Progetto Basi di Dati 2023/2024

SOCIAL NETWORK CON FUNZIONALITÀ PUBBLICITARIE

Marta Cazzin 883396, Asia Carraro 876588

INDICE

INTRODUZIONE

Il progetto implementato tratta una piattaforma web di social network che consente agli utenti di interagire tra loro rispettando le diverse modalità d'uso. L'applicazione è progettata per abbracciare due ruoli specifici: gli utenti e gli inserzionisti.

Gli utenti possono creare profili, pubblicare e condividere contenuti, seguire amici e interagire con loro tramite chat. Possono inoltre interagire con i post degli altri, lasciando commenti o mettendo "mi piace".

Gli inserzionisti, oltre a poter sfruttare tutte le funzionalità degli utenti, hanno la possibilità di creare annunci mirati per un pubblico specifico in base a criteri come sesso, età e interessi. Gli annunci possono essere personalizzati ulteriormente definendo un periodo di pubblicazione specifico e un budget preciso.

Struttura del progetto

Vista ad albero di tutta la cartella del progetto:

```
|    app.py  // file py con database, route e funzioni necessarie. Con questo file si fa partire il tutto
├───contenuti  // contiene tutte le immagini ed è qui che le foto e i video  vengono salvati
├───database  // database e le insert in sql
├───templates  // tutti i file html necessari
├───venv
└───__pycache__
```

Il documento principale, ovvero *app.py*, è strutturato come segue:

- Import: Inserimento di tutti gli import necessari alla nostra applicazione
- Accesso al server: Creazione dell'app con collegamento al database PostgreSQL
- Creazione del database: Definizione di un modello di database per una piattaforma
- Funzioni utili: Funzioni utili all'interno dell'applicazione come il controllo della password e della mail
- Rotte dell'Applicazione: definizione delle rotte principali per il login, logout, registrazione, gestione degli interessi, home page e molto altro.

Abbiamo scelto di inserire tutte le route, le funzioni e la gestione del database nello stesso file app.py così da non avere problemi con i vari import e far funzionare nel modo più semplice possibile il database e il sito.

Inoltre, abbiamo strutturato ulteriormente il codice attraverso l'uso dei blueprints, che hanno reso il codice più robusto e modulabile: soprattutto in un'ottica di lavoro di gruppo.

FUNZIONALITA' PRINCIPALI

L'applicazione è stata progettata per soddisfare le esigenze di due tipologie principali di utenti: utenti ordinari e inserzionisti. Questa distinzione ci ha permesso di organizzare meglio le rotte e le funzionalità dedicate a ciascun gruppo, garantendo un'esperienza personalizzata per ogni tipo di utente.

Una delle scelte fondamentali che abbiamo implementato è la possibilità, per gli utenti ordinari, di definire i propri interessi al momento della registrazione. Questa funzionalità consente di offrire pubblicità mirate, specifiche per ogni utente, migliorando così la rilevanza dei contenuti promozionali. Gli interessi possono essere modificati in qualsiasi momento, offrendo flessibilità e controllo all'utente.

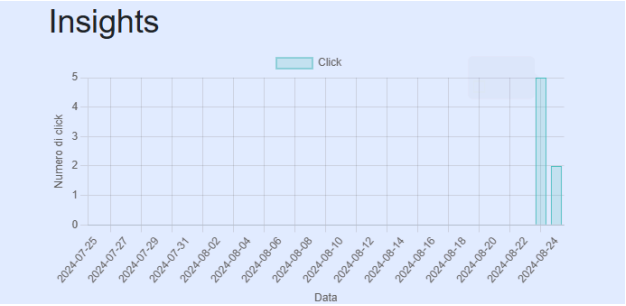
Per quanto riguarda la creazione dei post, abbiamo deciso di suddividere i contenuti in tre tipologie: testo, immagine e video. Questa categorizzazione non solo facilita la visualizzazione dei contenuti, ma semplifica anche la progettazione dell'HTML, rendendo l'interfaccia più intuitiva e ordinata.

Per la gestione delle reti di amicizia, abbiamo optato per rendere ogni account pubblico, consentendo così di accettare le richieste di amicizia in modo immediato. Tuttavia, per proteggere la privacy degli utenti, i contenuti di un profilo sono visibili solo dopo che un utente inizia a seguire un altro. Inoltre, gli utenti hanno la possibilità di rimuovere i follower dalla propria lista, permettendo loro di gestire in modo autonomo la propria cerchia di amici.

