

UNIVERSITÀ POLITECNICA DELLE MARCHE  
INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

---

**Sviluppo di Inkspire**  
**Applicazione per generare sfide artistiche**



Corso di  
PROGRAMMAZIONE MOBILE  
Anno accademico 2024-2025

Studente:  
Alessia Capancioni

Professore:  
Emanuele Storti



Dipartimento di Ingegneria dell'Informazione

# INDICE

<b>1. Introduzione.....</b>	<b>4</b>
<b>1.1. Descrizione in linguaggio naturale .....</b>	<b>4</b>
<b>1.2. Obiettivo del progetto .....</b>	<b>4</b>
<b>2. Progettazione e Sviluppo Android .....</b>	<b>5</b>
<b>2.1. Sintesi dell'approccio .....</b>	<b>5</b>
<b>2.2. Glossario dei termini.....</b>	<b>6</b>
<b>2.3. Requisiti .....</b>	<b>7</b>
<b>2.3.1. Requisiti funzionali.....</b>	<b>7</b>
<b>2.3.2. Requisiti non funzionali.....</b>	<b>7</b>
<b>2.3.3. Casi d'uso .....</b>	<b>8</b>
<b>2.4. Mockup UI .....</b>	<b>14</b>
<b>2.4.1. Accesso e uscita dall'app .....</b>	<b>15</b>
<b>2.4.2. Gestione Challenge .....</b>	<b>16</b>
<b>2.4.3. Profilazione.....</b>	<b>19</b>
<b>2.4.4. Community .....</b>	<b>20</b>
<b>2.5. Database e memorizzazione dati.....</b>	<b>21</b>
<b>2.5.1. Modello relazionale.....</b>	<b>21</b>
<b>2.5.2. Struttura delle tabelle in Supabase .....</b>	<b>22</b>
<b>2.5.3. Viste per aggregazione dati.....</b>	<b>24</b>
<b>2.5.4. Autenticazione con Supabase Auth .....</b>	<b>25</b>
<b>2.5.5. Policies di sicurezza .....</b>	<b>25</b>
<b>2.5.6. Supabase Storage .....</b>	<b>26</b>
<b>2.6. Architettura .....</b>	<b>27</b>
<b>2.6.1. MVVM Pattern .....</b>	<b>27</b>
<b>2.6.2. Repository Pattern.....</b>	<b>28</b>
<b>2.7. Sviluppo.....</b>	<b>29</b>
<b>2.7.1. Panoramica dello sviluppo .....</b>	<b>29</b>
<b>2.7.2. Struttura del progetto.....</b>	<b>30</b>
<b>2.7.3. Layout e UI.....</b>	<b>33</b>
<b>2.7.4. Autenticazione, Storage e accesso al DB .....</b>	<b>35</b>
<b>2.7.5. Model .....</b>	<b>37</b>
<b>2.7.6. Repository.....</b>	<b>38</b>
<b>2.7.7. ViewModel.....</b>	<b>40</b>

<b>2.7.8. Adapter .....</b>	<b>41</b>
<b>2.7.9. Activity e Fragment .....</b>	<b>42</b>
<b>2.7.10. Navigazione .....</b>	<b>44</b>
<b>2.8. Problematiche e considerazioni finali.....</b>	<b>45</b>
<b>Bibliografia.....</b>	<b>47</b>

# **1. Introduzione**

## **1.1. Descrizione in linguaggio naturale**

Nel contesto dell'arte digitale e tradizionale, la motivazione e l'ispirazione giocano un ruolo fondamentale per ogni artista. Per quanto grande possa essere la passione, può diventare difficile trovare nuove idee che spingano a disegnare con costanza.

Inkspire nasce proprio come risposta a questa esigenza: una app che aiuta gli utenti a generare sfide artistiche creative e a condividerle con una community di artisti.

L'applicazione permette di creare una sfida (challenge) definendo un titolo, un tema (concept) e un vincolo artistico (art constraint). Questi ultimi due attributi possono essere scelti liberamente o generati casualmente. In particolare, il vincolo artistico è una caratteristica speciale della sfida che impone una limitazione (ad esempio l'uso di soli tre colori o un limite temporale), aggiungendo un livello di difficoltà che spinge l'utente a trovare soluzioni innovative. Ogni sfida può essere arricchita con una descrizione e un'immagine del lavoro artistico realizzato, sia durante la creazione che in un secondo momento. L'app comprende inoltre un profilo utente che raccoglie tutte le sfide create e completate, offrendo un modo pratico per conservare i propri lavori e osservare i progressi e i cambiamenti artistici nel tempo.

Inkspire è quindi pensata per offrire uno spazio creativo dove gli utenti possano ispirarsi e migliorare le proprie competenze attraverso l'utilizzo di concetti e vincoli originali e la condivisione dei propri lavori.

## **1.2. Obiettivo del progetto**

L'obiettivo di questo progetto è la progettazione e lo sviluppo di Inkspire su Android. L'applicazione deve consentire agli utenti di:

- generare sfide artistiche personalizzate o casuali;
- completare le sfide caricando un'immagine e una descrizione facoltativa;
- consultare le sfide della community;
- gestire il proprio profilo e visualizzare le proprie attività artistiche.

Tutto questo cercando di garantire un'interfaccia intuitiva per un utilizzo semplice e immediato di tutte le funzionalità.

## 2. Progettazione e Sviluppo Android

### 2.1. Sintesi dell'approccio

La progettazione di Inkspire si è articolata in due fasi principali: una prima fase concettuale e una successiva fase implementativa.

Durante la fase concettuale sono stati identificati:

- gli obiettivi funzionali;
- i principali casi d'uso;
- la struttura dei dati e il modello relazionale;
- i flussi di navigazione tra le schermate principali.

Questa fase ha permesso di definire una base solida e chiara, utile a guidare lo sviluppo successivo. La fase implementativa ha invece previsto:

- l'adozione di un'architettura MVVM per separare la logica di business dall'interfaccia grafica;
- l'utilizzo di Supabase come backend per la gestione di autenticazione, database PostgreSQL e storage delle immagini;
- l'integrazione di Ktor per la comunicazione REST con il backend;
- la progettazione di tutti i layout xml e dei temi personalizzati;
- l'organizzazione del progetto in package tematici (model, repository, viewmodel, fragments, adapter);
- l'implementazione della navigazione e del passaggio di parametri tra i fragment con Navigation Component e Safe Args.

L'approccio è stato incrementale e iterativo, alternando momenti di progettazione teorica a cicli di sviluppo pratico e verifica continua del corretto funzionamento delle funzionalità.

