di ricerca dei prodotti da acquistare:



The screenshot shows a web browser window with the address bar displaying "localhost:5000/ricerca_prodotti". The page has a header with a logo "a" and navigation links: "Ricerca prodotti", "Vai al Carrello", "Cambia password", and "Logout". The main heading is "Ricerca Prodotti". Below it, a subheading reads "Inserisci i criteri per cercare i prodotti in vendita:". There are three input fields: "Nome del Prodotto:" with a placeholder "Inserisci il nome del prodotto", "Prezzo Minimo:" with a placeholder "Prezzo minimo", and "Prezzo Massimo:" with a placeholder "Prezzo massimo". At the bottom, there are two buttons: "Cerca" (blue) and "Torna alla Home" (grey).

Figura 4: Funzionalità di ricerca dei prodotti da acquistare

Inoltre l'acquirente può visualizzare il proprio carrello della spesa, rimuovere o acquistare dei prodotti.

In Figura 5 la schermata relativa al carrello della spesa:

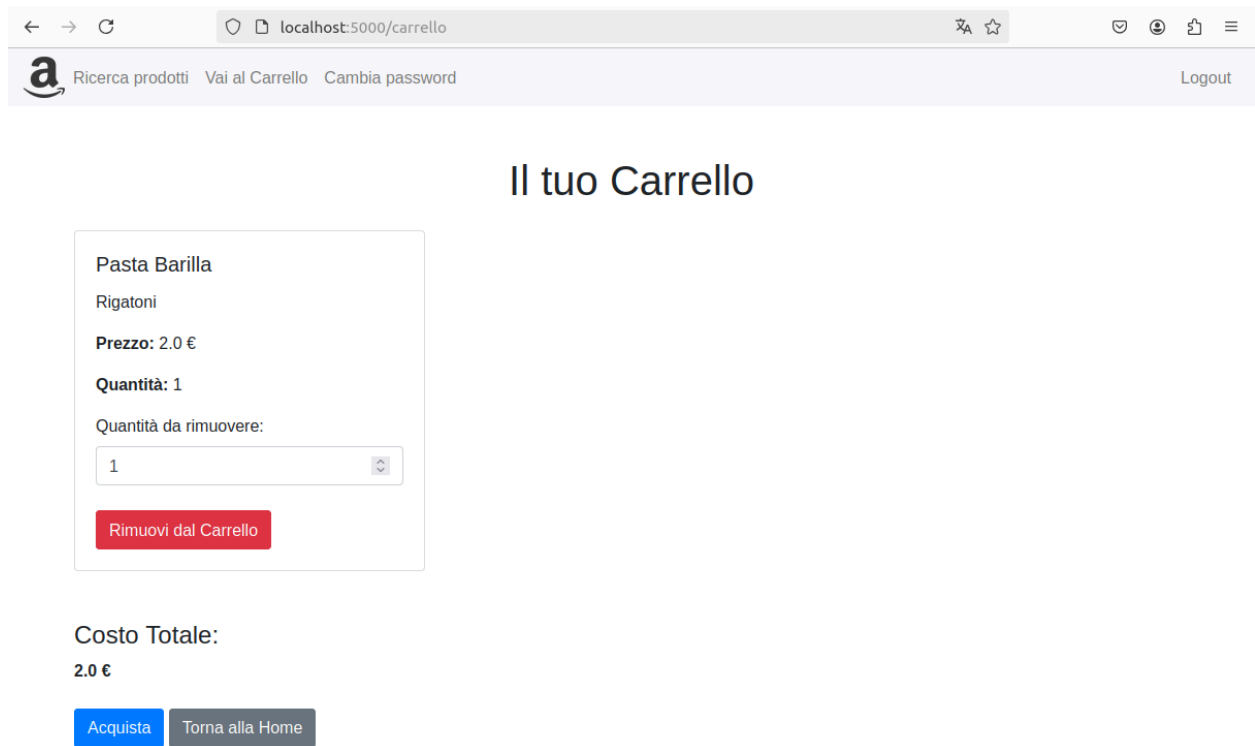
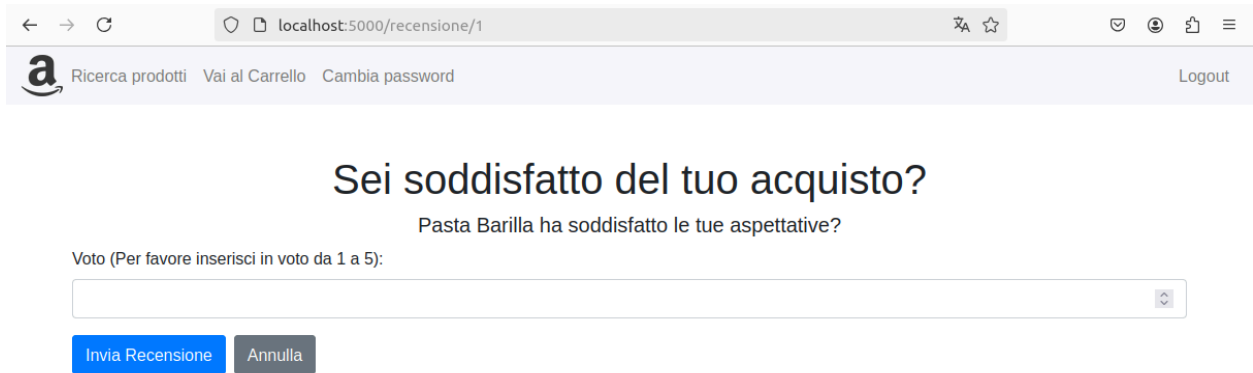


Figura 5: Carrello della spesa

Infine l'acquirente può recensire i prodotti acquistati.

