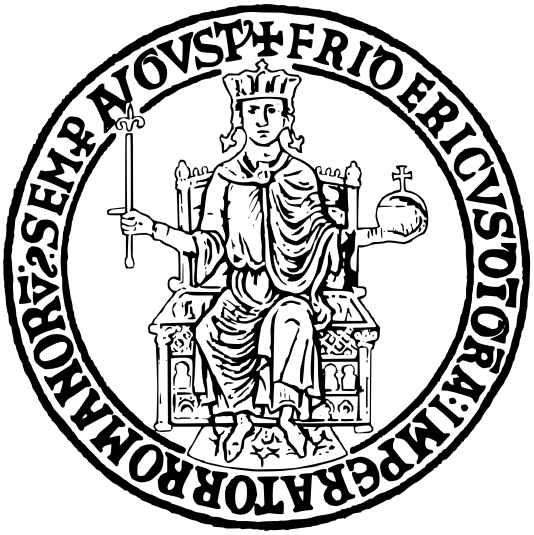


Università degli Studi di Napoli Federico II

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Triennale in Informatica

IMAGESPOT

Sistema di gestione di una galleria fotografica geolocalizzata

Professori:

Silvio Barra

Porfirio Tramontana

Autori:

Daniyl Popov N86003893

Gabriele Maione N86004117

ANNO ACCADEMICO 2022/2023

Indice

1 Introduzione **3**
1.1 Analisi del problema . . . 3

2 Progettazione Concettuale **5**
2.1 Introduzione . . . 5
2.2 Class Diagram UML . . . 5
2.3 Analisi delle ridondanze . . . 5
2.4 Rimozione delle Aggregazioni e delle Composizioni . . . 6
2.5 Rimozione degli attributi multipli . . . 6
2.6 Rimozione degli attributi strutturati . . . 6
2.7 Accorpamento/Partizionamento di Entità/Associazioni . . . 6
2.8 Identificazione chiavi primarie . . . 6
2.9 Class Diagram UML Ristrutturato . . . 7
2.10 Class Diagram E/R Ristrutturato . . . 7
2.11 Dizionario delle classi . . . 8
2.11.1 Descrizione delle classi . . . 8
2.11.2 Dizionario degli attributi . . . 8
2.12 Dizionario delle associazioni . . . 10
2.13 Dizionario dei vincoli . . . 11

3 Progettazione Logica **12**
3.1 Schema Logico . . . 12

4 Progettazione Fisica **13**
4.1 Creazione dei domini . . . 13
4.2 Schema delle tabelle . . . 13
4.2.1 Account . . . 13
4.2.2 Device . . . 14
4.2.3 Post . . . 14
4.2.4 Location . . . 14
4.2.5 Bookmark . . . 15
4.2.6 Tagged_User . . . 15
4.2.7 Following . . . 15
4.2.8 Collection . . . 15
4.2.9 Collection_Post . . . 16
4.2.10 Subject . . . 16
4.2.11 Subject_Post . . . 16
4.2.12 User_Device . . . 16
4.3 Defizione delle View . . . 17
4.3.1 Collection_Stats . . . 17
4.3.2 User_Stats . . . 17
4.4 Definizione dei Trigger . . . 18

4.4.1 Rimozione post privati 18
4.4.2 Rimozione dei post presenti nelle collezioni 18
4.4.3 Rimozione di post cancellati dalle collezioni 19
4.4.4 Rimozione soggetti "orfani" 20
4.4.5 Hashing delle password 20
4.5 Definizione delle funzioni 21
4.5.1 Inserimento del soggetto 21
4.5.2 Inserimento nuovo account 22

1 Introduzione

Questo progetto mira a sviluppare un sistema informativo che permetta agli utenti di condividere fotografie e informazioni correlate, come ad esempio il dispositivo di scatto, la posizione geografica e una possibile descrizione associata. Il sistema sarà in grado di gestire gli account utenti, consentendo loro di interagire con altre persone e le relative fotografie, oltre a fornire funzionalità come ad esempio la creazione di collezioni personalizzate.

1.1 Analisi del problema

Per l'analisi dei requisiti, intendiamo identificare le entità e le associazioni che saranno presenti nella nostra progettazione del database. Di seguito elenchiamo le principali entità e le loro associazioni:

- **Entità "Account":** Questa entità descrive il profilo dell'utente e includerà informazioni personali come nome, cognome, e-mail e credenziali di accesso (username e password).
- **Entità "Post" (Fotografia):** Questa entità rappresenta una fotografia pubblicata nel sistema. Ogni post avrà uno stato che può essere "pubblico" o "privato". Sarà associato all'utente che ha creato il post.
- **Entità "Location":** Questa entità descrive la posizione geografica in cui è stata scattata la fotografia. Il luogo può essere identificato tramite coordinate geografiche (latitudine e longitudine) e/o con un indirizzo mnemonico.
- **Entità "Device":** Questa entità descrive il dispositivo con cui la fotografia è stata scattata ed è associata al post corrispondente.
- **Entità "Subject":** Questa entità rappresenta i soggetti presenti nella fotografia. I soggetti devono essere categorizzati in modo da permettere una classificazione delle fotografie in base ai soggetti rappresentati.
- **Entità "Collection":** Questa entità raccoglie i post degli utenti, creando collezioni personalizzate. Una collezione può includere post di diversi utenti ed un post può stare in più collezioni.
- **Entità "Bookmark":** Questa entità permette agli utenti di salvare i post di interesse, mediante la creazione segnalibri per le fotografie preferite.
- **Associazione "Post-Subject":** Questa associazione collega i soggetti presenti nella fotografia al post corrispondente, consentendo di identificare i soggetti rappresentati in ogni fotografia.
- **Associazione "Account-Account":** Questa associazione rappresenta la possibilità per gli utenti di seguire altri utenti e viceversa. In questo modo, gli utenti potranno vedere i post di coloro che seguono nel loro feed.