## 2.2. Glossario dei termini

Termine	Descrizione	Sinonimo
Challenge	Sfida artistica creata da un utente, composta da titolo, tema, vincolo artistico e contenuti opzionali (descrizione e immagine). Una challenge è completa solo se ha associata un'immagine.	Sfida artistica
Concept	Tema ispiratore di una sfida, può essere generato casualmente o definito dall'utente (es. viaggio nel tempo, metamorfosi).	Tema, Concetto della sfida
Art Constraint	Vincolo artistico che introduce una limitazione creativa, può essere generato casualmente o definito dall'utente (es. solo tre colori, divieto di linee curve).	Vincolo artistico, Restrizione
User Profile	Profilo di un utente registrato, contenente username, foto profilo, breve descrizione (bio), statistiche ed elenco di tutte le challenge create.	Profilo utente
Community	Insieme di tutti gli utenti registrati che utilizzano l'app.	Insieme degli Utenti/Artisti.

## 2.3. Requisiti

### 2.3.1. Requisiti funzionali

- **RF1. Registrazione e Login:** l'utente deve poter registrarsi immettendo username, email e password e successivamente autenticarsi tramite email e password.
- **RF2. Visualizzazione elenco challenge:** l'utente deve poter consultare l'elenco delle challenge della community e dei singoli profili.
- **RF3. Creazione di una nuova challenge:** l'utente deve poter creare una nuova sfida inserendo gli attributi obbligatori titolo, tema e vincolo artistico.
- **RF3.1. Generazione casuale di tema e vincolo artistico:** l'utente deve poter generare casualmente il tema e il vincolo artistico della sfida tramite dei pulsanti dedicati.
- **RF3.2 Inserimento descrizione e immagine:** l'utente deve poter aggiungere una descrizione facoltativa e caricare un'immagine del lavoro artistico realizzato, al fine di rendere la challenge completa.
- **RF4. Visualizzazione dettagli challenge:** l'utente deve poter consultare la schermata di dettaglio di una challenge con tutte le informazioni relative ad essa e, se non appartiene a lui, all'utente che l'ha creata.
- **RF5. Modifica challenge:** l'utente deve poter modificare una challenge creata in precedenza aggiornando titolo, tema, vincolo, descrizione e immagine.
- **RF6. Eliminazione challenge:** l'utente deve poter eliminare una challenge creata in precedenza.
- **RF7. Visualizzazione profilo utente:** l'utente deve poter visualizzare il proprio profilo, comprendente username, bio, statistiche ed elenco delle challenge create e completate.
- **RF8. Modifica profilo utente:** l'utente deve poter modificare la propria biografia e l'immagine del profilo.
- **RF9. Visualizzazione profili degli altri utenti:** l'utente deve poter consultare il profilo di un altro utente della community.
- **RF10. Logout:** l'utente deve poter disconnettersi dal proprio account.

### 2.3.2. Requisiti non funzionali

- **RNF1. UI:** l'interfaccia deve essere semplice e intuitiva, per garantire una buona esperienza utente.
- **RNF2. Sviluppo:** il linguaggio di sviluppo è Kotlin.
- **RNF3. Architettura:** l'architettura dell'app utilizza il pattern MVVM (Model-View-ViewModel).

- **RNF4. Persistenza dati online:** tutte le informazioni (utenti, sfide, immagini) sono salvate e sincronizzate online tramite Supabase (PostgreSQL + Storage + Auth).
- **RNF5. Visualizzazione immagini:** il caricamento e la visualizzazione delle immagini sono gestiti tramite la libreria Glide.
- **RNF6. Compatibilità:** l'app è compatibile con Android 10 (API 29) e versioni successive.
- **RNF7. Sicurezza:** le operazioni di autenticazione devono utilizzare protocolli sicuri (HTTPS) e token JWT gestiti tramite Supabase Auth.

### 2.3.3. Casi d'uso

Per descrivere le funzionalità principali che il sistema deve offrire e le sue risposte in base alle azioni compiute dall'utente, viene riportato di seguito l'elenco dei casi d'uso, il quale costituisce una base per la progettazione dell'architettura, la realizzazione dell'interfaccia e la scrittura dei test di verifica dell'app.

I casi d'uso sono divisi in due macro aree: Gestione Utente e Gestione Challenge:

**Gestione Utente:** questi casi d'uso riguardano la gestione dell'autenticazione e del profilo personale dell'utente.

- **UC1. Registrazione utente**

Si verifica quando l'utente vuole creare un nuovo account.

Pre-condizioni: non deve esistere un altro account utente con lo stesso indirizzo email o lo stesso username e la password deve essere valida (almeno 6 caratteri).

Post-condizioni: l'utente è registrato nel sistema e può effettuare il login.

Sequenza di eventi principali:

1. l'utente avvia la registrazione dall'apposita schermata;
2. il sistema permette di inserire email, username e password;
3. l'utente conferma;
4. il sistema crea il nuovo account e reindirizza l'utente alla schermata di login.

- **UC2. Login utente**

Si verifica quando l'utente vuole accedere all'account.

Pre-condizioni: l'utente è già registrato e inserisce le credenziali corrette.

Post-condizioni: l'utente accede all'app.

Sequenza di eventi principali:

1. l'utente accede alla schermata di login;
2. il sistema permette di inserire le credenziali (email e password);
3. l'utente conferma;



4. il sistema verifica i dati ed effettua l'accesso.

- **UC3. Logout utente**

Si verifica quando l'utente decide di uscire dal proprio account.

Pre-condizioni: l'utente è autenticato e si trova nella schermata del proprio Profilo.

Post-condizioni: l'utente viene disconnesso.

Sequenza di eventi principali:

1. l'utente seleziona il pulsante di logout dal menu del suo profilo;
2. il sistema chiede conferma e in caso affermativo esegue la disconnessione e riporta alla schermata di login.

- **UC4. Visualizzazione profilo personale**

Si verifica quando l'utente vuole visualizzare le informazioni del proprio profilo.

Pre-condizioni: l'utente è autenticato.

Post-condizioni: nessuna.

Sequenza di eventi principali:

1. l'utente accede alla sezione "Profile" del proprio profilo;
2. il sistema mostra la schermata del profilo personale con username, bio, immagine profilo, statistiche ed elenco delle challenge create dall'utente.

- **UC5. Modifica profilo**

Si verifica quando l'utente vuole modificare le informazioni del profilo.

Pre-condizioni: l'utente è autenticato e si trova nella schermata del proprio Profilo.

Post-condizioni: le informazioni vengono aggiornate.

Sequenza di eventi principali:

1. l'utente accede alla schermata di modifica del proprio profilo;
2. il sistema permette di modificare bio e immagine profilo;
3. l'utente salva;
4. il sistema aggiorna i dati e riporta alla schermata del profilo.

- **UC6. Visualizzazione profilo di un altro utente**

Si verifica quando l'utente vuole consultare il profilo pubblico di un altro utente.

Pre-condizioni: l'utente è autenticato e si trova o nella schermata di visualizzazione di una challenge di un altro utente, o nella sezione "Artists" contenente l'elenco di tutti gli utenti registrati.

Post-condizioni: nessuna.

Sequenza di eventi principali:

1. l'utente seleziona il nome di un altro utente;
2. il sistema visualizza in lettura il profilo di tale utente selezionato.