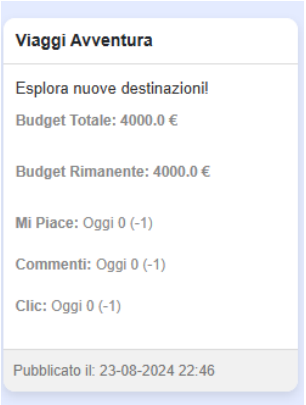
Gli utenti possono cercare altri membri della piattaforma direttamente dalla home page, facilitando così la connessione e l'interazione tra di loro.

Per aumentare ulteriormente l'interazione, abbiamo integrato una sezione chat, dove gli utenti possono scambiarsi messaggi e condividere pubblicazioni.

Ogni volta che si verifica un'interazione (come un nuovo follower, un like o un commento), questa viene visualizzata in una schermata dedicata alle notifiche, fornendo una panoramica completa delle attività relative al proprio account. Come in ogni social network, gli utenti possono mettere o togliere like e commentare i post. Se il post appartiene all'utente, quest'ultimo ha la possibilità di eliminare i commenti indesiderati.



Gli inserzionisti, d'altro canto, dispongono di una sezione specifica dove possono monitorare le loro campagne pubblicitarie attraverso grafici a colonna, che mostrano lo storico delle performance.



Nella home page, possono visualizzare tutte le loro pubblicità e accedere a informazioni dettagliate sulle interazioni ricevute, con la possibilità di confrontare i dati con quelli del giorno precedente.

PROGETTAZIONE CONCETTUALE E LOGICA DEL DATABASE

Il database relazionale che abbiamo utilizzato è PostgreSQL, oggetto di approfondimento nel corso. Per quanto riguarda il lato backend, abbiamo scelto la gestione e connessione al database utilizzando Flask-SQLAlchemy, che offre una vasta documentazione, in combinazione con Psycopg2. Come riportato nella documentazione: "Psycopg2 è l'adattatore per database PostgreSQL più popolare per il linguaggio di programmazione Python. Le sue caratteristiche principali includono una completa implementazione della specifica Python DB API 2.0 e la thread safety, permettendo a più thread di condividere la stessa connessione. È stato progettato per applicazioni multi-thread che creano e distruggono numerosi cursori e gestiscono un alto volume di operazioni 'INSERT' o 'UPDATE' simultanee."

Il database progettato rappresenta un sistema complesso per la gestione di utenti, contenuti, annunci pubblicitari e interazioni sociali. L'obiettivo principale del design è fornire una piattaforma robusta e scalabile che supporti un ampio ventaglio di funzionalità sociali, dalla pubblicazione di post alla gestione degli annunci pubblicitari, inclusa la possibilità di interazioni tra utenti come commenti, "like" e messaggi.

Struttura del Database

Dopo un'attenta sessione di riflessioni e discussioni, abbiamo deciso di strutturare il nostro database seguendo uno schema ben definito, che incorpora anche l'uso di enumerazioni per migliorare la chiarezza e la coerenza nella gestione delle informazioni. Le enumerazioni sono state

applicate in particolare ai campi relativi al sesso e al ruolo degli utenti, garantendo una gestione precisa delle preferenze e dei permessi all'interno della piattaforma.

La prima entità che abbiamo definito è quella degli **Users**, che contiene tutte le informazioni principali degli utenti registrati. Ogni utente avrà un username unico, accompagnato dai dati personali come nome, cognome, email, genere, età, e ruolo. Abbiamo anche previsto un campo opzionale per una biografia, offrendo agli utenti la possibilità di descriversi ulteriormente.

La seconda entità, denominata **Interessi**, raccoglie tutti gli interessi disponibili sulla piattaforma, come ad esempio sport, musica, arte, e altri. Questo permette agli utenti di selezionare i propri interessi personali, favorendo una personalizzazione più mirata dei contenuti, come gli annunci pubblicitari e le raccomandazioni di nuovi amici.

Per gestire la relazione tra utenti e interessi, abbiamo creato una tabella di associazione denominata **UserInteressi**. Questa entità rappresenta un'associazione molti-a-molti tra utenti e interessi, permettendo di registrare e gestire efficientemente i vari interessi degli utenti.