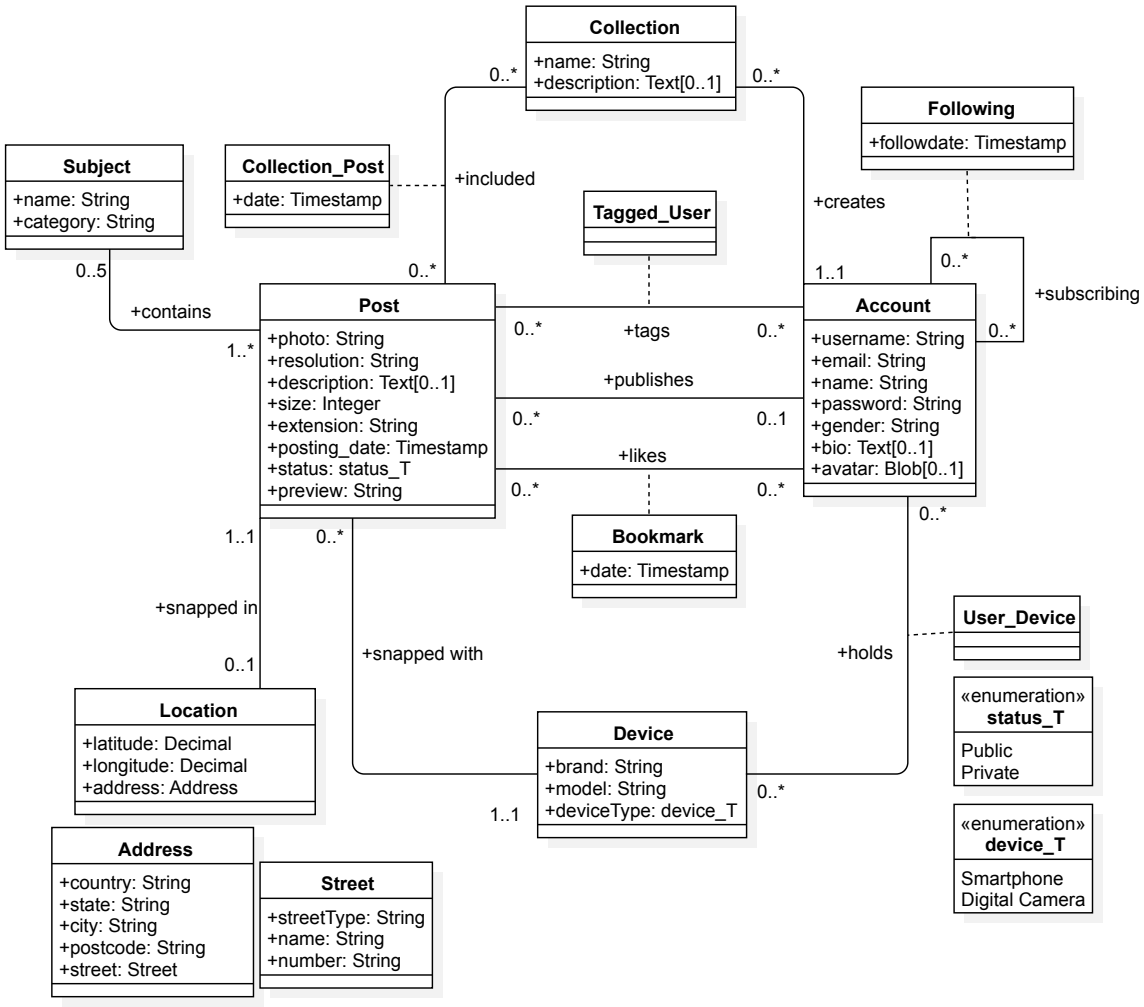
- **Associazione "Account-Post":** Questa associazione collega gli account utente ai post che hanno creato. Ogni post sarà associato all'account utente corrispondente, a meno che l'utente non decida di eliminare il post.
- **Associazione "Account-Post(Bookmark)":** Questa associazione collega gli account utente ai post che sono stati "salvati" come segnalibri. Un utente può avere molti post nei suoi segnalibri, e ogni post può essere presente nei segnalibri di più utenti.
- **Associazione "Post-Location":** Questa associazione collega i post ai luoghi in cui sono stati scattati. Un post può essere associato a un unico luogo.
- **Associazione "Account(taggato)-Post":** Questa associazione collega gli utenti taggati al post corrispondente. Un post può menzionare diversi utenti, e ogni utente può essere menzionato in molti post.
- **Associazione "User-Device":** Questa associazione collega gli account utente ai dispositivi che l'utente ha definito. Un utente può avere molti dispositivi.