- **UC7. Visualizzazione elenco utenti**

Si verifica quando l'utente si trova nella schermata "Artists".

Pre-condizioni: l'utente è autenticato.

Post-condizioni: nessuna.

Sequenza di eventi principali:

1. l'utente accede alla sezione "Artists";
2. il sistema mostra l'elenco di tutti gli utenti registrati per ordine di numero di challenge completate.

**Gestione Challenge:** questi casi d'uso riguardano la creazione, consultazione e gestione delle challenge

- **UC8. Visualizzazione elenco challenge**

Si verifica quando l'utente si trova nella schermata "Home" o in una schermata di Profilo, che sia il suo o di un altro utente.

Pre-condizioni: l'utente è autenticato.

Post-condizioni: nessuna.

Sequenza di eventi principali:

3. l'utente accede alla sezione "Home";
4. il sistema mostra l'elenco di tutte le challenge create dalla community.

- **UC9. Ricerca challenge**

Si verifica quando l'utente vuole filtrare la lista di tutte le challenge.

Pre-condizioni: l'utente si trova nella schermata "Home".

Post-condizioni: la lista viene filtrata.

Sequenza di eventi principali:

1. l'utente inserisce un termine di ricerca nella barra di ricerca del menu;
2. il sistema filtra le challenge in base al testo inserito.

- **UC10. Visualizzazione dettaglio challenge**

Si verifica quando l'utente seleziona una challenge dalla lista presente nella Home o in una schermata di Profilo per vederne i dettagli.

Pre-condizioni: la challenge deve essere presente nel sistema.

Post-condizioni: nessuna.

Sequenza di eventi principali:

1. l'utente seleziona una challenge dall'elenco;
2. il sistema mostra tutte le informazioni riguardanti quella challenge, quali: titolo, tema, vincolo, descrizione, immagine e autore.

- **UC11. Creazione challenge**

Si verifica quando l'utente vuole creare una nuova sfida.

Pre-condizioni: l'utente è autenticato e si trova in una delle schermate principali.

Post-condizioni: la challenge viene inserita nel sistema.

Sequenza di eventi principali:

1. l'utente preme il pulsante "+" del Fab;
2. il sistema mostra la form per la creazione di una nuova challenge;
3. l'utente inserisce titolo, tema, vincolo ed eventuali descrizione e immagine;
4. l'utente conferma;
5. il sistema salva la challenge.

- **UC12. Generazione concept/vincolo casuali**

Si verifica quando l'utente vuole generare un tema/vincolo casuale.

Pre-condizioni: l'utente si trova nel modulo di creazione o modifica di una challenge.

Post-condizioni: il campo "concept/art constraint" viene compilato automaticamente.

Sequenza di eventi principali:

1. l'utente preme l'icona di generazione casuale;
2. il sistema propone un tema/vincolo casuale.

- **UC13 Modifica challenge**

Si verifica quando l'utente vuole modificare una challenge creata in precedenza.

Pre-condizioni: la challenge è stata creata dall'utente.

Post-condizioni: la challenge viene aggiornata.

Sequenza di eventi principali:

1. l'utente seleziona la propria challenge dal profilo o dall'elenco nella schermata Home;
2. il sistema mostra la form di modifica della challenge;
3. l'utente aggiorna i dati;
4. l'utente conferma;
5. il sistema salva le modifiche.

- **UC14. Eliminazione Challenge**

Si verifica quando l'utente vuole eliminare una challenge creata.

Pre-condizioni: la challenge è stata creata dall'utente.

Post-condizioni: la challenge viene rimossa.

Sequenza di eventi principali:

1. l'utente seleziona la propria challenge;
2. l'utente preme l'icona di eliminazione;
3. il sistema chiede conferma;
4. l'utente conferma;
5. il sistema elimina la challenge.

- **UC15. Completamento challenge**

Si verifica quando l'utente vuole aggiungere l'immagine del risultato alla challenge.

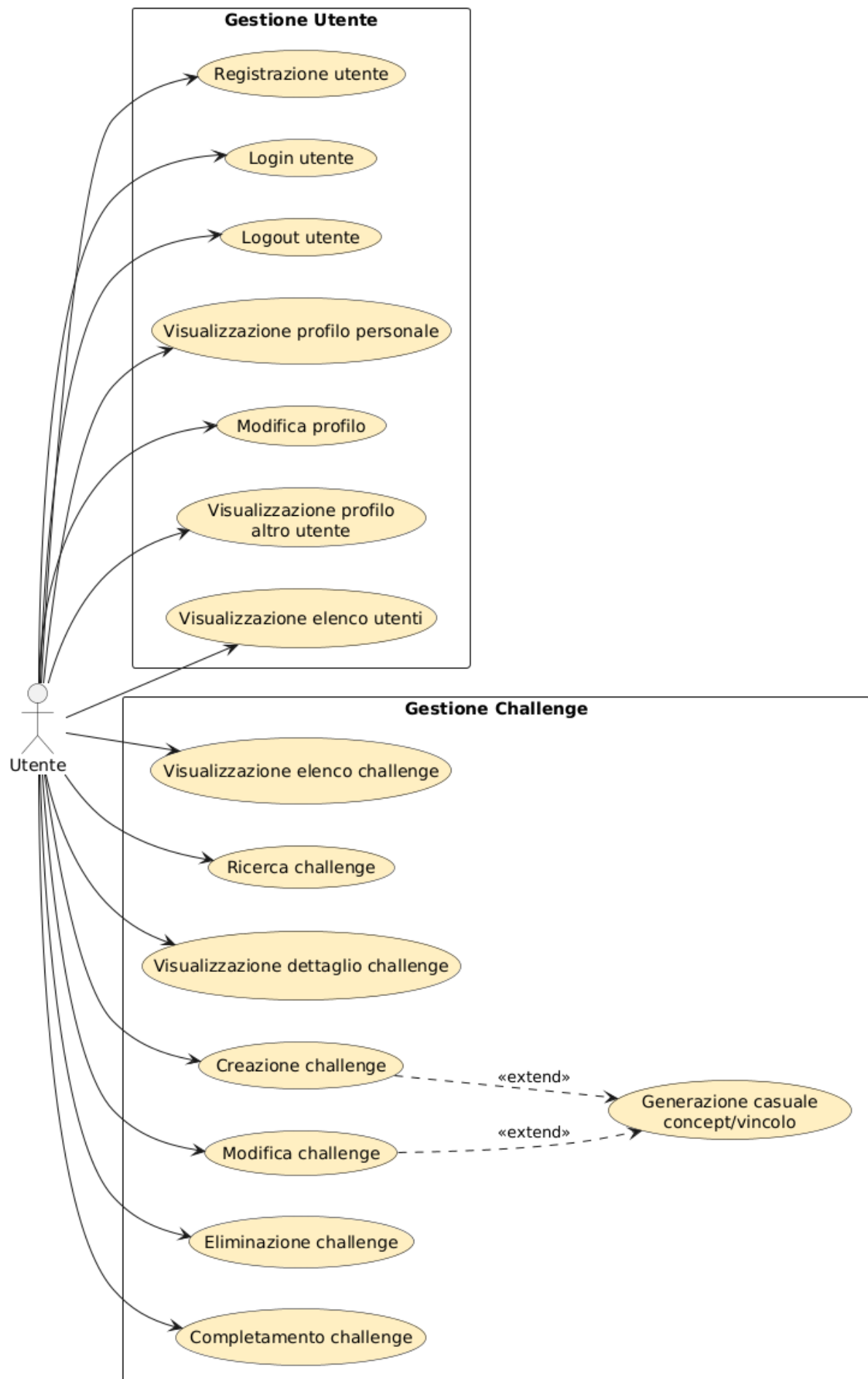
Pre-condizioni: la challenge esiste o, se in fase di creazione, presenta già tutti gli attributi obbligatori compilati.