Abbiamo poi definito la tabella **Amicizia**, che tiene traccia delle relazioni sociali tra gli utenti. Questa tabella non solo indica se due utenti sono amici, ma registra anche quando il legame è stato instaurato. È possibile gestire sia legami unidirezionali che bidirezionali, permettendo una rappresentazione più realistica delle reti sociali all'interno della piattaforma.

Per quanto riguarda i contenuti generati dagli utenti, abbiamo definito la tabella **Post**, che gestisce ogni tipo di contenuto pubblicato dagli utenti, inclusi testi, immagini, video, e altro. Ogni post è associato all'utente che lo ha creato e include informazioni come il tipo di contenuto, il testo associato e il timestamp di creazione.

I **PostComments** costituiscono un'entità separata che raccoglie i commenti fatti sui post dagli utenti. Ogni commento è associato a un post specifico e contiene il contenuto del commento stesso e il timestamp di creazione.

Abbiamo inoltre introdotto la tabella **PostLikes** per tenere traccia dei like sui post. Questa entità registra quale utente ha messo like a quale post e il momento in cui l'azione è avvenuta.

Per quanto riguarda la gestione degli annunci pubblicitari, abbiamo definito diverse entità. La tabella **Annunci** si occupa di gestire gli annunci pubblicitari sulla piattaforma. Contiene informazioni sull'inserzionista, il tipo di annuncio, il target di genere e di età, l'interesse target e le date di inizio e fine dell'annuncio. Questo consente una gestione dettagliata e mirata delle campagne pubblicitarie, migliorando l'efficacia degli annunci.

La tabella **AnnunciBudget** è stata creata per gestire il budget assegnato a ciascun annuncio pubblicitario. In questa entità viene registrato il budget totale inizialmente assegnato all'annuncio

e si tiene traccia del budget rimanente man mano che l'annuncio riceve clic, permettendo un controllo accurato dei costi delle campagne.

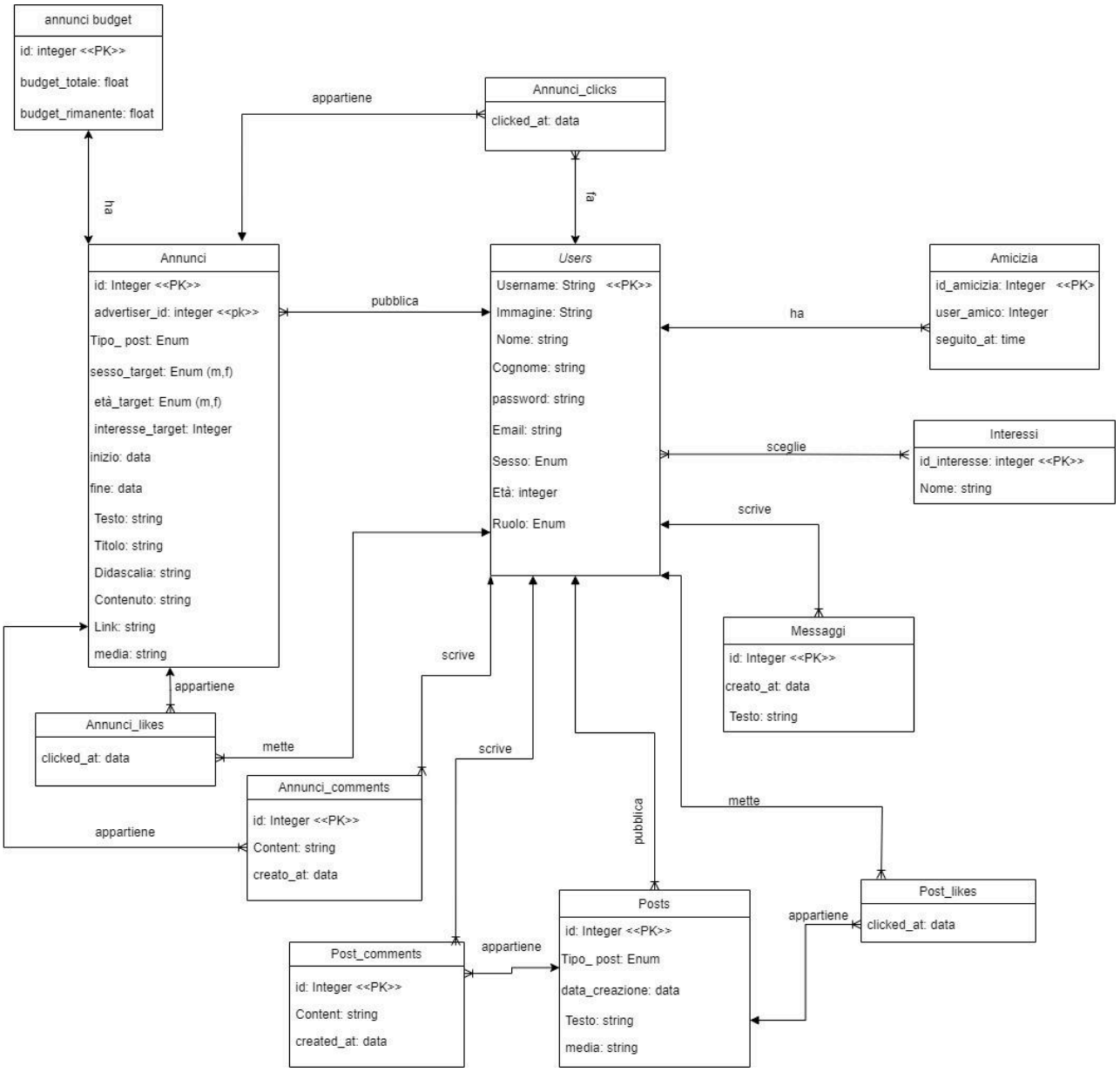
La tabella **AnnunciClicks** è dedicata alla registrazione dei clic sugli annunci. Ogni clic viene associato a un utente specifico e a un annuncio, e include il timestamp del clic, fornendo così preziose informazioni per l'analisi dell'efficacia degli annunci.