2 Progettazione Concettuale

2.1 Introduzione

In questa fase, trasformeremo i requisiti analizzati precedentemente in un modello concettuale rappresentato tramite il Class Diagram.

2.2 Class Diagram UML



2.3 Analisi delle ridondanze

Nell'entità Post sono presenti due link per ciascuna immagine uno all'immagine originale e uno alla sua versione scalata o preview. Questa ridondanza è stata deliberatamente incorporata nel sistema per affrontare la gestione delle immagini pesanti e la velocità di caricamento.
Caricare solo le anteprime anziché l'immagine completa riduce significativamente il carico sul nostro server. Inoltre, fornire anteprime consente agli utenti di ottenere un'idea più rapida e chiara del contenuto delle immagini, permettendo loro di deci-

dere se desiderano aprire la versione in dimensioni originali solo per le immagini che suscitano il loro interesse.

2.4 Rimozione delle Aggregazioni e delle Composizioni

Nello schema non sono state individuate aggregazioni/composizioni.

2.5 Rimozione degli attributi multipli

Nello schema non sono presenti attributi multipli.

2.6 Rimozione degli attributi strutturati

L'entità "Location" presentava originariamente attributi strutturati "Address" e "Street". Tuttavia, per semplificare il modello dei dati e ottimizzare le operazioni di gestione, si è proceduto con la rimozione degli attributi strutturati e l'integrazione dei loro campi direttamente nell'entità "Location".
L'eliminazione degli attributi strutturati semplifica la struttura complessiva dell'entità "Location". I campi ora sono accessibili senza la necessità di attraversare livelli di attributi annidati.
Inoltre, l'attributo strutturato "Street" è stato integrato in un unico attributo chiamato "Road". Questa scelta mira a ridurre la probabilità di campi vuoti.

2.7 Accorpamento/Partizionamento di Entità/Associazioni

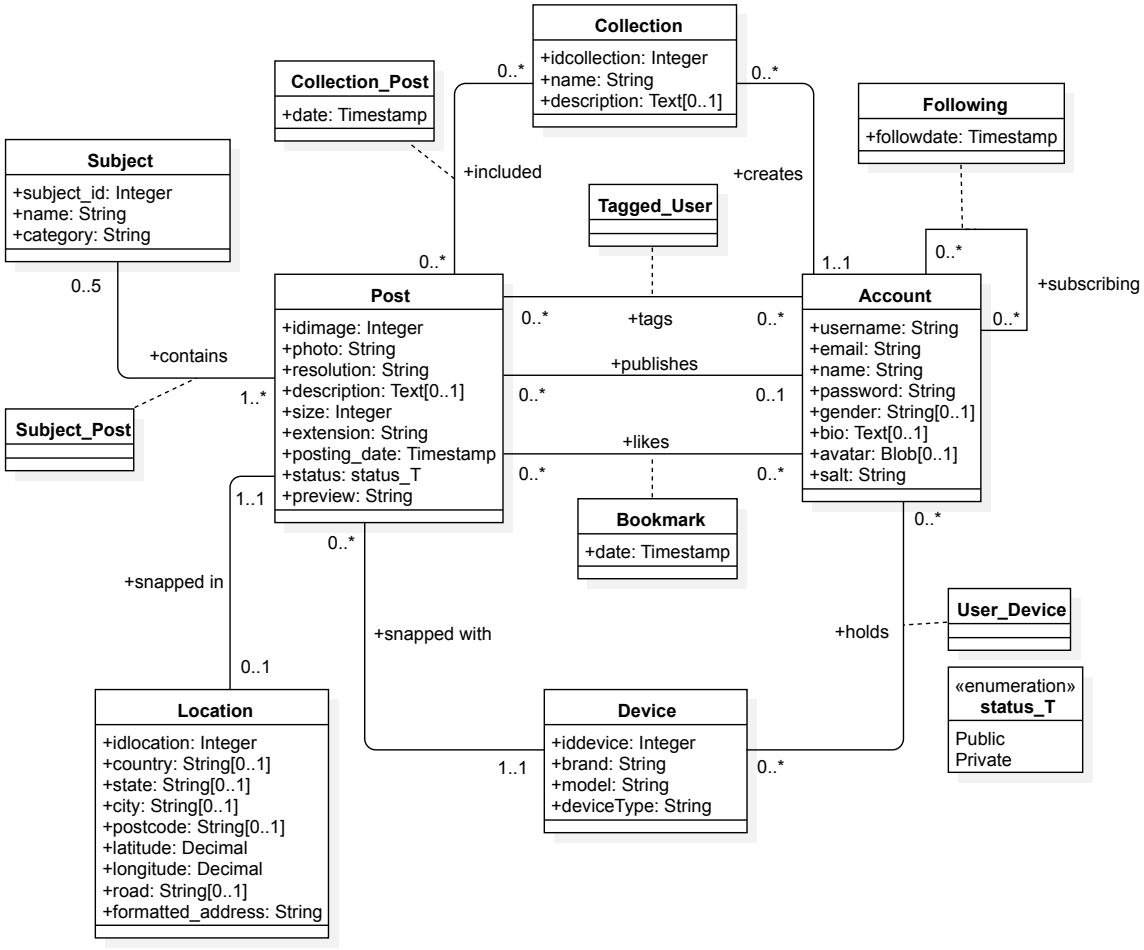
Non sono state identificate necessità di effettuare accorpamenti o partizionamenti delle entità o delle associazioni presenti nel modello.

