Progetto BASI DI DATI

Cusano Gabriele (897835), Zambon Davide (898103)

Introduzione

Il progetto che presentiamo è una web application che si interfaccia a un database relazionale per implementare tutte le funzionalità di base di un sito di e-commerce. Ci siamo ispirati principalmente al funzionamento di subito.it, in cui ogni utente compra e vende liberamente. Il sito offre un'interfaccia grafica minimale che però cerca di garantire una navigazione intuitiva all'interno dell'applicativo in modo da potersi concentrare principalmente sullo sviluppo delle funzionalità implementate al lato server.

Descrizione delle pagine

La struttura del sito è comune a ogni pagina: header e contenuto principale

L'header contiene, da sinistra a destra:

- Logo del sito: è cliccabile e permette di fare sempre riferimento alla homepage da qualsiasi parte del sito
- Barra di ricerca: indirizza l'utente a un form che permette di eseguire la ricerca dei prodotti all'interno del sito tramite filtri e parole chiave.
- Pulsante di Login: permette l'autenticazione al sito previo inserimento di credenziali.
- Pulsante Sign In: consente a un nuovo utente di registrarsi al sistema. Nel momento in cui si effettua l'accesso al sito, questo pulsante viene cambiato in quello di Logout, che serve per terminare la sessione

Il contenuto principale è specifico per ogni pagina.

Homepage

Fornisce la lista globale di tutti i prodotti in vendita disponibili. Vengono visualizzate le informazioni base: titolo del prodotto (cliccabile), descrizione riassuntiva del prodotto, eventuale immagine principale e prezzo. Inoltre, la pagina fornisce un pulsante per filtrare tutti i prodotti in modo equivalente alla barra di ricerca nell'header.

Bacheca personale

Si accede alla bacheca personale una volta eseguito il Login. In questa pagina si possono visualizzare le informazioni utente inserite al momento dell'iscrizione e una lista di operazioni che variano in base al ruolo con cui un utente si è iscritto.

Con il ruolo da cliente l'utente avrà accesso a queste funzionalità:

- Eseguire il cambio dell'email inserita dall'utente
- Eseguire il cambio del numero di telefono
- Accedere alla lista dei prodotti presenti nel carrello acquisti
- Accedere allo storico dei prodotti acquistati.
- Abilitare il profilo al ruolo di Venditore
- Eliminare il proprio account.

Solo nel momento in cui un utente abilita il proprio profilo al ruolo venditore potrà:

- visualizzare la lista degli annunci inseriti
- inserire un nuovo annuncio per vendere un prodotto.

Pagina carrello acquisti

Ci si arriva dalla bacheca personale. Viene mostrata la lista dei prodotti che si desidera acquistare. La scheda mostrata è simile a quella della pagina principale. È infatti possibile cliccare sui titoli per visualizzare la scheda del prodotto. In aggiunta viene visualizzata la quantità di ogni prodotto nel carrello, il costo totale e due pulsanti: il primo per rimuovere un prodotto dal carrello e il secondo per procedere al pagamento

Pagina storico acquisti

Ci si arriva dalla bacheca personale. Viene mostrato lo storico dei prodotti acquistati dal momento della registrazione dell'utente al sito. Le informazioni visualizzate sono titolo e data di acquisto. In caso di prodotti ancora disponibili, il titolo è cliccabile. Vengono comunque visualizzati tutti gli annunci, compresi quelli eliminati o non disponibili a magazzino.

Modulo per inserire gli annunci

A partire dalla bacheca personale è possibile accedere a una pagina contenente un modulo che permette all'utente di inserire tutte le informazioni relative al prodotto da vendere.

Pagina del prodotto

A questa pagina si accede nel momento in cui si clicca sul titolo del prodotto che si desidera visualizzare. La scheda quindi fornisce tutte le informazioni inserite dal venditore: titolo, descrizione, eventuali immagini, metodo di pagamento, possibilità di

spedizione (si/no), quantità disponibile in vendita, autore dell'annuncio, data di pubblicazione e condizione (nuovo o di seconda mano).

Da questa pagina è possibile aggiungere l'articolo al carrello (selezionando la quantità), scriverne una recensione, visualizzare le recensioni già presenti e, nel caso l'utente sia proprietario dell'annuncio, rimuoverlo o modificarlo.