Analogamente ai post, abbiamo creato la tabella **AnnunciComments**, che raccoglie i commenti fatti sugli annunci pubblicitari. Ogni commento è legato a un annuncio specifico e a un utente, e include il contenuto del commento e il timestamp di creazione, permettendo di monitorare le interazioni degli utenti con i contenuti pubblicitari.

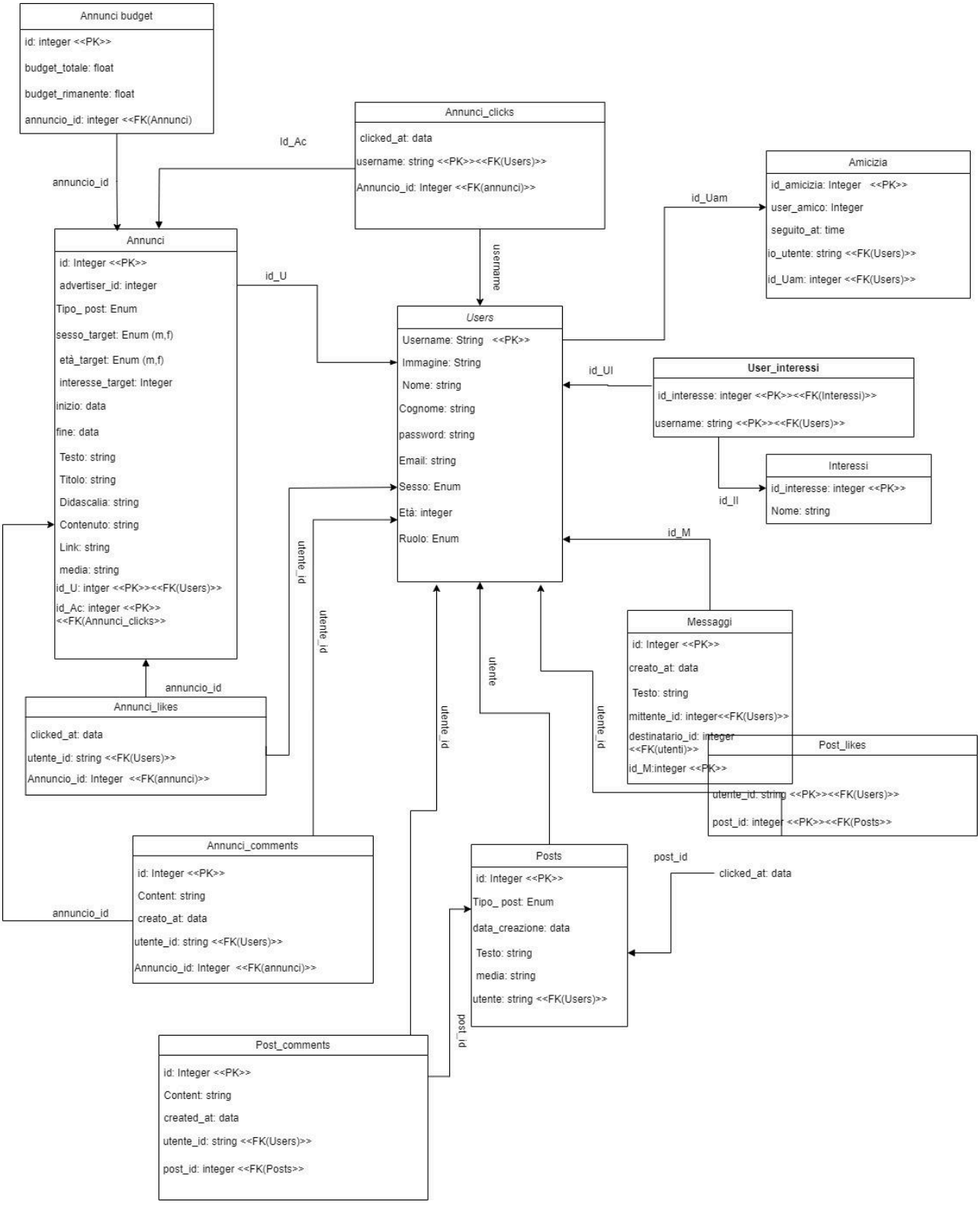
La tabella **AnnunciLikes** traccia i like ricevuti dagli annunci pubblicitari, registrando quale utente ha messo like a quale annuncio e quando, fornendo un ulteriore indicatore dell'engagement con i contenuti promozionali.