2.8 Identificazione chiavi primarie

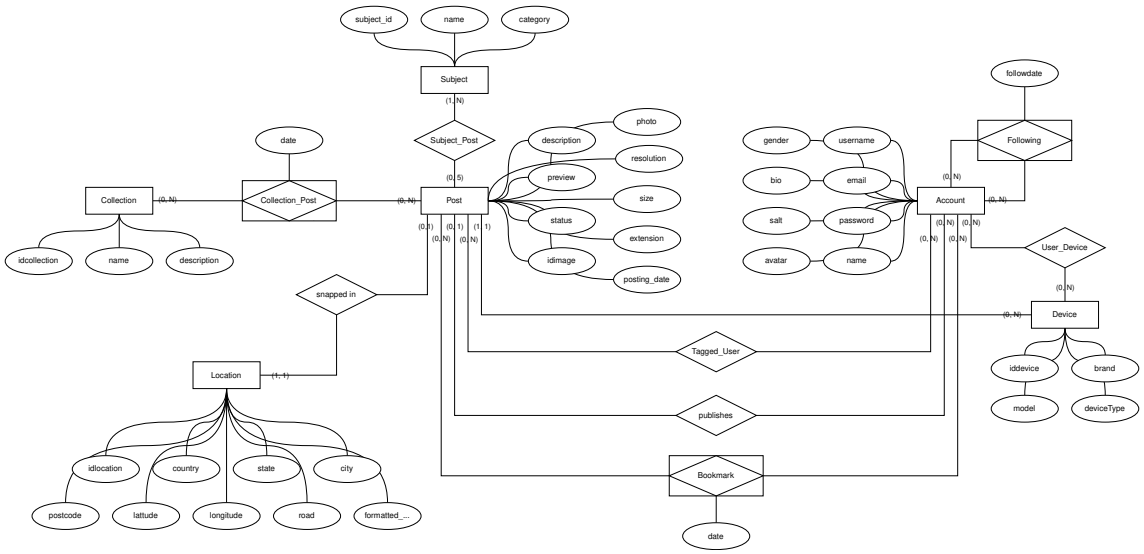
Sono state apportate modifiche per garantire l'identificazione univoca e la gestione delle relazioni tra le entità. Le chiavi primarie sono state introdotte per fornire un modo affidabile per identificare ciascuna istanza delle tabelle coinvolte. Ecco un elenco delle modifiche apportate:

- **idcollection**: Aggiunto come identificatore primario alla tabella "collection".
- **subject_id**: Introdotta come chiave primaria nella tabella "subject".
- **idlocation**: Introdotto come identificatore primario nella tabella "location".
- **iddevice**: Aggiunto come chiave primaria alla tabella "device".
- **idimage**: Introdotta come identificatore primario nella tabella "post".
- **username**: Utilizzato come chiave primaria nella tabella "account". Questa scelta è basata sulla presenza di un campo univoco e non nullo che rappresenta un identificatore affidabile per gli account.

2.9 Class Diagram UML Ristrutturato



2.10 Class Diagram E/R Ristrutturato



2.11 Dizionario delle classi

2.11.1 Descrizione delle classi

Account: Descrive il profilo dell'utente, incluse le informazioni personali e le credenziali.
Post: Descrive la foto e le informazioni ad essa associate.
Location: Descrive un luogo mediante un nome mnemonico e le coordinate.
Device: Descrive i dispositivi che l'utente ha definito e che potrebbe utilizzare per scattare foto.
Subject: Descrive il soggetto presente nella foto.
Subject_Post: Descrive e categorizza i post in cui è presente l'utente.
Collection: Descrive una collezione di fotografie.
Collection_Post: Descrive i post presenti in una collezione e viceversa.
Bookmark: Descrive post "salvati" dell'utente.
Following: Descrive gli account che l'utente sta seguendo.
Tagged_User: Descrive gli utenti mensionati nella foto.
User_Device: Descrive i dispositivi che l'utente ha definito.
Subject_Post: Descrive i soggetti presenti nella foto.

2.11.2 Dizionario degli attributi

Classe	Attributi
Account	**username** (*String*): Identificatore univoco dell'utente **email** (*String*): Email associata all'utente **name** (*String*): Nome associato all'utente **password** (*String*): Password dell'utente **salt** (*String*): Stringa casuale utilizzata per crittografare la password **gender** (*String*): Genere dell'utente **bio** (*String*): Biografia dell'utente dove racconta se stesso **avatar** (*Blob*): Foto profilo dell'utente