Modulo Inserimento Recensione

Dalla pagina prodotto è possibile aggiungere una recensione compilando il modulo in cui viene richiesto di inserire il titolo, il numero di stelle fino a un massimo fino a 5 e una descrizione utile a spiegare agli altri utenti cosa si pensa del prodotto.

Pagina Recensioni

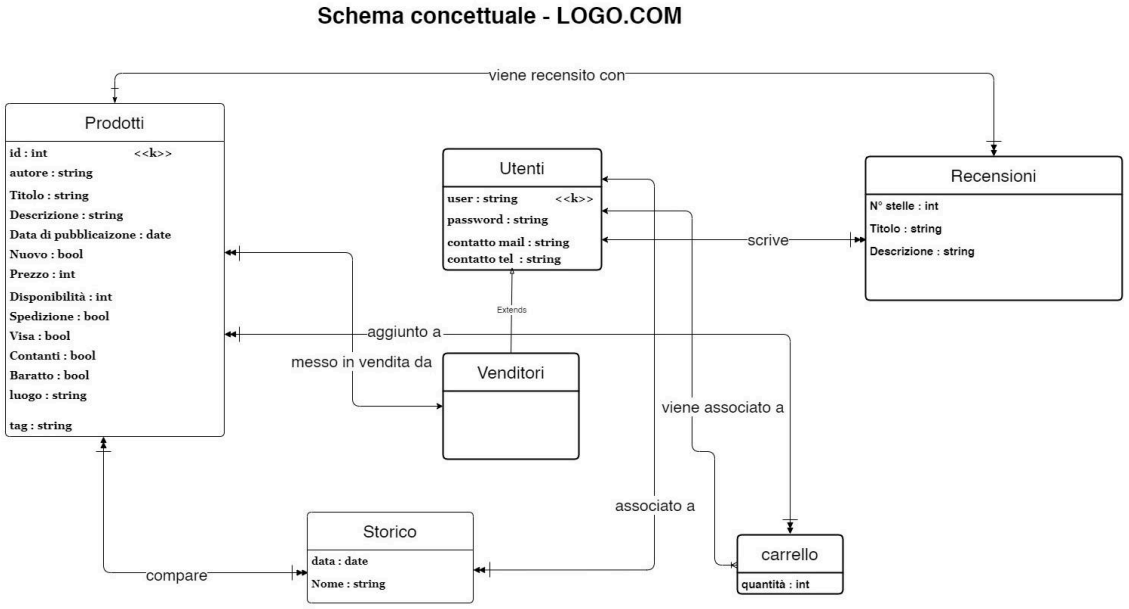
Dalla pagina prodotto è possibile visualizzare la lista delle recensioni relative a quel prodotto e vederne l'indice di gradimento.L'utente può anche ordinare le recensioni per ordine di pubblicazione e per ordine di valutazione, in modo sia crescente sia decrescente

Pagina Login e Signin

Dalla pagina login si effettua l'accesso al sito, inserendo le proprie credenziali. Dalla pagina signin si può creare un nuovo utente. Durante la registrazione, l'utente assume automaticamente il ruolo di cliente, con la possibilità però di flaggare l'opzione per assumere il ruolo di venditore. In questo modo, oltre a poter effettuare acquisti, l'utente avrà anche la possibilità di inserire nuovi annunci nel sistema.

Schema concettuale e modello relazionale

Schema Concettuale



Classi

Partendo dallo schema concettuale,abbiamo deciso di modellare le informazioni necessarie attraverso sei classi tra cui una è una sottoclasse. Le classi principali sono: Utenti, Prodotti, Recensioni, Carrello, Storico. Invece la sottoclasse è Venditori la quale eredita gli attributi da Clienti.

Utenti: modella le informazioni che servono a descrivere un utente con i seguenti attributi:

- User: nome identificativo di un utente (nessun utente può avere lo stesso nome)
- Password: necessaria insieme ad utente come credenziale per accedere alle informazioni personali.
- Email: mail di un utente.
- Telefono: recapito telefonico di un utente

Un utente iscritto nel sistema assume automaticamente il ruolo di cliente.

Venditori: Sottoclasse di Utenti, modella la possibilità aggiuntiva di inserire degli annunci all'interno del sito e poter visualizzare la lista di tutti quelli da loro pubblicati.