In Figura 6 la schermata relativa alla funzionalità di recensione dei prodotti acquistati:



The screenshot shows a web browser window with the address bar displaying 'localhost:5000/recensione/1'. The page header features the Amazon logo, navigation links for 'Ricerca prodotti', 'Vai al Carrello', and 'Cambia password', and a 'Logout' link. The main content area is titled 'Sei soddisfatto del tuo acquisto?' and asks 'Pasta Barilla ha soddisfatto le tue aspettative?'. Below this, a prompt says 'Voto (Per favore inserisci in voto da 1 a 5):' followed by a text input field. At the bottom, there are two buttons: 'Invia Recensione' (highlighted in blue) and 'Annulla'.

Figura 6: Funzionalità di recensione dei prodotti acquistati

Per quanto riguarda i venditori invece, questi possono creare nuovi prodotti, modificando o rimuovendo quelli esistenti.

In Figura 7 la schermata relativa alla funzionalità di creazione dei prodotti da vendere

The screenshot shows a web browser window with the address bar displaying 'localhost:5000/vendi_prodotto'. The page has a header with a logo 'a' and navigation links: 'Metti in Vendita', 'Gestisci ordini', 'Cambia password', and a 'Logout' button. The main heading is 'Metti in vendita il tuo prodotto'. Below this, there are four input fields: 'Nome' with placeholder 'Nome Prodotto', 'Descrizione' with placeholder 'Descrivi il tuo prodotto', 'Prezzo' with placeholder 'Prezzo di vendita' and a dropdown arrow, and 'Quantità' with placeholder 'Quantità di prodotti che desideri vendere' and a dropdown arrow. At the bottom, there are two buttons: 'Conferma' (blue) and 'Torna alla Home' (grey).

Figura 7: Funzionalità di creazione dei prodotti da vendere

Inoltre il venditore può visualizzare l'elenco dei prodotti in vendita.

In Figura 8 la schermata relativa all'elenco dei prodotti in vendita:

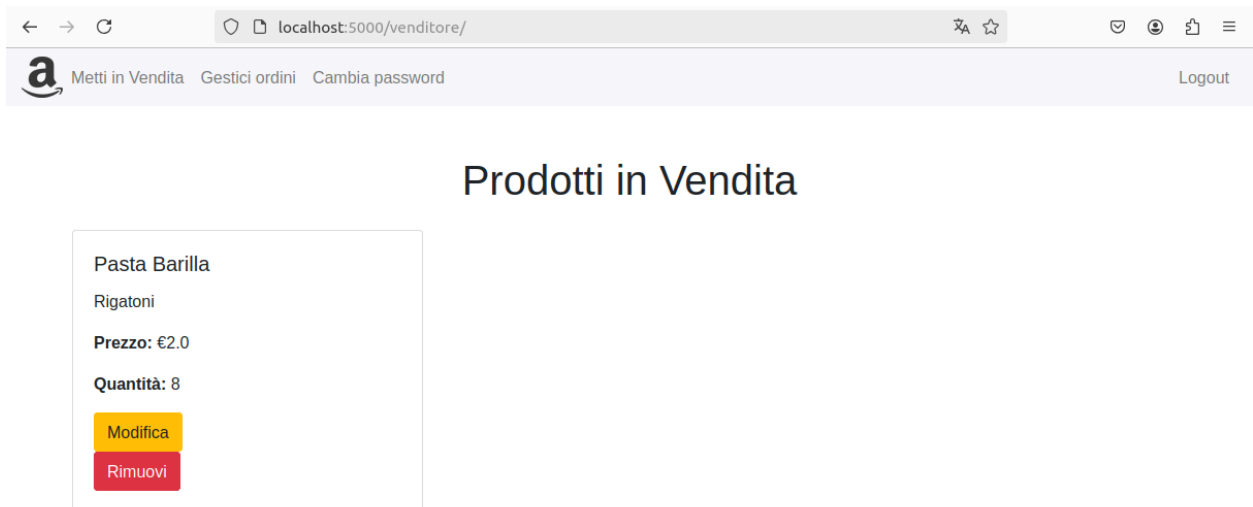


Figura 8: Prodotti in vendita

Infine il venditore può gestire gli ordini (ad esempio spedirli).