Classe	Attributi
Post	**idimage** (*Integer*): Identificatore univoco del post **photo** (*String*): URL della foto in risoluzione originale **resolution** (*String*): Risoluzione della foto **description** (*Text*): Descrizione associata alla fotografia **size** (*Integer*): Dimensione della foto espressa in KB **extension** (*String*): Estensione della foto **posting_date** (*Timestamp*): Data è ora della pubblicazione della foto **status** (*status_T*): Determina se la foto è pubblica o privata (accessibile o meno) **preview** (*String*): URL alla versione scalata della foto originale usata come preview del post
Location	**idlocation** (*Integer*): Identificatore univoco della posizione geografica **country** (*String*): Nome del paese **state** (*String*): Nome dello stato/regione **city** (*String*): Nome della città/comune **postcode** (*String*): Codice di Avviamento Postale **latitude** (*Decimal*): Latitudine del luogo **longitude** (*Decimal*): Longitudine del luogo **formatted_address** (*String*): Nome formattato del luogo **road** (*String*): Nome della via
Device	**iddevice** (*Integer*): Identificatore univoco del dispositivo **brand** (*String*): Nome della marca del dispositivo **model** (*String*): Nome del modello del dispositivo **deviceType** (*String*): Tipo di dispositivo (smartphone/camera digitale)
Subject	**subject_id** (*Integer*): Identificatore univoco del soggetto **category** (*String*): Categoria associata al soggetto **subject** (*String*): Nome associato al soggetto
Collection	**idcollection** (*Integer*): Identificatore univoco della collezione **name** (*String*): Nome della collezione **description** (*Text*): Descrizione della collezione
Collection_Post	**date** (*Timestamp*): Data e ora dell'aggiunta del post alla collezione
Bookmark	**date** (*Timestamp*): Data e ora dell'aggiunta del "segnalibro" al post
Following	**followdate** (*Timestamp*): Data e ora del "follow"

2.12 Dizionario delle associazioni

Associazione	Descrizione	Classi coinvolte
Included	Esprime i post inclusi nelle collezioni.	**Collection**[0...*]: Un post può appartenere a 0 o N collezioni. **Post**[0...*]: In una collezione ci sono 0 o N post.
Creates	Esprime le collezioni create dall'utente.	**Account**[1...1]: Una collezione è creata da uno ed un solo account. **Collection**[0...*]: Un account può creare da 0 a N collezioni.
Tags	Esprime gli account taggati nei post.	**Post**[0...*]: Un account è taggato in 0 o N post. **Account**[0...*]: In un post possono essere taggati da 0 a N account.
Publishes	Esprime i post pubblicati da un account.	**Post**[0...*]: Un account pubblica da 0 a N post. **Account**[0...1]: Un post è pubblicato da 1 o nessun account.
Likes	Esprime i post aggiunti ai segnalibri dagli account.	**Post**[0...*]: Un account può mettere like a 0 o N post. **Account**[0...*]: Un post può avere da 0 a N like.
Contains	Esprime i soggetti associati ai post.	**Post**[1..*]: Un soggetto può essere associato a 1 o N post. **Subject**[0...5]: Un post può avere da 0 a 5 soggetti.
Subscribing	Esprime gli utenti seguiti da altri utenti.	**Account**[0...*]: Un utente può seguire da 0 a N utenti. **Account**[0...*]: Un utente può essere seguito da 0 a N utenti.
Snapped in	Esprime l'associazione di un luogo a un post.	**Post**[1...1]: Un luogo è associato ad uno ed uno solo post. **Location**[0...1]: Un post può avere 1 nessun luogo associato.
Snapped with	Esprime l'associazione di un dispositivo ai post.	**Post**[0...*]: Un dispositivo può essere associato a 0 o N post con cui è stata scattata la foto. **Device**[1...1]: Un post può essere associato a uno ed uno solo dispositivo con il quale è stata scattata la foto.

Holds	Esprime il possesso di dispositivi da parte degli account.	**Account**[0...*]: Un dispositivo può appartenere a 0 o N account. **Device**[0...*]: Un account può avere da 0 a N dispositivi.

2.13 Dizionario dei vincoli

Classe	Descrizione vincoli
Account	**username**: Chiave primaria. **email**: Non nullo, unico e deve rispettare il format: `_%@_%._%`
Post	**idImage**: Chiave primaria. **resolution**: Non nullo e deve rispettare il format: NxN. **status**: Non nullo e i valori consentiti sono: Public/Private. **device**: Riferimento esterno alla tabella Device, non nullo. **profile**: Riferimento esterno alla tabella Account
Device	**iddevice**: Chiave primaria.
Bookmark	**username**, **idimage**: Chiavi composite, entrambi riferimenti esterni alla tabella Account e Post rispettivamente, su cancellazione in cascata.
Tagged_user	**nickname**, **idimage**: Chiavi composite, entrambi riferimenti esterni alla tabella Account e Post rispettivamente, su cancellazione in cascata.
Following	**nickname**, **idfollowing**: Chiavi composite, entrambi riferimenti esterni alla tabella Account e Account.
User_device	**device**, **profile**: Chiavi composite, rispettivamente riferimenti esterni alle tabelle Device e Account.
Collection	**idcollection**: Chiave primaria. **owner**: Riferimento esterno alla tabella Account, non nullo.
Collection_post	**collection**, **post**: Chiavi composite, rispettivamente riferimenti esterni alle tabelle Collection e Post, su cancellazione in cascata.
Subject	**subject_id**: Chiave primaria.
Subject_post	**subject**, **post**: Chiavi composite, rispettivamente riferimenti esterni alle tabelle Subject e Post, su aggiornamento e cancellazione in cascata.
Location	**idlocation**: Chiave primaria. **post**: Riferimento esterno alla tabella Post, su aggiornamento e cancellazione in cascata.

3 Progettazione Logica

3.1 Schema Logico

Nel seguente schema logico le chiavi primarie sono evidenziate in **grassetto** e le chiavi esterne sono indicate con una sottolineatura singola.