Prodotti: Contiene tutte le caratteristiche necessarie per descrivere il prodotto in vendita:

- id: codice identificativo univoco che associa ad ogni id uno e un solo prodotto.
- Autore: nome dell'autore che ha pubblicato l'annuncio.
- Titolo: nome del prodotto
- Descrizione: descrizione del prodotto
- Data di pubblicazione: data di pubblicazione dell'annuncio
- Condizione: nuovo o di seconda mano
- Prezzo: costo unitario
- Disponibilità: quantità che un utente dichiara di voler mettere in vendita
- Spedizione: disponibilità del venditore a spedire il prodotto
- Visa: disponibilità di pagamento elettronico
- Contanti: disponibilità di pagamento in contanti.
- Baratto: disponibilità di pagamento tramite baratto del prodotto.
- Luogo: luogo di provenienza dichiarato dal venditore (nel caso non fosse disponibile la spedizione)
- Tag: categoria del prodotto (in caso il venditore ritenesse che il prodotto non appartiene a nessuna categoria di default il prodotto viene segnato con il tag "altro".)

Recensioni: Rappresenta tutte le recensioni pubblicate di ogni rispettivo prodotto :

- N° stelle : rappresenta il livello di gradimento (da 1 a 5) di un utente.
- Titolo : Nome riassuntivo del giudizio su un prodotto
- Descrizione : commento sul prodotto.

Carrello: Rappresenta tutta la lista di prodotti che un utente inserisce nel suo carrello personale, congiuntamente alla quantità selezionata.

Storico: Rappresenta tutta la lista di prodotti che sono stati acquistati da un utente. Altre informazioni aggiuntive sono :

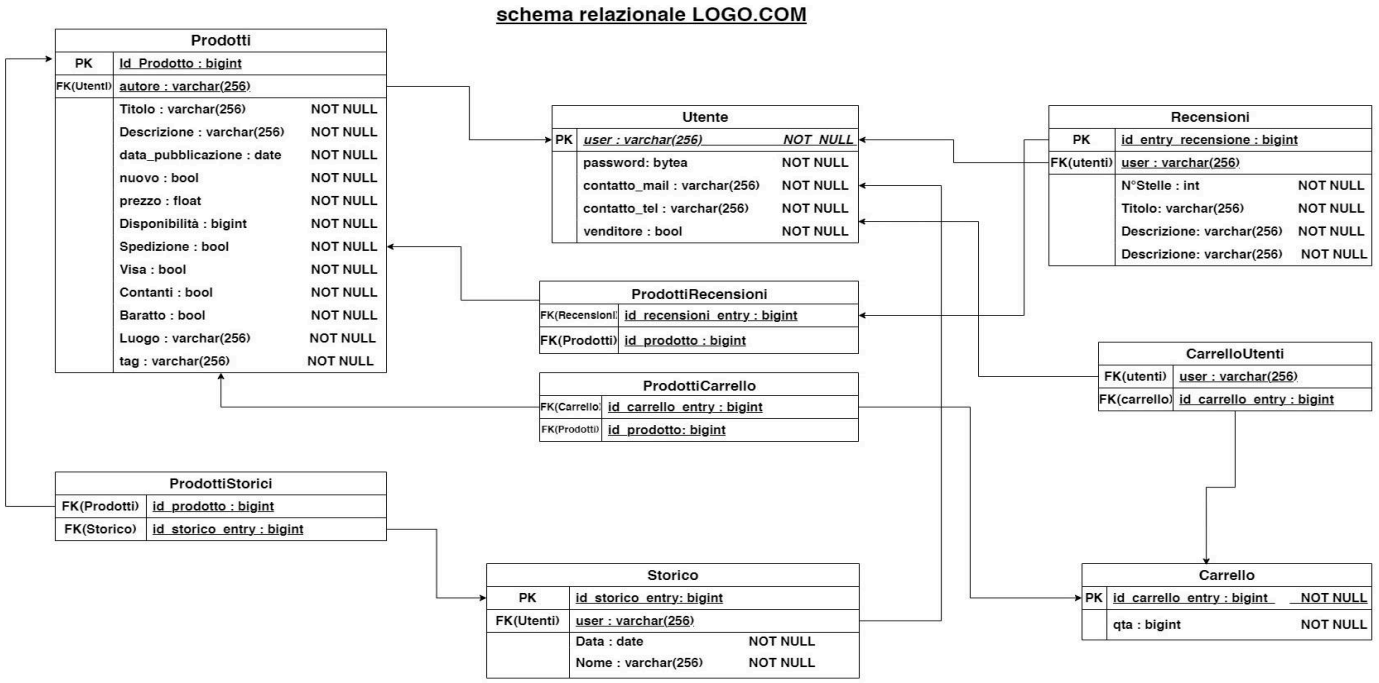
- Data: data in cui è stato acquistato un prodotto (in particolare, data del pagamento)
- Nome: Titolo originale del prodotto al momento dell'acquisto.