In Figura 9 la schermata relativa alla gestione degli ordini:

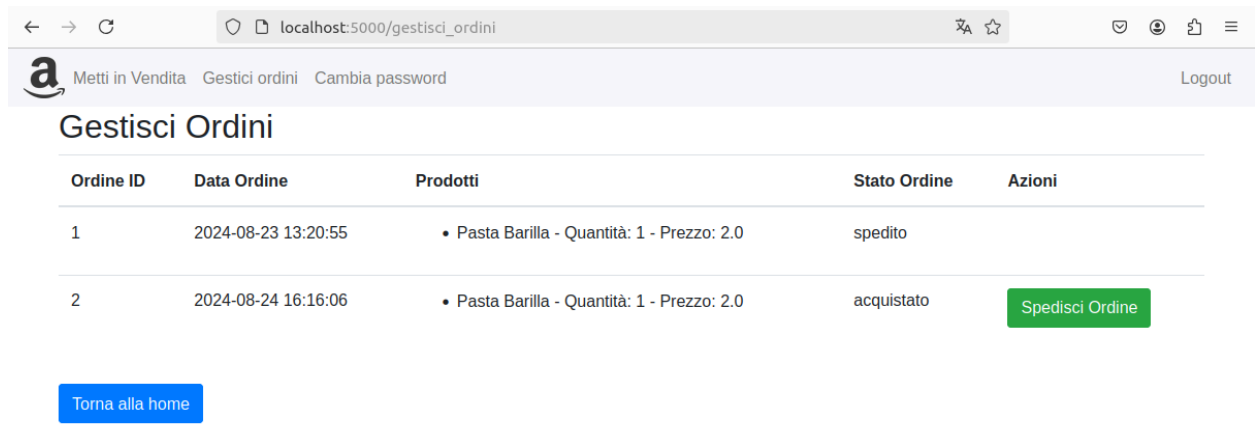


Figura 9: Gestione degli ordini

2.2 Requisiti del progetto

Tutti i requisiti richiesti per lo sviluppo del progetto per la Piattaforma di E-commerce sono stati soddisfatti.

Gli utenti si possono infatti registrare come acquirenti o venditori, accedere e gestire i propri profili (ad esempio cambiare la password).

E' stato creato un database per memorizzare le informazioni sui prodotti che i venditori possono gestire.

Gli acquirenti possono cercare i prodotti da acquistare per sottostringa o per intervallo di prezzo.

E' stato implementato un carrello della spesa che permette agli acquirenti di gestire i propri prodotti da acquistare, comprese le disponibilità dei prodotti.

Gli acquirenti possono inoltre visualizzare dati relativi ai propri ordini (come ad esempio la data di acquisto).

Infine gli acquirenti possono lasciare delle recensioni riguardanti i prodotti acquistati.

3 Progettazione concettuale e logica della basi di dati

In questo Paragrafo vengono descritti lo schema grafico ad oggetti, le tabelle, le relazioni ed il modello logico relazionale della base di dati.

3.1 Modello Grafico ad Oggetti

In Figura 10 il modello grafico ad oggetti:

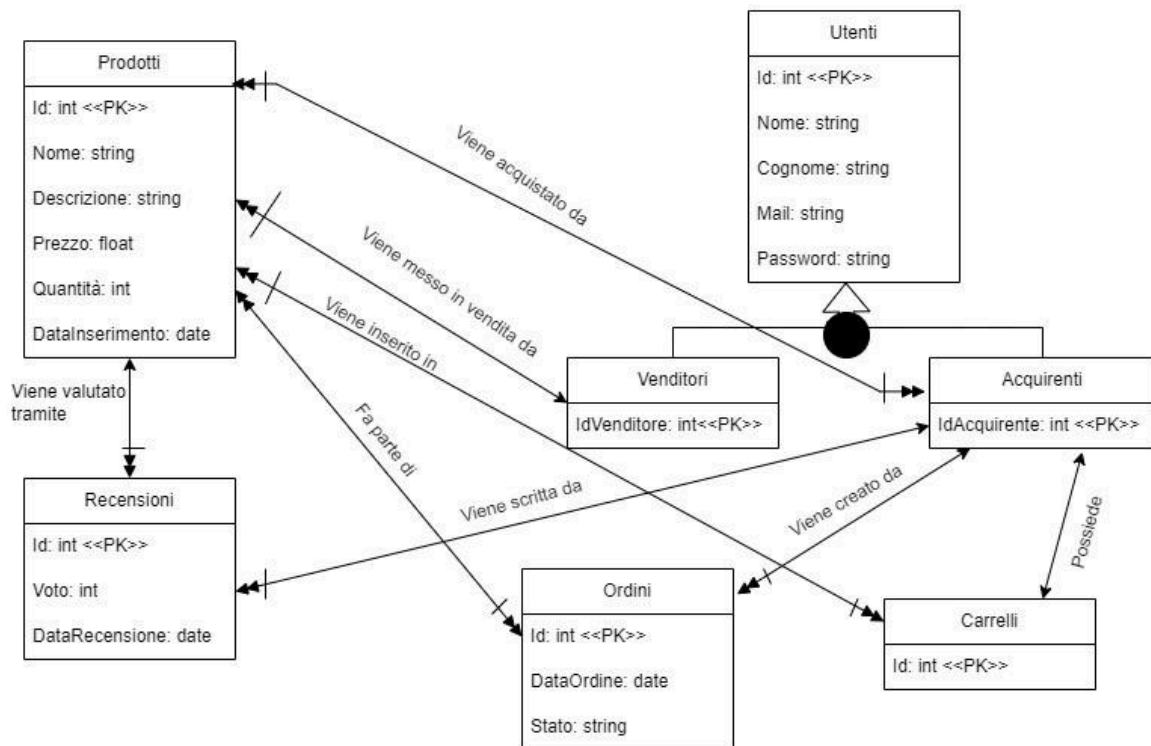


Figura 10: Modello grafico ad oggetti

3.2 Classi (tabelle)

In questa Sezione vengono descritte le classi (tabelle) utilizzate per modellare la base di dati.

Le tabelle sono definite cioè come classi ORM (Object Relational Mapping) di SQLAlchemy.

Si utilizzano così funzioni Python per manipolare i dati invece di scrivere direttamente codice SQL.

Definizione classe (tabella) Utenti

```
class Utenti(db.Model, UserMixin):
```

```
    id = db.Column(db.Integer, primary_key=True, index=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    password = db.Column(db.String(), nullable=False)
    ruolo = db.Column(db.String(10), index=True)
```

La classi ereditano da `db.Model`, che è una classe base fornita da SQLAlchemy per rappresentare una tabella nel database, e da `UserMixin`, che è utilizzata per fornire metodi utili all'autenticazione come `is_authenticated`, `is_active`, `is_anonymous`, e `get_id()`.

Definizione classe (tabella) Acquirente

```
class Acquirente(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
    nome = db.Column(db.String(50), nullable=False)
    cognome = db.Column(db.String(50), nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('utenti.id'), index=True)
    user = db.relationship('Utenti', backref='acquirenti')
```

Il campo `user_id` è una chiave esterna che fa riferimento al campo `id` della tabella `Utenti`.

Serve a creare una relazione tra la tabella `Acquirente` e la tabella `Utenti`, indicando quale utente è associato a quale acquirente.

È indicizzato (`index=True`) per ottimizzare le ricerche e le operazioni di join basate su questo campo.

Il campo user definisce invece una relazione tra l'acquirente e la tabella Utenti.

La relazione è configurata in modo tale che, dato un oggetto Acquirente, sia possibile accedere all'utente associato tramite l'attributo user.

L'argomento backref='acquirenti' crea infine un attributo virtuale acquirenti nella classe Utenti, che rappresenta l'insieme di tutti gli acquirenti associati a un dato utente.

Serve per accedere a tutti gli acquirenti di un utente direttamente dall'oggetto Utenti.

Definizione classe (tabella) Venditore

```
class Venditore(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
```

```
    nome = db.Column(db.String(50), nullable=False)
```

```
    cognome = db.Column(db.String(50), nullable=False)
```

```
    user_id = db.Column(db.Integer, db.ForeignKey('utenti.id'), index=True)
```

```
    user = db.relationship('Utenti', backref='venditori')
```

```
    prodotti = db.relationship("Prodotto", back_populates="venditore")
```


Definizione classe (tabella) Prodotto

```
class Prodotto(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    nome = db.Column(db.String, nullable=False)  
    descrizione = db.Column(db.String, nullable=False)  
    prezzo = db.Column(db.Float, nullable=False)  
    quantita = db.Column(db.Integer, nullable=False)  
    data_inserimento = db.Column(db.DateTime, default=datetime.now)  
    venditore_id = db.Column(db.Integer, db.ForeignKey('venditore.id'), index=True)  
    venditore = db.relationship("Venditore", back_populates="prodotti")  
    def media_voti(self):  
        recensioni = Recensione.query.filter_by(prodotto_id=self.id).all()  
        if recensioni:  
            return sum(r.voto for r in recensioni) / len(recensioni)  
        return None
```

media_voti è un metodo che calcola la media dei voti (recensioni) associati a un determinato prodotto.

Utilizza una query per ottenere tutte le recensioni associate al prodotto corrente.

Se ci sono recensioni, calcola la media dei voti sommando i voti e dividendo per il numero di recensioni.

Se non ci sono recensioni, il metodo restituisce None.