Post-condizioni: la challenge presenta un'immagine e viene considerata completata.

Sequenza di eventi principali:

1. l'utente si trova nella forma di creazione o modifica di una sua challenge;
2. l'utente carica l'immagine del lavoro artistico;
3. l'utente conferma;
4. il sistema salva/aggiorna i dati.

Di seguito è riportato il diagramma UML dei casi d'uso, realizzato con l'editor online PlantUML:

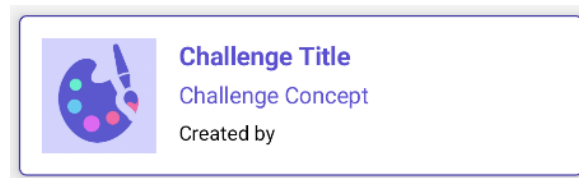


## 2.4. Mockup UI

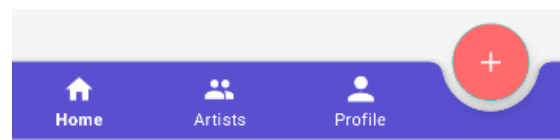
Di seguito sono riportate le immagini del mockup dell'interfaccia di Inkspire con la rappresentazione grafica dei casi d'uso.

Dopo aver effettuato la registrazione e successivamente il login, l'utente accede direttamente alla Home, la schermata principale dell'app, che mostra l'elenco di tutte le challenge create dagli utenti. Da qui può:

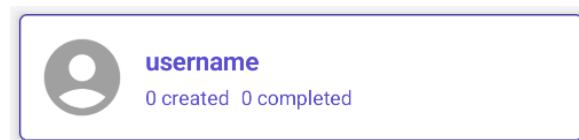
- visualizzare e filtrare tramite la barra di ricerca le challenge della community elencate nella sezione “Home”: cliccando sopra una card si entra nella schermata di visualizzazione dettaglio di quella challenge (View Challenge). Nel caso in cui la challenge sia propria, nella schermata di visualizzazione sarà presente anche un'icona con una matita per passare alla schermata di modifica (Edit Challenge);



- creare una nuova challenge cliccando sul pulsante “+” (Floating Action Button);



- visualizzare e filtrare tramite la barra di ricerca tutti gli utenti della community elencati nella sezione “Artists”: cliccando sopra una card si entra nella schermata di visualizzazione del profilo di quell'utente (Other User Profile);

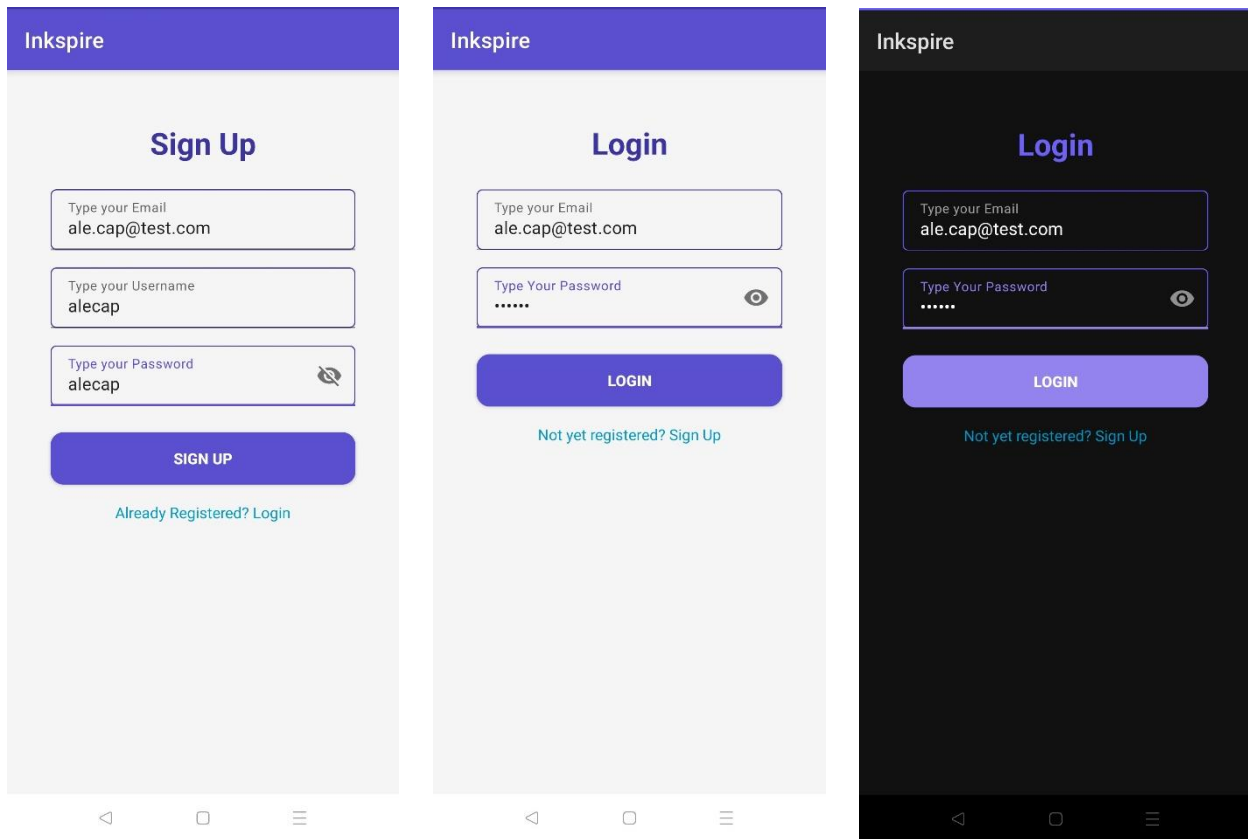


- visualizzare il proprio profilo personale navigando alla sezione “Profile”, dove sono mostrati username, foto profilo, bio, statistiche ed elenco delle proprie challenge e dove è possibile entrare nella schermata di modifica (Edit User Profile) per aggiornare foto profilo e bio.