Associazioni

- Un Utente può essere associato a molte o nessuna voce dello Storico: ciò riflette la cronologia degli acquisti effettuati.
- Un Utente può essere associato a molte o nessuna voce del Carrello: rappresenta i prodotti che l'utente ha aggiunto per un potenziale acquisto.
- Un Utente può scrivere molte o nessuna recensione.

- Molte Recensioni o nessuna posso recensire un prodotto
- Molti Prodotti o nessuno possono essere aggiunti a molte o a nessuna voce del Carrello
- Molti Prodotti o nessuno possono comparire in molte o nessuna voce dello Storico.
- Un Venditore può mettere in vendita molti o nessun prodotto.

Schema Relazionale



Dallo schema concettuale proponiamo una sua possibile conversione in schema relazionale. È costituito da nove tabelle, di cui cinque sono state mantenute dallo schema precedente e quattro sono state aggiunte per modellare le associazioni molti-a-molti e uno-a-molti.

I cambiamenti significativi apportati con la traduzione sono principalmente nella tabella utenti in cui si cambia il tipo della password (da stringa di caratteri diventa una stringa di bit necessaria per memorizzare la password hashata). Altra differenza è nell'introduzione di un booleano "venditore" che serve per modellare il fatto che un utente sia un venditore o meno.L'aggiunta finale consiste nelle foreign key necessarie per rappresentare le associazioni descritte in precedenza.

Sviluppo dell’applicazione e scelte progettuali

Struttura del codice

Il codice python è suddiviso in più sorgenti.

- app.py: codice base per creare e avviare l'applicazione
- sqlclass.py: codice con cui si creano le tabelle e ci si connette al database
- extention.py: variabili globali, librerie e funzioni comuni
- cart_and_history.py: funzioni relative al carrello e allo storico: visualizzazione, aggiunta e rimozione prodotti etc.
- product.py: funzioni relative alla gestione dei prodotti: visualizzazione, creazione, eliminazione e modifica
- reviews.py: funzioni relative alle recensioni: visualizzazione e creazione
- search_and_filter.py: funzionalità di ricerca degli annunci, con relativa applicazione dei filtri
- user.py: funzioni relative all'utente: login, logout, creazione ed eliminazione account, modifica dell'anagrafica, visualizzazione dei propri dati e dei propri annunci etc.

Connessione al DB

Per la connessione al DB abbiamo optato per usare la funzionalità di automap di sqlalchemy, così da creare “on the fly” un modello ad oggetti relativo al database già esistente, con corrispondenza 1 a 1 tra le classi e le tabelle. L’accesso ai dati avviene poi tramite reflection.
Le query e gli statement relativi al DB sono tutti eseguiti tramite l’ORM della libreria, così da gestire in modo automatizzato la sicurezza e sanitizzazione degli input, e da avere un linguaggio SQL-equivalente a più alto livello.

Utenti e ruoli nel database

Gli utenti sono differenziati tra clienti e venditori tramite un attributo booleano nella tabella Utenti nel Database. Gli utenti hanno accesso con username (univoco, chiave primaria della tabella) e password (memorizzata hashata concatenata a un salt). L’utente può diventare venditore tramite un apposito collegamento nella bacheca. Da quel momento è in grado di pubblicare e gestire annunci.
Gli utenti non hanno accesso diretto al database, quindi ad ogni utente dell’applicazione non corrisponde un utente del database. La gestione degli utenti è infatti interamente delegata alla logica dell’app, che si occupa di autenticare e autorizzare gli utenti in base alla necessità. Ad esempio, nel caso di modifica a un