Definizione classe (tabella) Carrello

```
class Carrello(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    user_id = db.Column(db.Integer, db.ForeignKey('utenti.id'), nullable=False)  
    prodotti = db.relationship('CarrelloProdotto', back_populates='carrello')
```

Definizione classe (tabella) CarrelloProdotto

```
class CarrelloProdotto(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
    carrello_id = db.Column(db.Integer, db.ForeignKey('carrello.id'), nullable=False)
    prodotto_id = db.Column(db.Integer, db.ForeignKey('prodotto.id'), nullable=False)
    quantita = db.Column(db.Integer, nullable=False)
    sconto = db.Column(db.Boolean, default=False)
    carrello = db.relationship('Carrello', back_populates='prodotti')
    prodotto = db.relationship('Prodotto')
```

Definizione classe (tabella) Acquisto

```
class Acquisto(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('utenti.id'), nullable=False)
    prodotto_id = db.Column(db.Integer, db.ForeignKey('prodotto.id'), nullable=False)
    quantita = db.Column(db.Integer, nullable=False)
    data_acquisto = db.Column(db.DateTime, default=datetime.now, nullable=False)
    user = db.relationship('Utenti', backref='acquisti')
    prodotto = db.relationship('Prodotto', backref='acquisti')
    def ha_recensione(self):
        return Recensione.query.filter_by(prodotto_id=self.prodotto_id, user_id=self.user_id).first() is
not None
```

ha_recensione è un metodo che verifica se esiste già una recensione per l'acquisto corrente.

Se esiste una recensione, il metodo restituisce True, altrimenti restituisce False.

Definizione classe (tabella) Ordine

```
class Ordine(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    user_id = db.Column(db.Integer, db.ForeignKey('utenti.id'), nullable=False)  
    data_ordine = db.Column(db.DateTime, default=datetime.now)  
    prodotti = db.relationship('OrdineProdotto', backref='ordine', lazy=True)  
    stato = db.Column(db.String(20), nullable=False, default='acquistato')
```

Definizione classe (tabella) OrdineProdotto

```
class OrdineProdotto(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    ordine_id = db.Column(db.Integer, db.ForeignKey('ordine.id'), nullable=False)  
    prodotto_id = db.Column(db.Integer, db.ForeignKey('prodotto.id'), nullable=False)  
    quantita = db.Column(db.Integer, nullable=False)  
    spedito = db.Column(db.Boolean, default=False)  
    prodotto = db.relationship('Prodotto')
```

Definizione classe (tabella) Recensione

```
class Recensione(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    prodotto_id = db.Column(db.Integer, db.ForeignKey('prodotto.id'), nullable=False)  
    user_id = db.Column(db.Integer, db.ForeignKey('utenti.id'), nullable=False)  
    voto = db.Column(db.Integer, nullable=False)  
    data_recensione = db.Column(db.DateTime, default=datetime.now, nullable=False)  
    prodotto = db.relationship('Prodotto', backref='recensioni')  
    user = db.relationship('Utenti', backref='recensioni')
```

3.3 Associazioni

In questa sezione vengono descritte le associazioni (relazioni tra le tabelle) scelte per modellare la base di dati:

Prodotti <<— Vengono acquistati da —>> Acquirenti

Associazione di tipo M:M che permette ai Prodotti di essere acquistati dagli Acquirenti.

Prodotti <<— Vengono messi in vendita da —> Venditori

Associazione di tipo M:1 che permette ai Prodotti di essere messi in vendita dai Venditori.

Prodotti <<— Vengono inseriti in —>> Carrelli

Associazione di tipo M:M che permette ai Prodotti di essere inseriti nei Carrelli.

Prodotti <<— Fanno parte di —>> Ordini

Associazione di tipo M:M che permette ai Prodotti di far parte degli Ordini.

Prodotti <— Vengono valutati tramite —>> Recensioni

Associazione di tipo 1:M che permette alle Recensioni di essere associate ai Prodotti.

Recensioni <<— Vengono scritte da —> Acquirenti

Associazione di tipo M:1 che permette alle Recensioni di essere scritte dagli Acquirenti.

Ordini <<— Vengono creati da —> Acquirenti

Associazione di tipo M:1 che permette agli Ordini di essere creati dagli Acquirenti.

Acquirenti <— Possiedono —> Carrelli

Associazione di tipo 1:1 che permette ai Carrelli di essere posseduti dagli Acquirenti.

3.4 Modello Logico Relazionale

In Figura 10 il modello logico relazionale:

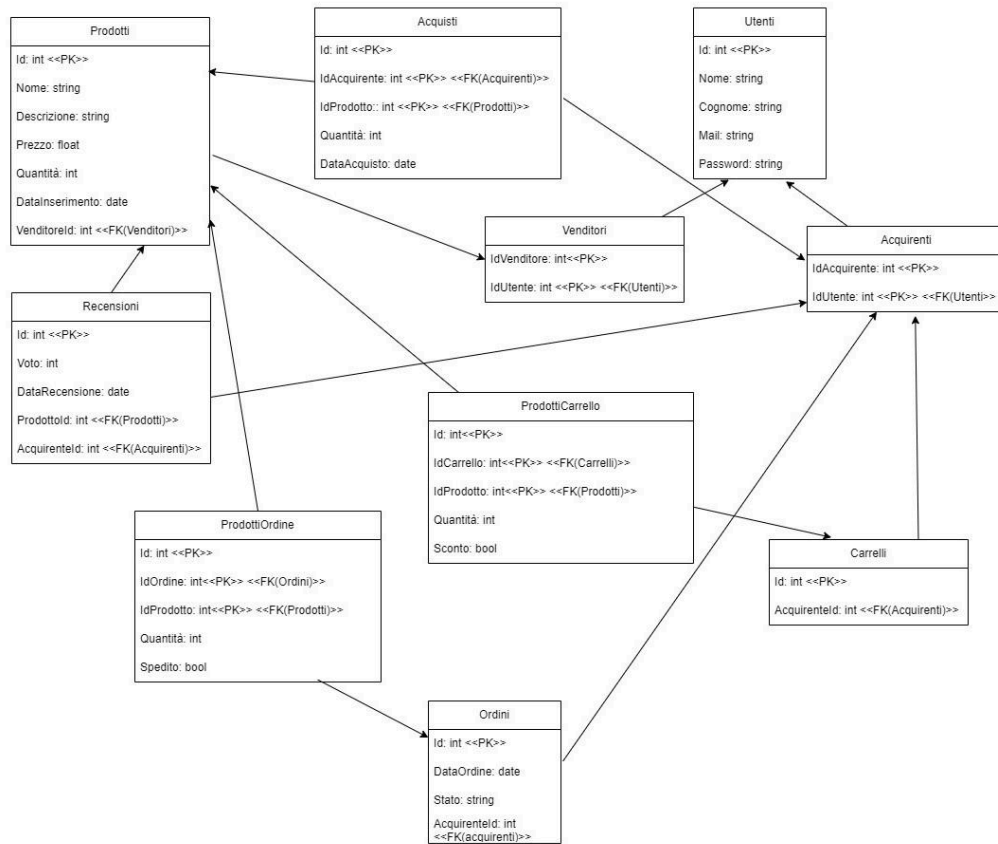


Figura 10: Modello logico relazionale

4 Query

In questo Paragrafo vengono descritte le query utilizzate, tradotte in linguaggio SQL:

```
SELECT *  
FROM Acquisto  
WHERE user_id = <current_user_id>;
```

Recupera tutti gli acquisti effettuati dall'utente corrente.

```
SELECT *  
FROM Prodotti  
WHERE user_id = <current_user_id>;
```

Recupera tutti i prodotti messi in vendita dall'utente corrente.

```
SELECT *  
FROM Utenti  
WHERE username = '<username>'  
LIMIT 1;
```

Recupera il record dell'utente dalla tabella Utenti che ha un determinato username.

```
SELECT *  
FROM Utenti  
WHERE username = '<email>'  
LIMIT 1;
```

Verifica se esiste già un utente con l'indirizzo email fornito (utilizzato come username).

```
SELECT *  
FROM Recensione  
WHERE prodotto_id = <prodotto_id>;
```

Recupera tutte le recensioni associate ad un determinato prodotto, identificato dal suo prodotto_id.

```
SELECT *  
FROM Recensione  
WHERE prodotto_id = <prodotto_id> AND user_id = <user_id>  
LIMIT 1;
```

Verifica se esiste già una recensione di un determinato prodotto fatta da un determinato utente, restituendo solo il primo risultato.

```
INSERT INTO Utenti (username, password, ruolo)  
VALUES ('<email>', '<hashed_password>', 'Venditore');
```

Inserisce un nuovo utente nella tabella Utenti con ruolo Venditore.


```
INSERT INTO Venditore (nome, cognome, user_id)
VALUES ('<nome>', '<cognome>', <user_id>);
```

Inserisce i dettagli del venditore nella tabella Venditore, collegandolo all'utente appena creato tramite user_id.

```
INSERT INTO Utenti (username, password, ruolo)
VALUES ('<email>', '<hashed_password>', 'Acquirente');
```

Inserisce un nuovo utente nella tabella Utenti con ruolo Acquirente.

```
INSERT INTO Acquirente (nome, cognome, user_id)
VALUES ('<nome>', '<cognome>', <user_id>);
```

Inserisce i dettagli dell'acquirente nella tabella Acquirente, collegandolo all'utente appena creato tramite user_id.

```
SELECT *
FROM Prodotto
WHERE id = <prodotto_id>
LIMIT 1;
```

Recupera un prodotto per id.

```
SELECT *  
FROM Carrello  
WHERE user_id = <user_id>  
LIMIT 1;
```

Recupera il carrello di un utente.

```
SELECT *  
FROM CarrelloProdotto  
WHERE carrello_id = <carrello_id>  
    AND prodotto_id = <prodotto_id>  
LIMIT 1;
```

Recupera uno specifico prodotto da uno specifico carrello.

```
SELECT *  
FROM CarrelloProdotto  
WHERE carrello_id = <carrello_id> AND prodotto_id = <prodotto_id>  
LIMIT 1;
```

Inserisce un nuovo prodotto nel carrello.

```
UPDATE CarrelloProdotto  
SET quantita = quantita + <quantita>  
WHERE carrello_id = <carrello_id> AND prodotto_id = <prodotto_id>;
```

Aggiorna la quantità di un prodotto nel carrello.

```
SELECT *  
FROM Prodotto  
WHERE venditore_id = <venditore_id>;
```

Recupera i prodotti venduti da uno specifico venditore.

```
SELECT DISTINCT o.*  
FROM Ordine o  
JOIN OrdineProdotto op ON o.id = op.ordine_id  
WHERE op.prodotto_id IN (<prodotti_venditore_ids>);
```