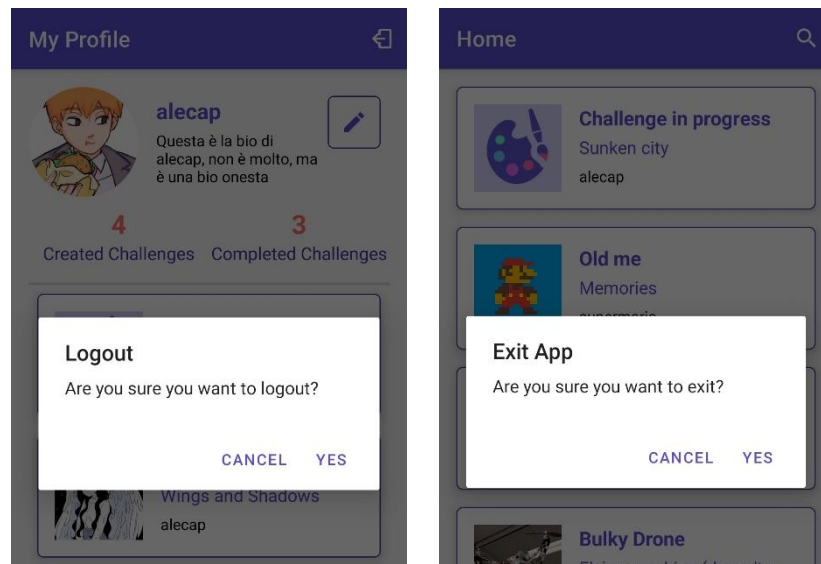
La navigazione avviene tramite la barra inferiore che consente di passare rapidamente alle sezioni principali, seguendo un flusso logico e intuitivo.

## 2.4.1. Accesso e uscita dall'app

### Schermate di SignUp/Login



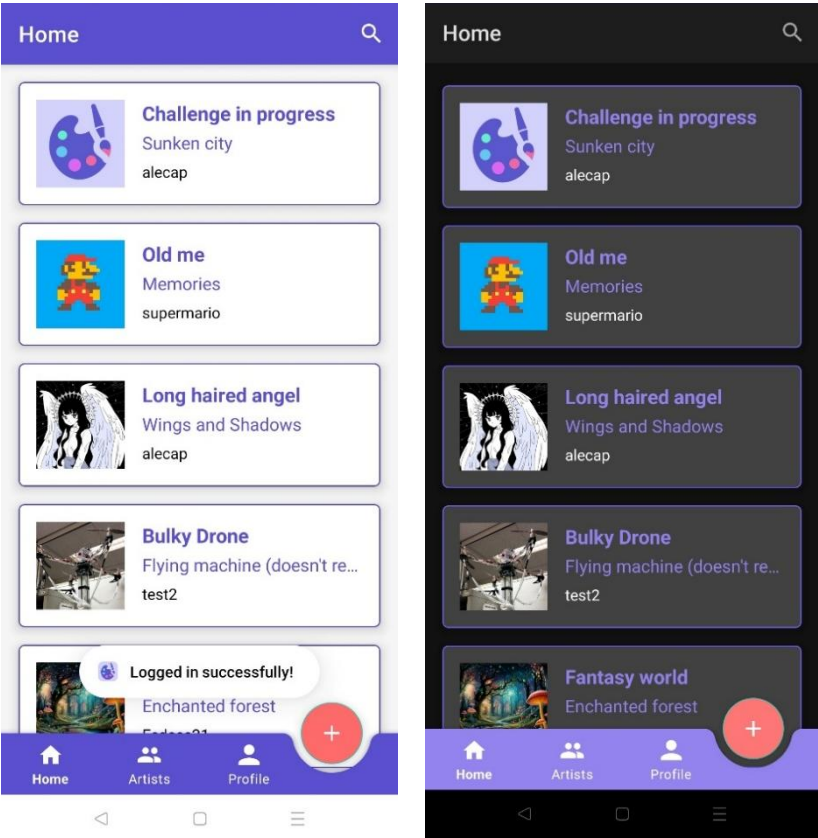
### Finestre di dialogo



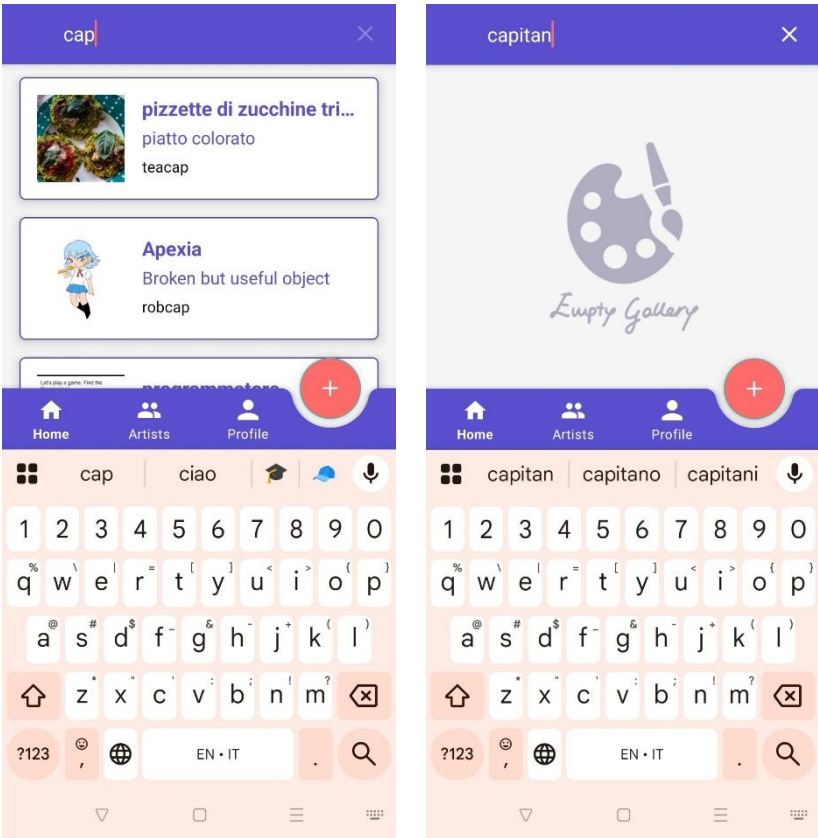
Il dialog di Logout compare appena l'utente clicca sul pulsante di logout nel menu del proprio profilo. Confermando, esso viene disconnesso e reindirizzato alla schermata di Login. Il dialog di Exit compare appena l'utente clicca il tasto "back" dalla schermata Home. Confermando esso esce dall'app senza disconnessione.