annuncio di vendita, è direttamente il server, nel codice python, a verificare che il "current user" sia l'autore dell'annuncio e a permettergli o meno di effettuare modifiche.
Nel contesto dell'applicazione, non abbiamo ritenuto quindi necessario definire due ruoli "compratore" e "venditore" all'interno del database, essi sono differenziati solo tramite attributi delle tabelle. Gli utenti non hanno accesso diretto al database, il quale serve come mero archivio dati.
Il ruolo che invece abbiamo ritenuto necessario creare all'interno del database è quello con cui il server si connette. È un utente con i permessi di vedere, creare, modificare, eliminare i dati ed eseguire i trigger della tabella. Non ha invece i permessi per modificare la struttura del database e delle tabelle.

Ricerca e Filtri

La ricerca può essere effettuata per parole (ricercate nel titolo e nella descrizione). I risultati possono essere ristretti tramite l'applicazione di filtri, in particolare si può filtrare per range di date, range di prezzo, condizione dell'articolo, metodi di pagamento accettati e disponibilità alla spedizione.
I filtri sono gestiti, a livello di codice, tramite una lista che contiene le varie condizioni, che vengono messe in "and logico" tra loro nel "where" della query. In questo modo la query è una sola e le condizioni variano dinamicamente.

Immagini

Durante l'upload del prodotto, possono essere caricate fino a 5 immagini. Di esse non vi è traccia nel database. Abbiamo infatti ritenuto più efficiente salvarle in una cartella nel server. Nemmeno il relativo path è presente nel database, in quanto l'informazione sarebbe stata ridondante, portando a inefficienze. Esso è infatti costruito a partire dalle informazioni già contenute nei dettagli del prodotto, ovvero da username dell'autore e id dell'annuncio, in questo modo:
`[path della cartella]/username/id/[immagini]`
Viene effettuato un controllo sulle estensioni. Sono ammessi solo file di tipo png, jpg, gif e jpeg. Non è obbligatorio caricare un'immagine, in tal caso infatti nella pagina dell'annuncio viene mostrato un "placeholder" (un immagine stock di un pacco).

Gestione dei prodotti e viste

Nella pagina di upload dei prodotti vengono effettuati direttamente dal front end (nei "form" HTML) dei controlli sull'accettabilità dei dati (ad esempio prezzo >0). Inizialmente, al momento del check-out nel carrello, avevamo pensato di eliminare tramite trigger gli articoli con disponibilità 0. Poi siamo tornati sui nostri passi, per

permettere al venditore di vedere i suoi annunci nonostante non fossero più disponibili. Di conseguenza devono essere eliminati manualmente, o al limite vengono eliminati in automatico quando l'autore elimina il suo account. Non avendo però senso mostrare sulla homepage anche annunci senza disponibilità a magazzino, abbiamo creato una vista nel database. Ogni volta che è necessario recuperare una lista di prodotti (per esempio nella homepage o come risultato di una ricerca) viene effettuata la query sulla vista al posto che sulla tabella prodotti.

Storico

Lo storico acquisti contiene solo informazioni base sui prodotti acquistati: titolo e data d'acquisto. Abbiamo pensato a questa soluzione per permettere di visualizzare anche prodotti non più presenti nel database. Infatti, dallo storico è possibile risalire, tramite la tabella intermedia "ProdottiStorici", all'id del prodotto. Se quest'ultimo esiste ancora, cliccando sul titolo si viene reindirizzati alla pagina relativa contenente tutte le informazioni, altrimenti il titolo non è cliccabile, ma il compratore ha comunque traccia di cosa ha comprato.