Recupera tutti gli ordini che includono i prodotti di uno specifico venditore.

```
UPDATE Ordine  
SET stato = CASE  
    WHEN tutti_spediti THEN 'spedito'  
    ELSE 'in elaborazione'  
END  
WHERE id = <ordine_id>;
```

Aggiorna lo stato di un ordine.

```
SELECT *  
FROM Prodotto  
WHERE nome ILIKE '%' || <nome> || '%'  
AND prezzo >= <prezzo_min> AND prezzo <= <prezzo_max>;
```

Recupera i prodotti in base al nome e al prezzo.

```
SELECT *  
FROM Recensione  
WHERE prodotto_id = <prodotto_id>  
AND user_id = <user_id>  
LIMIT 1;
```

Recupera la recensione di uno specifico prodotto di uno specifico utente.

```
INSERT INTO Recensione (prodotto_id, user_id, voto)  
VALUES (<prodotto_id>, <user_id>, <voto>);
```

Inserisce una nuova recensione.

```
UPDATE Recensione  
SET voto = <voto>  
WHERE prodotto_id = <prodotto_id>  
AND user_id = <user_id>;
```

Aggiorna una recensione già esistente.

5 Principali scelte progettuali

In questo Paragrafo vengono descritte le principali scelte progettuali.

E' stato innanzitutto creato un database PostgreSQL:

```
CREATE DATABASE progetto;
```

Per la creazione delle tabelle è necessario eseguire l'applicativo:

```
export FLASK_APP=main.py  
flask run
```

Sono stati quindi creati nel database tre utenti con compiti diversi, amministratore, acquirente e venditore:

```
\c progetto  
CREATE ROLE amministratore WITH PASSWORD 'password';  
CREATE ROLE acquirente WITH PASSWORD 'password';  
CREATE ROLE venditore WITH PASSWORD 'password';
```

A questi utenti viene poi data la possibilità di effettuare il login:

```
ALTER ROLE amministratore WITH LOGIN;  
ALTER ROLE acquirente WITH LOGIN;  
ALTER ROLE venditore WITH LOGIN;
```

Vengono infine aggiunti i trigger:

```
\i trigger.sql
```

L'utente amministratore serve per creare gli utenti e fare il login, quindi ha un accesso limitato solo alle tabelle Utente, Acquirente e Venditore.

Abbiamo discusso anche la possibilità di creare come primo utente alla prima esecuzione dell'applicazione, un utente amministratore, il quale poi avrebbe concesso il ruolo venditore su specifica richiesta, mentre tutti gli altri ruoli di default sarebbero stati acquirenti.

Il primo collegamento sarebbe stato indicato da un flag in una apposita tabella di sistema.

Si è optato invece per la scelta del ruolo in fase di registrazione utente visto lo scopo didattico del progetto.

L'utilizzo dei Blueprint (acquirente, venditore e logica) ci ha permesso di organizzare le rotte e la logica dell'applicazione in moduli separati.

Questo ci ha aiutato a mantenere il codice ben strutturato e modulare.

Ogni Blueprint gestisce quindi le funzionalità specifiche per acquirenti, venditori e logica di base dell'applicazione.

Abbiamo utilizzato Flask-Login per gestire l'autenticazione (login_user, logout_user e login_required).

Questo ci ha permesso di mantenere l'utente autenticato tra le richieste e di gestire diverse sessioni utente.

La verifica dei ruoli (Acquirente e Venditore) e dei permessi di accesso ai dati e alle azioni (ad esempio il controllo dell'accesso per eliminare o modificare i prodotti) è stata implementata tramite la logica all'interno delle rotte.

Abbiamo utilizzato sessionmaker per creare sessioni con SQLAlchemy e gestire le transazioni.

In caso di errore (ad esempio IntegrityError), le transazioni vengono annullate (session.rollback), garantendo la coerenza dei dati.

Avendo utilizzato database separati per amministratori, acquirenti e venditori, diverse sessioni sono state utilizzate per ruoli specifici (get_autho_session, get_venditore_session e get_acquirente_session).

Le classi Utenti, Acquirente, Venditore, Prodotto, Carrello, CarrelloProdotto, Acquisto, Ordine, OrdineProdotto, e Recensione definiscono le tabelle del database.

Le relazioni tra le tabelle (ad esempio backref e relationship) sono definite per gestire le associazioni tra le diverse classi visibili graficamente alla Sezione 3.1 e commentate alla Sezione 3.3.

Funzioni come ad esempio crea_venditore, crea_acquirente, login_utente, e registrazione_utente gestiscono la logica di business per creare e autenticare utenti.

Queste funzioni includono la logica per la validazione delle password e dei dati degli utenti.

La validazione dei dati di input viene eseguita in diverse rotte (ad esempio nella validazione del prezzo e della quantità durante la creazione e la modifica dei prodotti).

Abbiamo utilizzato flash per fornire feedback all'utente riguardo al successo o ai problemi durante le operazioni (ad esempio nella creazione di un prodotto, nel login e nella registrazione di un nuovo utente).