## 2.4.2. Gestione Challenge

- Schermata Home





- Ricerca Challenge






- **Schermata New Challenge**


← New Challenge


Upload Result Image  
(recommended size: 1080×1080 px)

Enter title



Enter concept 

Enter art constraint 

Enter description (optional)





← New Challenge


Upload Result Image  
(recommended size: 1080×1080 px)

Enter title

Enter concept 


Enter art constraint 


Enter description (optional)



- **Schermata View Challenge** (con collegamento al profilo dell'autore, raggiungibile cliccando sul suo username sotto l'immagine)

← View Challenge



 Fedeee21


**Fantasy world**


Concept: Enchanted forest

Constraint: Only 3 colors

No description

← View Challenge



 Fedeee21

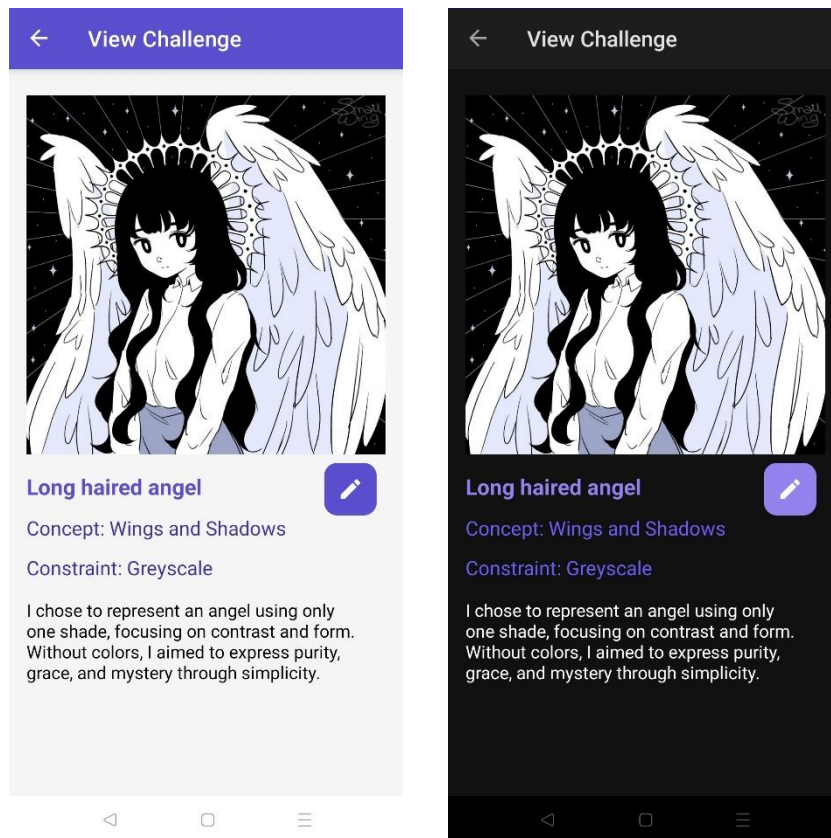
**Fantasy world**

Concept: Enchanted forest

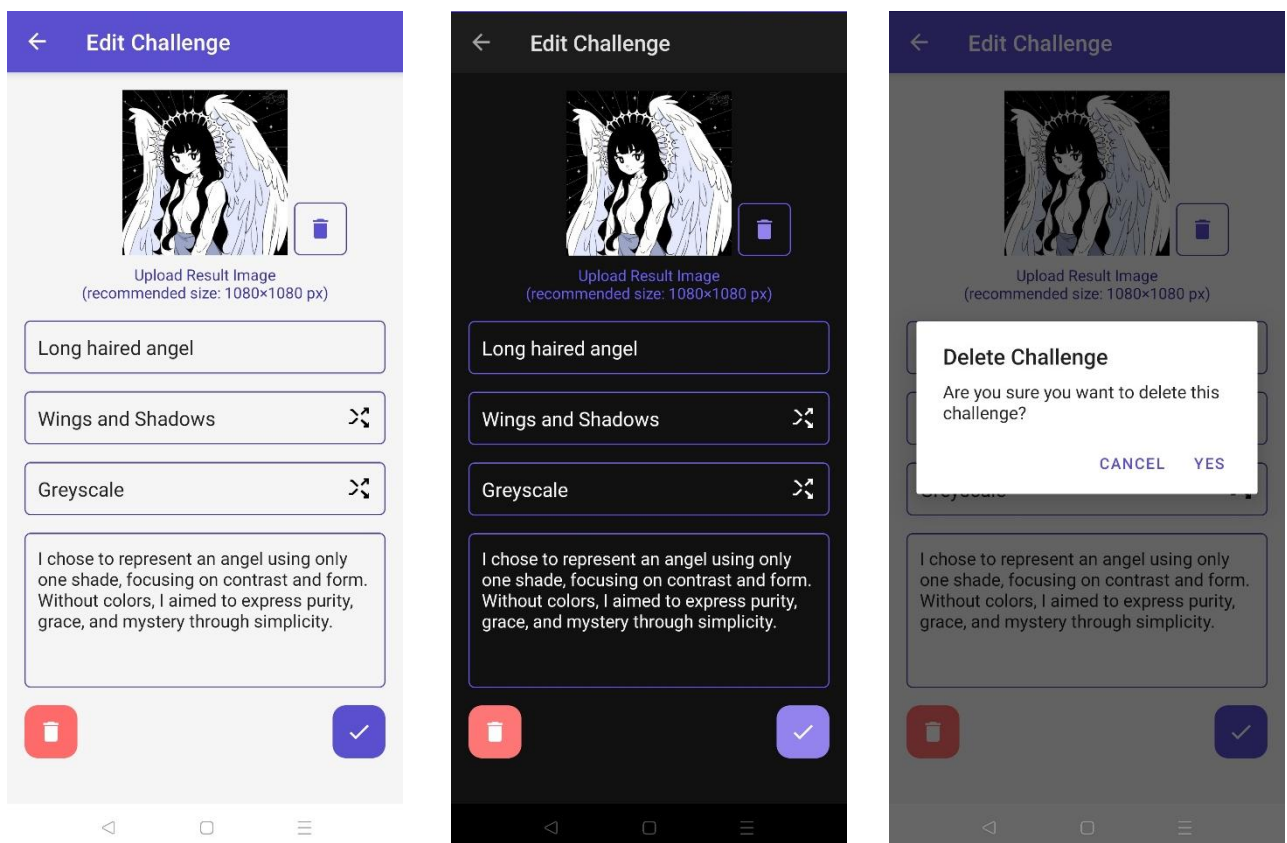
Constraint: Only 3 colors

No description

- **Schermata View Challenge** (propria challenge)