Infine, abbiamo definito la tabella **Messaggi**, che gestisce la messaggistica privata tra gli utenti. Ogni messaggio contiene il testo del messaggio stesso, l'ID del mittente e del destinatario, il timestamp di creazione e, se applicabile, un riferimento a un post che può essere stato inviato insieme al messaggio. Questa entità facilita la comunicazione diretta tra gli utenti della piattaforma, aggiungendo un ulteriore livello di interazione sociale.

Schema concettuale e logico



(schema concettuale)



(schema logico)

QUERY PRINCIPALI

Per quanto riguarda le funzionalità implementate nell'ambito dell'applicativo, le query eseguite sono risultate almeno a livello python molto semplici. A rendere il tutto di facile implementazione è stato il supporto che SQLAlchemy utilizza per gestire le relazioni molti a molti e i backref in modo da rendere le relazioni/vincoli di chiave esterna percorribili anche nel verso opposto a quello prestabilito. Tra queste riteniamo rilevante segnalarne alcune.

Query 1: Recupero Annunci Basati sugli Interessi e il Budget

Semantica: Questa sottoquery della route per la homepage dell'utente recupera gli annunci pubblicitari pertinenti per un utente specifico in base a criteri come interessi, sesso e età. Gli annunci sono filtrati in modo che corrispondano agli interessi dell'utente, e solo quelli non scaduti e con un budget > 0 sono mostrati. Viene restituito un massimo di 3 annunci, ordinati per data di scadenza.

```
SELECT
      ab.annuncio_id
  FROM
      AnnunciBudget ab
  JOIN
      Annunci a ON a.id = ab.annuncio_id
  WHERE
      a.interesse_target IN (/* lista di interessi */) AND
      (a.sesso_target = 'user_sesso' OR a.sesso_target = 'tutti') AND
      a.eta_target <= user_eta AND
      a.fine > NOW()
  ORDER BY
      ab.budget DESC,
      a.inizio DESC
  LIMIT 3
```

Query 2 Recupero dei "Like" dei Post

Semantica: Questa query recupera i dettagli dei "like" sui post pubblicati da un specifico utente. Viene fatta un'unione tra le tabelle PostLikes, Post e Users per ottenere informazioni sui like, sui post e sugli utenti che hanno messo il like, escludendo l'utente che sta visualizzando i post.

```
SELECT PostLikes., Post., Users.*
FROM PostLikes
     JOIN Post ON PostLikes.post_id = Post.id
     JOIN Users ON PostLikes.utente_id = Users.id_utente
WHERE Post.utente = :user_id AND PostLikes.utente_id != :user_id;
```

Queste query sono similari nell'utilizzo anche per gli annunci.