Gli errori sono gestiti e comunicati all'utente attraverso messaggi di errore flash e conseguente redirectione.

Le funzionalità per aggiungere e rimuovere prodotti dal carrello, visualizzare il carrello e calcolare il totale del carrello sono implementate in rotte specifiche.

Questo ci ha permesso (e permetterebbe di raffinare ulteriormente) la gestione della spesa e dell'inventario.

La ricerca dei prodotti è gestita tramite la rotte /ricerca_prodotto e /ricerca_prodotti, permettendo agli utenti di cercare prodotti basati su vari criteri (ad esempio nome e prezzo).

Infine Il codice è stato suddiviso in moduli e funzioni ben definiti, aggiungendo commenti dove necessario, favorendone così la leggibilità e la manutenibilità.

Ogni parte dell'applicazione ha un compito specifico e le responsabilità sono chiaramente separate.

5.1 Autorizzazioni

In questa Sezione vengono descritte le autorizzazioni che devono essere inserite nel database per poter eseguire l'applicazione:

```
GRANT INSERT, SELECT ON Utente, Acquirente, Venditore TO amministratore;
```

Permette all'utente amministratore di eseguire query di inserimento e lettura sulle tabelle Utente, Acquirente e Venditore.

```
GRANT ALL PRIVILEGES ON SEQUENCE utenti_id_seq TO amministratore;
```

Concede all'utente amministratore tutti i diritti sulla sequenza utenti_id.

```
GRANT ALL PRIVILEGES ON SEQUENCE acquirente_id_seq TO amministratore;
```

Concede all'utente amministratore tutti i diritti sulla sequenza acquirente_id.

```
GRANT ALL PRIVILEGES ON SEQUENCE venditore_id_seq TO amministratore;
```

Concede all'utente amministratore tutti i diritti sulla sequenza venditore_id.

5.2 Trigger

In questa Sezione vengono descritti i trigger che sono stati utilizzati.

Il primo trigger permette di applicare uno sconto per quantità:

```
CREATE OR REPLACE FUNCTION applica_sconto()
RETURNS TRIGGER AS $$
BEGIN
    -- Verifica se la quantità è maggiore di 1 e se lo sconto non è già stato applicato
    IF NEW.quantita > 1 AND NOT NEW.sconto THEN
        -- Imposta il flag dello sconto su TRUE
        NEW.sconto := TRUE;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_applica_sconto
BEFORE INSERT OR UPDATE ON carrello_prodotto
FOR EACH ROW
EXECUTE FUNCTION applica_sconto();
```

Il secondo trigger permette di eliminare lo sconto nel caso la quantità di prodotti da acquistare sia tornata all'unità:

```
CREATE OR REPLACE FUNCTION rimuovi_sconto()
RETURNS TRIGGER AS $$
BEGIN
    -- Verifica se la quantità è diventata minore di 2 per rimuovere lo sconto
    IF NEW.quantita < 2 THEN
        -- Imposta il flag dello sconto su FALSE
        NEW.sconto := FALSE;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigger_rimuovi_sconto
BEFORE UPDATE ON carrello_prodotto
FOR EACH ROW
WHEN (OLD.quantita >= 2 AND NEW.quantita < 2)
EXECUTE FUNCTION rimuovi_sconto();
```

6 Ulteriori informazioni

Per la parte di autenticazione è stata utilizzata la libreria flask_login.

Per la parte grafica è stato utilizzato il framework Bootstrap.

In tutte le funzioni, per garantire la robustezza del codice, è stato utilizzato il costrutto try-except-finally.

6.1 Comandi utili

Di seguito un riassunto dei comandi necessari all'utilizzo dell'applicazione.

Creazione del database:

```
# sudo -u postgres psql
postgres=# CREATE DATABASE progetto;
postgres=# \q
```

Cancellazione del database:

```
# sudo -u postgres psql
postgres=# DROP DATABASE progetto;
postgres=# \q
```

Cambio password di default:

```
postgres=# ALTER USER postgres PASSWORD 'Venezia123';
```

Elenco dei database ed accesso:

```
# sudo -u postgres psql
postgres=# \l
postgres=# \c progetto
progetto=# \q
```

Riavvio PostgreSQL

```
# service postgresql restart
```

Esecuzione applicazione:

```
# export FLASK_APP=main.py
```

```
# flask run
```

In alternativa per l'esecuzione dell'applicazione:

```
# python3 main.py
```

Cancellazione tabelle:

```
# sudo -u postgres psql
```

```
postgres=# \c progetto
```

```
progetto=# DELETE from utenti;
```

```
progetto=# DELETE from acquirente;
```

```
progetto=# DELETE from venditore;
```

```
progetto=# \q
```

7 Contributo al progetto

Massimo Costantini - 877336

Sviluppo codice (autenticazione)

Documentazione

Samuel Obeng Takyi - 881431

Sviluppo codice (acquirente)

Documentazione

Filippo Tiozzo - 887683

Sviluppo codice (venditore)

Documentazione