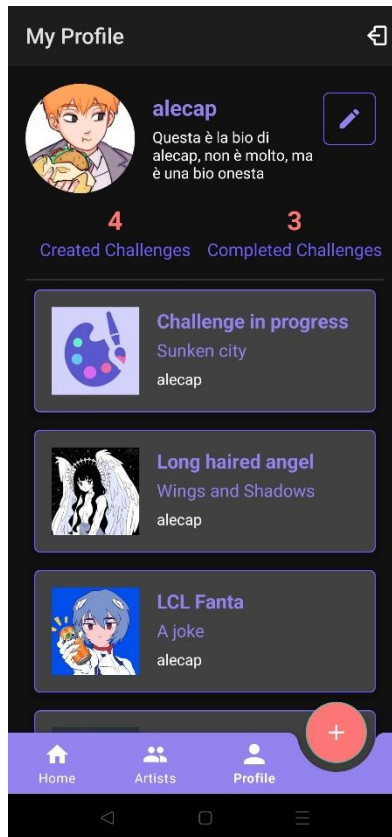
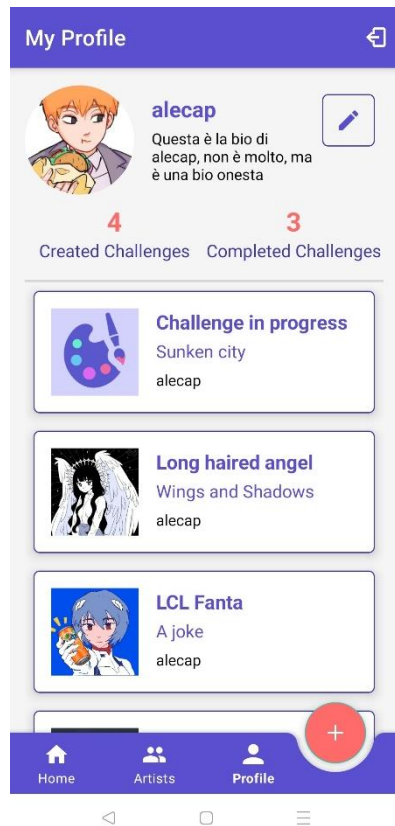


- **Schermata Edit Challenge** (raggiungibile cliccando sull'icona con la matita)

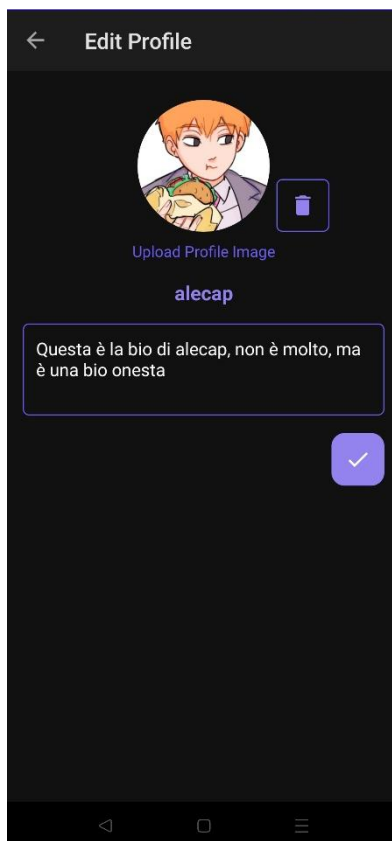


### 2.4.3. Profilazione

- **Schermata User Profile (proprio profilo)**

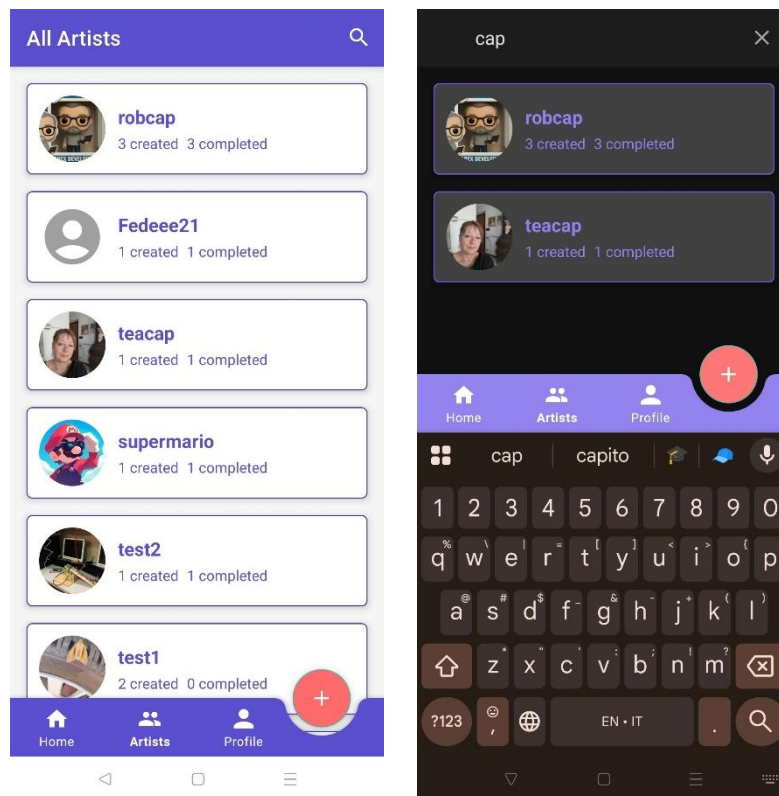


- **Schermata Edit User Profile**

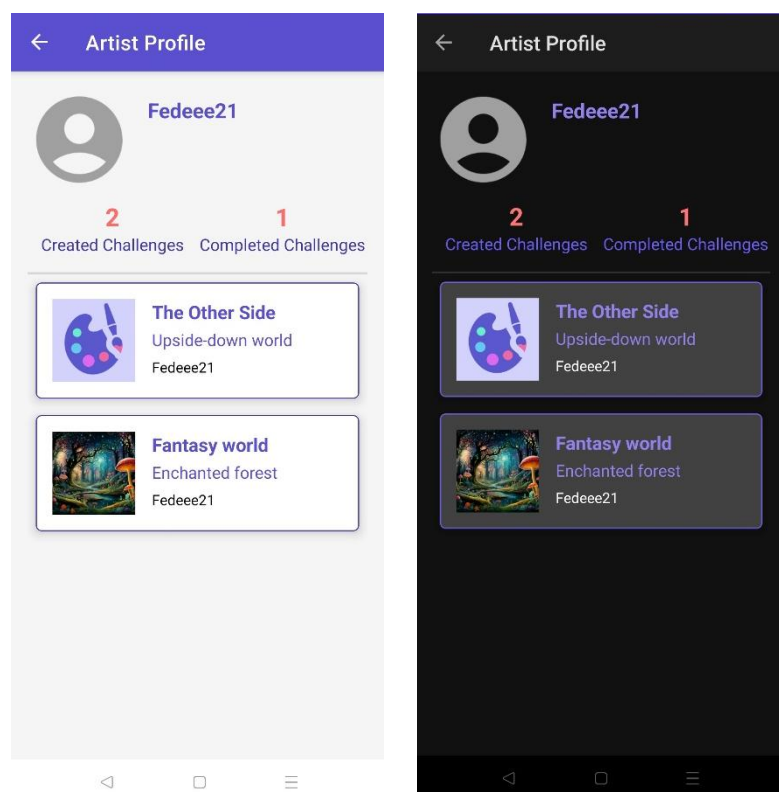


## 2.4.4. Community

- **Schermata Artists con Ricerca utenti**



- **Schermata Other User Profile** (profilo di un altro utente, raggiungibile o della sezione Artists o da una View Challenge di tale utente)



## 2.5. Database e memorizzazione dati

Inkspire utilizza **Supabase** come backend per la gestione del database relazionale PostgreSQL, dell'autenticazione utente e dello storage per le immagini.