Account: (**username**, email, name, password, salt, gender, bio, avatar)
Device: (**iddevice**, brand, model, devicetype)
Post: (**idimage**, photo, resolution, description, size, extension, posting_date, status, preview, device, profile)
device -> Device.iddevice, profile -> account.username
Location: (**idlocation**, country, state, city, poscode, latitude, longitude, formatted_address, road, post)
post -> Post.idimage
Subject: (**subject_id**, category, name)
Subject_Post: (**subject**, **post**)
subject -> Subject.subject_id, post -> Post.idimage
Collection: (**idcollection**, name, description, owner)
owner -> Account.username
Collection_Post: (**collection**, **post**, date)
collection -> Collection.idcollection, post -> Post.idimage
Bookmark: (**username**, **idimage**, date)
username -> Account.username, idimage -> Post.idimage
Following: (**nickname**, **idfollowing**, followingdate)
nickname -> Account.username, idfollowing -> Account.username
TaggedUser: (**nickname**, **idimage**)
nickname -> Account.username, idimage -> Post.idimage
User_Device: (**device**, **profile**)
device -> Device.iddevice, profile -> Account.username

4 Progettazione Fisica

In questa sezione, procederemo con la trasformazione della versione concettuale del sistema nella sua implementazione fisica mediante l'uso del DBMS PostgreSQL. Qui, definiremo tutte le classi, le relative associazioni, i vincoli, i domini, nonché i trigger e le funzioni necessari per il corretto funzionamento del sistema.

4.1 Creazione dei domini

```
-- Definizione del dominio per gli indirizzi email
CREATE DOMAIN emailt AS VARCHAR(64)
    CONSTRAINT email_check CHECK ((VALUE)::TEXT LIKE '_%@_%._%
   '::TEXT);

-- Definizione del dominio per le risoluzioni
CREATE DOMAIN resolutiont AS VARCHAR(16)
    CONSTRAINT resolution_check CHECK ((VALUE)::TEXT LIKE '_%
   x_%'::TEXT);

-- Definizione del tipo enumerato per lo stato dei post
CREATE TYPE statust AS ENUM ('Public', 'Private');
```

4.2 Schema delle tabelle

Questa sezione elenca le definizioni delle tabelle utilizzate nel sistema, insieme ai loro vincoli.

4.2.1 Account

```
--tabella Account
CREATE TABLE ACCOUNT
(
    username VARCHAR(16)  PRIMARY KEY,
    email    emailT       NOT NULL UNIQUE,
    name     VARCHAR(16)  NOT NULL,
    password VARCHAR(128) NOT NULL,
    gender   VARCHAR(30),
    bio      TEXT,
    avatar   BYTEA,
    salt     VARCHAR(32)  NOT NULL
);
```

4.2.2 Device

```
--tabella Device
CREATE TABLE DEVICE
(
    idDevice    SERIAL PRIMARY KEY,
    brand       VARCHAR(64) NOT NULL,
    model       VARCHAR(64) NOT NULL,
    deviceType VARCHAR(30) NOT NULL
);
```

4.2.3 Post

```
--tabella Post
CREATE TABLE POST
(
    idImage       SERIAL PRIMARY KEY,
    photo         VARCHAR(80)       NOT NULL,
    resolution    resolutionT       NOT NULL,
    size          INTEGER           NOT NULL,
    extension     VARCHAR(10)       NOT NULL,
    posting_date TIMESTAMP
        DEFAULT CURRENT_TIMESTAMP NOT NULL,
    preview       VARCHAR(80)       NOT NULL,
    description   TEXT,
    status        statusT           NOT NULL,
    device        INTEGER           NOT NULL
        REFERENCES DEVICE,
    profile       VARCHAR(16)
        REFERENCES ACCOUNT
);
```

4.2.4 Location

```
--tabella Location
CREATE TABLE LOCATION
(
    idlocation         serial
        PRIMARY KEY,
    country            VARCHAR(64),
    state              VARCHAR(64),
    city               VARCHAR(64),
    postcode           VARCHAR(64),
    latitude           NUMERIC(9, 6) NOT NULL,
    longitude          NUMERIC(9, 6) NOT NULL,
    formatted_address VARCHAR(200)  NOT NULL,
    road               VARCHAR(64),
    post               INTEGER
        REFERENCES post
);
```

4.2.5 Bookmark

```
--tabella Bookmark
CREATE TABLE BOOKMARK
(
    username VARCHAR (16) REFERENCES ACCOUNT
    ON DELETE CASCADE ,
    idImage INTEGER REFERENCES POST
    ON DELETE CASCADE ,
    date TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    PRIMARY KEY ( username , idImage )
);
```

4.2.6 Tagged_User

```
--tabella Tagged_User
CREATE TABLE TAGGED_USER
(
    nickname VARCHAR(16)
        REFERENCES ACCOUNT
            ON DELETE CASCADE,
    idImage  INTEGER
        REFERENCES POST
            ON DELETE CASCADE,
    PRIMARY KEY (nickname, idImage)
);
```

4.2.7 Following

```
--tabella Following
CREATE TABLE FOLLOWING
(
    nickname    VARCHAR(16)                              NOT NULL
        REFERENCES account,
    idfollowing VARCHAR(16)                              NOT NULL
        REFERENCES account,
    followdate  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    PRIMARY KEY (nickname, idfollowing),
    CONSTRAINT prevent_self_following
        CHECK ((nickname)::text <> (idfollowing)::text)
);
```