Query 3 Recupero dei Commenti sui Post

Semantica: Similare alla query precedente, questa istruzione estrae i commenti sui post pubblicati da un specifico utente. Viene effettuata un'unione tra le tabelle PostComments, Post e Users per ottenere informazioni sui commenti e sugli autori, escludendo l'utente corrente.

```
SELECT PostComments., Post., Users.*
FROM PostComments
```

```
    JOIN Post ON PostComments.post_id = Post.id
    JOIN Users ON PostComments.utente_id = Users.id_utente
WHERE Post.utente = :user_id  AND PostComments.utente_id != :user_id;
```

Query 4 Recupero dei Dettagli di un Post Specifico

Semantica: Questa query ottiene i dettagli di un post specifico, inclusi il titolo, il contenuto, l'autore e, opzionalmente, i commenti associati al post con i relativi autori. Usa joins per collegare le informazioni tra le tabelle Post, Users, e PostComments.

```
SELECT  p.id AS post_id,
    p.titolo AS post_titolo,
    p.contenuto AS post_contenuto,
    u.username AS post_autore,
    c.id AS comment_id,
    c.contenuto AS comment_contenuto,
    cu.username AS comment_autore
FROM Post p JOIN Users u ON p.utente = u.id_utente
     LEFT JOIN PostComments c ON p.id = c.post_id
     LEFT JOIN Users cu ON c.utente_id = cu.id_utente
WHERE  p.id = :post_id;
```

Query 5 Recupero Profilo di un Amico e i Suoi Post

Semantica: Recupera le informazioni del profilo di un amico e i suoi post se esiste una relazione di amicizia.

-- **Query**: se esiste una relazione di amicizia tra l'utente attuale e un amico specificato.

```
SELECT 1
FROM  Amicizia
WHERE io_utente = :current_user_id  AND user_amico = :amico_id;
```

-- **Query**: Se esiste una relazione di amicizia, questa query recupera i post pubblicati dall'utente amico. I dettagli restituiti includono l'id del post, il titolo, il contenuto e la data di creazione.

```
SELECT p.id AS post_id,
    p.titolo AS post_titolo,
    p.contenuto AS post_contenuto,
    p.creato_at AS data_post
FROM Post p
WHERE p.utente = amico_id;
```

PRINCIPALI SCELTE PROGETTUALI

Nel progetto, sono state attuate diverse scelte progettuali per garantire l'integrità, la sicurezza e le performance del sistema. Di seguito, vengono illustrate le principali strategie adottate.

Politiche di integrità

Le politiche di integrità sono cruciali per garantire che i dati all'interno del sistema siano corretti, coerenti e affidabili. Per raggiungere questo obiettivo, abbiamo implementato diverse misure:

1. **Integrità referenziale:** Abbiamo utilizzato chiavi esterne per collegare le varie tabelle del database, assicurando che ogni dato sia collegato a un'entità valida. Ad esempio, ogni post, commento e like è associato a un utente e a un post specifico, prevenendo la creazione di riferimenti orfani e mantenendo la coerenza dei dati.
2. **Trigger e vincoli a livello di database:** Questi meccanismi garantiscono che le operazioni rispettino le regole di integrità definite. Ad esempio, non è possibile aggiungere un commento se il post corrispondente non esiste, né registrare un like se l'utente ha già espresso un mi piace per lo stesso post. Inoltre, il sistema verifica la coerenza temporale degli annunci, assicurando che la data di fine sia sempre successiva a quella di creazione. Ecco alcuni esempi di trigger implementati:

Cancellazione a cascata di commenti e like associati a un post:

```
CREATE OR REPLACE FUNCTION cascade_delete_post()
RETURNS TRIGGER AS $$
BEGIN
   DELETE FROM post_comments WHERE post_id = OLD.id;
   DELETE FROM post_likes WHERE post_id = OLD.id;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER delete_post_cascade
BEFORE DELETE ON posts
FOR EACH ROW EXECUTE FUNCTION cascade_delete_post();
```

Prevenzione di amicizie duplicate:

```
CREATE OR REPLACE FUNCTION prevent_duplicate_amicizia()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM amici WHERE (io_utente = NEW.io_utente AND
user_amico = NEW.user_amico) OR (io_utente = NEW.user_amico AND
user_amico = NEW.io_utente)) THEN
        RAISE EXCEPTION 'Questa amicizia già esiste';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_duplicate_amicizia
BEFORE INSERT ON amici
FOR EACH ROW
EXECUTE FUNCTION prevent_duplicate_amicizia();
```

3. **Gestione delle eccezioni:** In caso di violazione delle politiche di integrità, come il tentativo di inserire chiavi duplicate, il sistema cattura e gestisce le eccezioni, riportando il database a uno stato coerente. Questo garantisce la stabilità e l'affidabilità del sistema.