Query e trigger principali

Query

Carrello

Nella query vengono selezionate le voci del carrello relative al prodotto e che sono state inserite allo stesso tempo dall'utente.

query in python:

```python
carrelloutente = (
   s.query(c, p)
   .join(cu, c.id_carrello_entry == cu.id_carrello_entry)
   .join(pc, c.id_carrello_entry == pc.id_carrello_entry)
   .join(p, pc.id_prodotto == p.id_prodotto)
   .filter(cu.user == actualUser)
   .all()
)
```

Traduzione in postgresql:

```
Unset
select c.*,p.*
from Carrello as c
join  Carrello_Utenti as cu on c.id_carrello_entry = cu.id_carrello_entry
join ProdottiCarrello as pc on c.id_carrello_entry = pc.id_carrello_entry
join Prodotti as p on p.id_prodotto = pc.id_prodotto
where cu.user = “utente che che vuole accedere al suo carrello”.
```

Aggiornamento inventariale

Quando si procede col checkout, per ogni prodotto nel carrello viene fatto l’update della quantità disponibile a magazzino

Query in python:

```
Python
for carrello, prodotto in carrelloutente:
   if carrello.qta <= prodotto.disponibilita:
       stmt = (
              update(Prodotti)
              .where(Prodotti.id_prodotto == prodotto.id_prodotto)
              .values(disponibilita=Prodotti.disponibilita - carrello.qta)
       )
       s.execute(stmt)
   else:
       insufficient_stock.append(prodotto.Titolo)
```

Traduzione in postgresql:

```
Unset
UPDATE Prodotti
SET disponibilita = disponibilita - (
       SELECT COALESCE(SUM(c.quantita), 0)
       FROM Carrello c
       JOIN ProdottiCarrelli pc ON c.id_carrello_entry = pc.id_carrello_entry
       WHERE pc.id_prodotto = id_da_modificare
)
WHERE id_prodotto = id_da_modificare;
```

Qui al posto di sottrarre le quantità relative parzialmente vengono sottratte in una sola volta eseguendo la somma di tutte le quantità e sottraendole infine con la disponibilità in magazzino e si assume che la operazione di update finale sia valida(la disponibilità alla fine non può mai essere minore di zero)

Media Recensioni

Query per calcolare la media delle recensioni di un prodotto

Query in python

```python
media = s.query(func.avg(r.numero_stelle)).join(pr, r.id_recensione_entry == pr.id_recensione_entry).filter(pr.id_prodotto == id).scalar()
```

Traduzione in Postgresql

```
select avg(r.numero_stelle)
from Recensioni r
join ProdottiRecensioni pr on r.id_recensione_entry = pr.id_recensione_entry
where pr.id_prodotto = id
```

Filtri

Questa query è interessante a livello di codice python, in quanto utilizza la funzione di sqlalchemy “_and”. I filtri sono inseriti in una lista e la query viene eseguita sempre in modo diverso, applicando quelli necessari. In questo modo la query è una sola e le condizioni variano dinamicamente.

```python
def filters (min_price, max_price, min_date, max_date, condition, payment_methods, sped):
	filt = []
	if min_price:
	if double(min_price) >= DOUBLEMIN:
		filt.append(Prodotti.Prezzo >= double(min_price))
	if max_price:
```

```
        if double(max_price) <= DOUBLEMAX:
                filt.append(Prodotti.Prezzo <= double(max_price))
        if min_date:
                filt.append(Prodotti.data_pubblicazione >= min_date)
        if max_date:
                filt.append(Prodotti.data_pubblicazione <= max_date)
        if condition and condition != '':
        if condition == 'new':
                filt.append(Prodotti.nuovo == True)
        else:
                filt.append(Prodotti.nuovo == False)
        if payment_methods and '' not in payment_methods:
        if 'visa' in payment_methods:
                filt.append(Prodotti.visa == True)
        if 'cash' in payment_methods:
                filt.append(Prodotti.contanti == True)
        if 'barter' in payment_methods:
                filt.append(Prodotti.baratto == True)
        if sped and sped != '':
        if sped == "new":
                filt.append(Prodotti.spedizione == True)
        else :
                filt.append(Prodotti.spedizione == False)
        return  filt
# ...
ris = filters(min_price, max_price, min_date, max_date, condition,
payment_methods, sped)
prodotti = s.query(prodotti_disponibili).where(and_(*ris)).all()
```

Trigger

I trigger presenti nel database sono tre e si occupano principalmente di eliminare istanze non più valide per la integrità dei dati nelle tabelle.
Questi trigger vengono attivati solo dopo un'operazione di delete ed esegue una procedura per ogni riga.