4.2.8 Collection

```
--tabella Collection
CREATE TABLE COLLECTION
(
    idCollection SERIAL PRIMARY KEY,
    name         VARCHAR(64) DEFAULT 'New Album' NOT NULL,
    description  TEXT,
    owner        VARCHAR(16)
        REFERENCES ACCOUNT
);
```

4.2.9 Collection_Post

```
--tabella Collection_Post
CREATE TABLE COLLECTION_POST
(
    collection SERIAL
        REFERENCES COLLECTION
            ON DELETE CASCADE,
    post       SERIAL
        REFERENCES POST,
    date       TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    PRIMARY KEY (collection, post)
);
```

4.2.10 Subject

```
--tabella Subject
CREATE TABLE SUBJECT
(
    subject_id SERIAL PRIMARY KEY,
    name       VARCHAR(32) NOT NULL,
    category   VARCHAR(32) NOT NULL
);
```

4.2.11 Subject_Post

```
--tabella Subject_Post
CREATE TABLE SUBJECT_POST
(
    subject INTEGER NOT NULL
        REFERENCES SUBJECT
            ON DELETE CASCADE,
    post    INTEGER NOT NULL
        REFERENCES POST
            ON DELETE CASCADE,
    PRIMARY KEY (subject, post)
);
```

4.2.12 User_Device

```
--tabella User_Device
CREATE TABLE USER_DEVICE
(
    device  INTEGER     NOT NULL
        REFERENCES DEVICE,
    profile VARCHAR(16) NOT NULL
        REFERENCES ACCOUNT,
    PRIMARY KEY (device, profile)
);
```

4.3 Defizione delle View

In seguito vengono mostrate le definizioni delle view

4.3.1 Collection_Stats

```
--View per le statistiche della collezione
CREATE OR REPLACE VIEW collection_stats(idcollection, posts,
   members) AS
SELECT collection.idcollection,
       count(post.idimage)          AS posts,
       count(DISTINCT post.profile) AS members
FROM collection
         LEFT JOIN collection_post
                   ON collection.idcollection =
   collection_post.collection
         LEFT JOIN post
                   ON collection_post.post = post.idimage
GROUP BY collection.idcollection;
```

4.3.2 User_Stats

```
--View per le statistiche pubbliche dell'utente
CREATE OR REPLACE VIEW user_stats(username, post, follower,
   following) AS
SELECT a.username,
       (SELECT count(post.idImage) AS post
        FROM post
        WHERE post.profile = a.username
          AND post.status = 'Public')             AS post,
       (SELECT count(*) AS follower
        FROM following
        WHERE following.idFollowing = a.username) AS follower,
       (SELECT count(*) AS following
        FROM following
        WHERE following.nickname = a.username)    AS following
FROM ACCOUNT a;
```

4.4 Definizione dei Trigger

In questa sezione viene fornita un'implementazione dei trigger.

4.4.1 Rimozione post privati

Quando lo stato di un post viene modificato da pubblico a privato, il trigger provvede a rimuovere il post dalle eventuali collezioni e bookmark in cui è stato salvato.

```
CREATE FUNCTION delete_post_from_collections_and_bookmarks()
   RETURNS TRIGGER
    LANGUAGE PLPGSQL
AS
$$
BEGIN
    IF OLD.status = 'Public'::STATUST AND NEW.status = '
   Private'::STATUST THEN
        DELETE
        FROM collection_post
        WHERE collection_post.post = OLD.idimage;

        DELETE
        FROM bookmark
        WHERE bookmark.idimage = OLD.idimage;
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER delete_from_collections_and_bookmarks_trigger
    BEFORE UPDATE OF status
    ON post
    FOR EACH ROW
EXECUTE PROCEDURE delete_post_from_collections_and_bookmarks()
   ;
```

4.4.2 Rimozione dei post presenti nelle collezioni

Quando un post viene cancellato, il trigger verifica se il post è ancora associato a una collezione. Se il post è ancora in una collezione, il campo "profile" del post viene impostato su NULL rimuovendo il post dall'profilo dell'utente, ma mantenendo intatta l'associazione con la collezione. D'altra parte, se il post non è associato a nessuna collezione (post orfano), verrà eliminato definitivamente.

```
CREATE FUNCTION update_post_profile() RETURNS TRIGGER
    LANGUAGE PLPGSQL
AS
$$
DECLARE
    post_id INTEGER = NULL;
```

```
7  BEGIN
8      SELECT post INTO post_id FROM collection_post WHERE post =
       OLD.idimage;
9
10     IF (post_id IS NOT NULL) THEN
11         UPDATE post
12         SET profile = NULL
13         WHERE post.idimage = OLD.idimage;
14     ELSE
15         DELETE FROM post WHERE idimage = OLD.idimage;
16     END IF;
17
18     RETURN NEW;
19 END;
20 $$;
21
22 CREATE TRIGGER update_post_profile_trigger
23     BEFORE DELETE
24     ON post
25     FOR EACH ROW
26     WHEN (pg_trigger_depth() = 0)
27 EXECUTE PROCEDURE update_post_profile();
```

*Nota: Dato che la funzione richiamata dal trigger effettua operazioni di cancellazione sulla tabella Post, il trigger verrebbe riattivato ad ogni esecuzione della funzione, creando una catena di chiamate ricorsive senza fine. Utilizzando la funzione pg_trigger_depth() alla riga 26 per controllare il livello di ricorsione, il trigger viene eseguito solo quando il suo livello di profondità è zero, ovvero quando non è già stato attivato da altre chiamate. In questo modo, si evita la catena di ricorsione infinita.