Definizione di ruoli e politiche di autorizzazione

Per garantire che solo gli utenti autorizzati possano accedere a determinate funzionalità, abbiamo implementato un sistema di autenticazione e autorizzazione.

1. **Autenticazione**: L'accesso alle aree protette dell'applicazione è riservato agli utenti autenticati, grazie all'uso del decoratore @login_required. Gli utenti non autenticati vengono reindirizzati automaticamente alla pagina di login, proteggendo i contenuti riservati.
2. **Gestione dei ruoli:** Gli utenti sono classificati in base ai loro ruoli, come utenti normali e pubblicitari. Questa suddivisione permette di personalizzare le funzionalità disponibili per ciascun tipo di utente, come la possibilità di pubblicare annunci riservata ai soli pubblicitari.
3. **Controlli di accesso:** Sono stati implementati controlli rigorosi per assicurare che gli utenti possano eseguire solo le azioni per cui sono autorizzati. Ad esempio, solo l'autore di un post o di un commento ha il diritto di eliminarlo, prevenendo abusi e mantenendo un ambiente sicuro.

Uso di indici

L'uso di indici è stato fondamentale per ottimizzare le performance delle query all'interno del database.

1. **Indici su chiavi esterne**: Sono stati creati indici sulle colonne contenenti chiavi esterne, come utente_id nelle tabelle dei post, commenti e like. Questi indici migliorano notevolmente le performance delle query, facilitando le operazioni di join e riducendo i tempi di risposta.
2. **Indici per ricerche**: Per velocizzare le ricerche relative agli utenti, è stato implementato un indice sulla colonna username. Questo indice accelera l'accesso e la ricerca degli utenti in base al loro nome, rendendo più efficienti le query che filtrano o cercano specifici username.

SCELTE SPECIFICHE

Quando si tratta di gestire la pubblicazione dei post e la gestione dei file multimediali in un'applicazione, ci sono diverse pratiche chiave per garantire un'esperienza utente fluida e sicura, queste sono quelle che abbiamo seguito noi.

Per quanto riguarda i file multimediali, come immagini e video, questi vengono salvati in una directory specifica chiamata "contenuti". Questo approccio organizzato aiuta a tenere tutto in ordine e a semplificare l'accesso ai file. Per evitare conflitti e garantire un accesso sicuro, adottiamo un percorso di salvataggio standardizzato. Questo significa che i file vengono salvati in modo sistematico, riducendo il rischio di sovrapposizioni e mantenendo i dati protetti.

Passando alle notifiche, abbiamo un sistema che raccoglie informazioni da diverse fonti, come i like, i commenti e le richieste di amicizia. Tutte queste notifiche vengono poi aggregate in una lista unica, ordinata cronologicamente. Questo permette agli utenti di vedere tutte le loro notifiche in sequenza, facilitando la consultazione e la gestione delle interazioni recenti.

Per rendere il tempo delle notifiche più comprensibile, utilizziamo una funzione chiamata 'time_since'. Questa funzione converte i timestamp in un formato facilmente leggibile, adattato al fuso orario italiano. In questo modo, gli utenti possono vedere quanto tempo è passato dall'ultimo evento.

Per quanto riguarda la chat tra amici, abbiamo organizzato i messaggi in base alle conversazioni. Gli utenti possono vedere chiaramente le chat con i loro amici, mantenendo tutto ben ordinato. Inoltre, è possibile inviare messaggi di testo e condividere post direttamente nella chat, offrendo così una maggiore flessibilità e migliorando l'esperienza comunicativa complessiva.

Quando si parla di gestione degli annunci all'interno dell'applicazione, ci sono due aspetti principali che vengono curati con attenzione: la targetizzazione e la durata degli annunci.

La targetizzazione degli annunci è un processo che consente agli inserzionisti di indirizzare i loro messaggi a gruppi specifici di utenti. Gli annunci possono essere personalizzati in base a vari criteri come sesso, età e interessi degli utenti. Questo significa che un inserzionista può scegliere di mostrare il suo annuncio solo a un pubblico che corrisponde a determinate caratteristiche, aumentando così l'efficacia della campagna pubblicitaria e migliorando le probabilità di raggiungere persone realmente interessate.