Elimina Carrelli Orfani

questo trigger viene attivato nel momento in cui un utente elimina un prodotto dalla voce dei carrelli. La procedura che viene eseguita cancella le righe da ProdottiCarrelli e da Carrelli.

```
Unset
create function elimina_carrelli_orfani() return trigger as $$
BEGIN
	If not exists(
		select 1 from public.”ProdottiCarrelli” where id_carrello_entry
		= OLD.id_carrello_entry)
	AND if not exists(
		select 1 from public.”CarrelloUtenti” where id_carrello_entry =
		OLD.id_carrello_entry)
	Then
		delete from public.”Carrello”
		where id_carrello_entry= OLD.id_carrello_entry
	end if;
	Return OLD;
End;
$$ language plpgsql;

create trigger trigger_elimina_carreli_da_utenti after delete on
"CarrelliUtenti” for each row execute procedure elimina_carrelli_orfani();
```

Elimina recensioni orfane

Viene attivato nel momento in cui si decide di cancellare una riga dalla tabella “Prodotti” e serve a cancellare tutte le recensioni associate.

```
Unset
create function elimina_recensioni_orfane() returns trigger as $$
BEGIN
	Delete from public.”ProdottiRecensioni” Where id_prodotto is NULL;
	Delete from public.”Recensioni” Where id_recensione_entry not in(
		select id_recensione_entry From public.”ProdottiRecensioni”);
	Return NULL;
END;
$$ language plpgsql;

Create trigger “remove_review” after delete on “Prodotti” for each row
execute procedure elimina_recensioni_orfane();
```

Elimina recensioni orfane utenti

Questo trigger si attiva solo nel momento in cui un utente decide di eliminare il proprio account.

```
Unset
create function elimina_recensioni_orfane_utenti() returns trigger as $$
BEGIN
	delete from public.”Recensioni” where “Recensioni”.”User” IS NULL;
	delete from public.”ProdottiRecensioni” where id_recensione_entry not
		in (select id_recensione_entry from “Recensioni”);
	return NULL;
END;
$$ language plpgsql;

Create trigger delete_review_after_account after delete on “Utenti” for each
row execute procedure elimina_recensioni_orfane_utenti();
```

Ulteriori aspetti tecnologici

Nel corso dello sviluppo dell’applicazione oltre alle librerie di base proposte per creare il server e connettersi al database (Flask, Flask-Login e SqlAlchemy), abbiamo usato la libreria “os” per interfacciarsi al sistema operativo per salvare le immagini e la libreria “hashlib” che ci ha permesso in modo rapido di crittografare le password con l’algoritmo sha256.
Ultimo aspetto è la presenza di codice javascript che ci ha permesso di rendere alcune pagine più interattive inserendo dei messaggi di alert come per esempio :

```
JavaScript
document.getElementById("delete").addEventListener("click", function () {
	var conferma = confirm("Sei sicuro di voler procedere?");
	if (conferma) {
		window.location.href = "/deleteAccount";
	} else {
		alert("Hai annullato l'azione.");
	}
});
```

Contributo al progetto

Nelle primissime fasi del progetto abbiamo diviso il lavoro tra front-end e back-end. Uno ha costruito i primi template HTML, l'altro ha creato il primo embrione di database e il primo codice python per la connessione del server al DB. Il grosso del lavoro l'abbiamo però svolto insieme, a 4 mani, trovandoci in campus per scrivere il codice e sviluppare il DB insieme. La prima fase è stata costruire il modello concettuale, poi abbiamo cominciato a mettere mano al codice vero e proprio.
Definita la struttura del sito, abbiamo creato un file con, punto per punto, tutte le funzionalità che avremmo desiderato implementare, dettagliatamente descritte e suddivise in task più piccoli. Ad ogni incontro selezionavamo i task da implementare, concentrandoci su quelli. Abbiamo notato come avere il progetto suddiviso in punti portasse a uno sviluppo molto più lineare e veloce rispetto ad avere solamente tutto in testa. Ovviamente ci sono stati molti cambiamenti in corso d'opera. Se relativi al punto in fase di implementazione, si procedeva direttamente, altrimenti venivano aggiunti o modificati punti del file "to-do". Poi eventualmente ognuno per conto proprio poteva continuare lo sviluppo, a seconda del tempo disponibile, comunicando al collega i cambiamenti effettuati e condividendo il codice nuovo tramite github.
Per quanto riguarda questa documentazione, è stata scritta in parte in presenza, in parte a distanza su un file "Documenti google" condiviso.