4.4.3 Rimozione di post cancellati dalle collezioni

Questo trigger si attiva quando una collezione viene eliminata. La sua funzione è quella di rimuovere i post cancellati dall'utente che non sono presenti in nessuna altra collezione. Se un post è stato cancellato e non è incluso in nessun'altra collezione, verrà rimosso definitivamente dal database.

```
1  CREATE FUNCTION delete_orphaned_posts() RETURNS TRIGGER
2      LANGUAGE PLPGSQL
3  AS
4  $$
5  BEGIN
6      DELETE
7      FROM post
8      WHERE profile IS NULL
9        AND idimage NOT IN (SELECT post
10                           FROM collection_post);
11
12     RETURN OLD;
13 END;
```

```
$$;

CREATE TRIGGER delete_collection_trigger
    AFTER DELETE
    ON collection
    FOR EACH ROW
EXECUTE PROCEDURE delete_orphaned_posts();
```

4.4.4 Rimozione soggetti "orfani"

Il trigger si assicura che i soggetti non associati a nessun post vengano rimossi dalla tabella "Subject", mantenendo solo i soggetti effettivamente collegati ai post presenti nel database.

```
CREATE FUNCTION delete_subject() RETURNS TRIGGER
    LANGUAGE PLPGSQL
AS
$$
BEGIN
    DELETE
    FROM subject
    WHERE subject_id NOT IN (SELECT subject
                             FROM subject_post);

    RETURN OLD;
END;
$$;

CREATE TRIGGER delete_subject_trigger
    AFTER DELETE
    ON subject_post
    FOR EACH ROW
EXECUTE PROCEDURE delete_subject();
```

4.4.5 Hashing delle password

La funzione hash_password crea una stringa di hash a partire dalla password dell'account e genera un salt per la crittografia.
La funzione gen_salt viene utilizzata per generare il salt e accetta come parametro l'algoritmo di crittografia da utilizzare, in questo caso 'bf' per Blowfish.
La funzione crypt viene utilizzata per criptare la password con il salt generato e restituisce una stringa di hash.

```
--funzione per fare il hash della password prima di inserirla
CREATE EXTENSION IF NOT EXISTS pgcrypto;

CREATE OR REPLACE FUNCTION hash_password() RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
```

```
BEGIN
    NEW.salt := gen_salt('bf');
    NEW.password := crypt(NEW.password, NEW.salt);
    RETURN NEW;
END;
$$;

CREATE TRIGGER hash_password_trigger
    BEFORE INSERT OR UPDATE OF password
    ON account
    FOR EACH ROW
EXECUTE PROCEDURE hash_password();
```

4.5 Definizione delle funzioni

In questa sezione sono elencate le definizioni di tutte le funzioni utilizzate.

4.5.1 Inserimento del soggetto

Questa funzione prende come parametro il soggetto, la categoria e l'identificativo del post da associare al soggetto, controlla se il soggetto esiste per la determinata categoria, nel caso positivo recupera l'identificativo del soggetto e lo associa al post, altrimenti viene creato un nuovo soggetto.

```
--funzione per inserire un soggetto nel post
CREATE OR REPLACE FUNCTION insert_subject_post(subject_name
   character varying, category_name character varying,
                                                image_id
   integer) RETURNS VOID
    LANGUAGE plpgsql
AS
$$
DECLARE
    subjectID INTEGER;
BEGIN
    SELECT subject_id
    INTO subjectID
    FROM subject
    WHERE name = subject_name
      AND category = category_name;
    IF (subjectID IS NULL) THEN
        INSERT INTO subject (name, category)
        VALUES (subject_name, category_name)
        RETURNING subject_id INTO subjectID;
    END IF;

    INSERT INTO subject_post (subject, post)
    VALUES (subjectID, image_id);
END;
$$;
```

4.5.2 Inserimento nuovo account

Questa funzione è usata per inserire un nuovo utente nel sistema durante la registrazione. Verifica se esiste già un utente con lo stesso username o la stessa email e restituisce un valore intero a seconda dell'esito.

```
--funzione per inserire un nuovo utente durante la registrazione
CREATE OR REPLACE FUNCTION insert_account(p_username character varying, p_email emailt, p_name character varying,
                                          p_password character varying) RETURNS INTEGER
    LANGUAGE plpgsql
AS
$$
DECLARE
    username_c INTEGER;
    email_c    INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO username_c
    FROM account
    WHERE username = p_username;

    SELECT COUNT(*)
    INTO email_c
    FROM account
    WHERE email = p_email;

    IF username_c = 0 AND email_c = 0 THEN
        INSERT INTO account (username, name, password, email)
        VALUES (p_username, p_name, p_password, p_email);
        RETURN 1;
    ELSIF username_c > 0 AND email_c > 0 THEN
        RETURN 0;
    ELSIF username_c > 0 THEN
        RETURN -1;
    ELSE
        RETURN -2;
    END IF;
END;
$$;
```