Mentre per la gestione della durata degli annunci, ogni annuncio ha una durata specificata dall'inserzionista. Una volta che questo periodo scade, l'annuncio non viene più visualizzato dagli utenti. Questo sistema non solo garantisce che gli utenti vedano solo contenuti pertinenti e aggiornati, ma consente anche agli inserzionisti di aggiornare regolarmente le loro campagne pubblicitarie con nuovi annunci. In questo modo, gli spazi pubblicitari rimangono dinamici e rilevanti, offrendo opportunità fresche e tempestive per il pubblico e per gli inserzionisti.

ULTERIORI INFORMAZIONI

Scelte usate per programmare in gruppo

Tutti i membri del team hanno utilizzato l'IDE Visual Studio Code (VSCode) per lo sviluppo, sfruttando plugin specifici che hanno reso la collaborazione più efficiente e produttiva.

Data la natura collaborativa del progetto, è stato essenziale utilizzare un Version Control System (VCS) per gestire i file e le versioni del codice. Abbiamo scelto Git come VCS e GitHub per l'hosting, permettendo a ogni membro del team di lavorare sulla versione più aggiornata e di apportare modifiche in modo ordinato e tracciabile.

Le comunicazioni e i chiarimenti sono stati gestiti tramite Discord, facilitando la collaborazione in tempo reale.

Librerie usate

Per l'implementazione di funzionalità complesse e strutturate nel backend, ci siamo affidati a librerie di terze parti, molte delle quali sono ben note e mantenute su GitHub. Queste librerie, che si integrano come estensioni di Flask, fungono da wrapper progettati per potenziare e arricchire le capacità del framework.

Alcune di esse sono:

flask-sqlalchemy: è un layer di astrazione che si frappone tra Flask e SQLAlchemy ORM, facilitando la gestione delle sessioni e le connessioni al database. Introduce una sintassi semplificata per le query, consentendo di gestirle attraverso chiamate di metodo e l'uso di classi. Offrendo un'alternativa all'utilizzo di istruzioni SQL passate come stringhe, o prepered statement che è comunque ancora possibile utilizzare.

flask-login: gestisce le sessioni utente, con particolare attenzione all'autenticazione, nelle operazioni di login, registrazione e logout.

flask-session: libreria utilizzata per le routes e la gestione delle sessioni lato server.

flask_SocketIO: è un estensione di Flask per supportare WebSocket, consentendo una comunicazione bidirezionale in tempo reale tra il server e il client. Usato per aggiungere funzionalità di comunicazione in tempo reale alla applicazione, come chat, notifiche in tempo reale, o aggiornamenti live dei dati.

sqlalchemy: è un'estensione di Python per la gestione dei database che funge da ORM (Object-Relational Mapping), consentendo di interagire con i database in modo più intuitivo e orientato agli oggetti. SQLAlchemy permette di definire modelli Python che rappresentano le tabelle del database, gestire relazioni complesse e costruire query SQL in modo programmatico.

Contributo al progetto:

Durante lo sviluppo del progetto, abbiamo monitorato la frequenza dei commit su GitHub, che mostra chiaramente come il gruppo si sia concentrato maggiormente nelle ultime settimane di Luglio, per completare il lavoro entro l'inizio di Agosto. Il gruppo ha suddiviso i compiti in modo efficace per rispettare la scadenza prefissata.

Andiamo ora a vedere come è stata attuata la suddivisione dei lavori.

Per quanto riguarda il **backend**, tutte le componenti del gruppo hanno collaborato attivamente. Quando qualcuno incontrava difficoltà nel risolvere un bug, un altro componente interveniva per offrire supporto. Il **frontend** è stato prevalentemente gestito da Marta, anche se Asia ha contribuito alla realizzazione di alcune pagine. La **fase di ideazione** è stata affrontata in modo equo, il gruppo ha lavorato insieme per raggiungere una visione comune. Questo processo di brainstorming e decisione è stato portato avanti tramite chiamate, assicurando che il gruppo fosse allineato sull'obiettivo finale.

Infine, la **documentazione** del progetto è stata avviata da Asia e successivamente rivista e completata da Marta, garantendo una conclusione accurata e condivisa del lavoro.