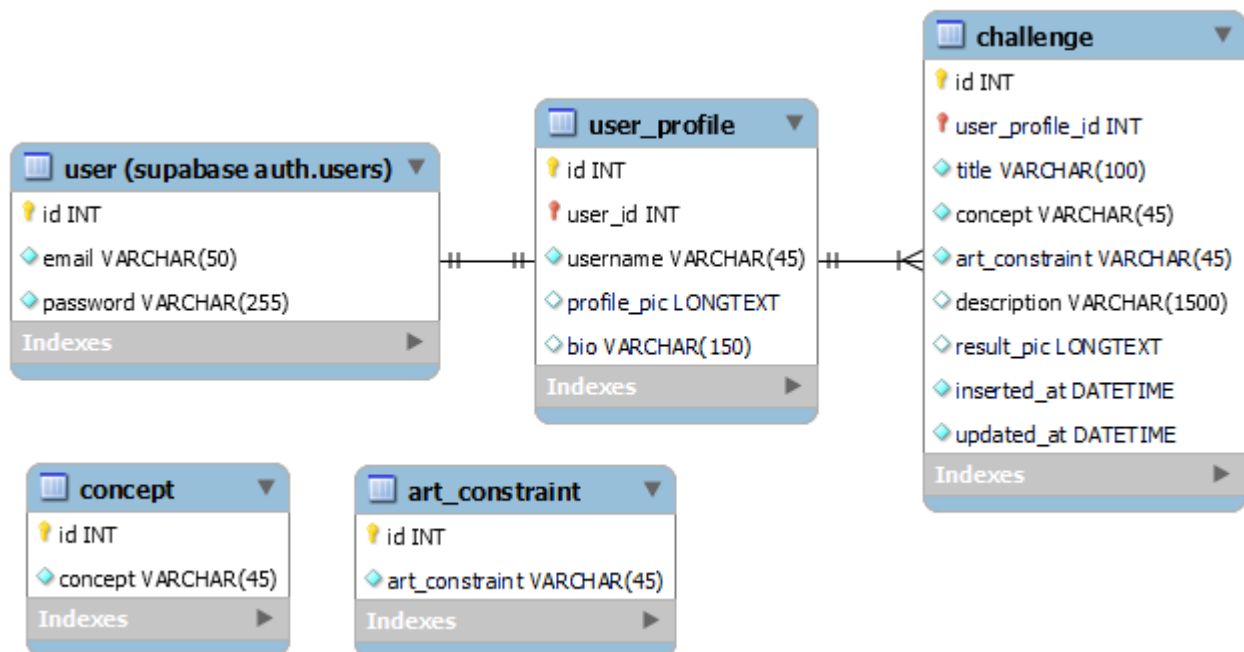
La progettazione del database è partita da una modellazione concettuale tramite diagramma E-R in MySQL Workbench, tradotta poi in una struttura fisica implementata in Supabase attraverso script SQL.

L'obiettivo principale è garantire semplicità, integrità referenziale e sicurezza, pur mantenendo un accesso efficiente alle informazioni, anche aggregate.

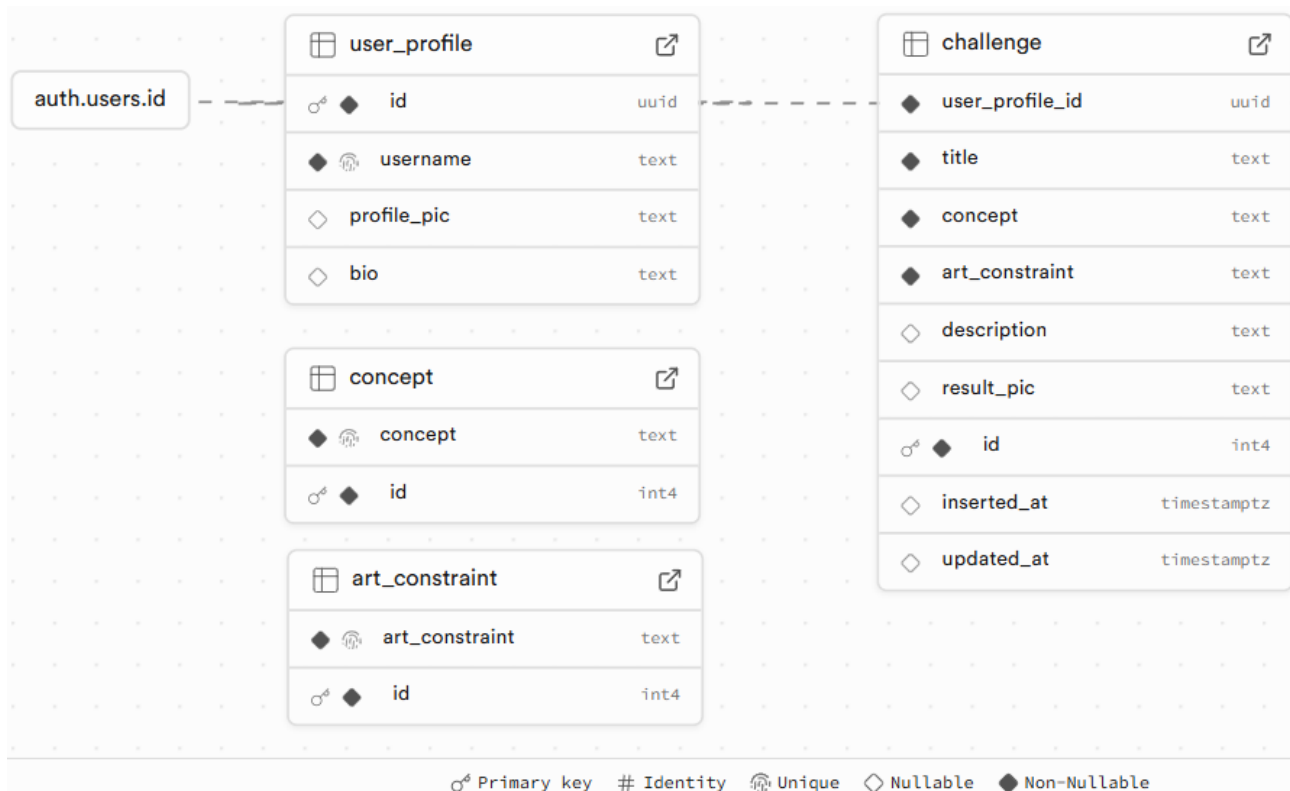
### 2.5.1. Modello relazionale

Il modello relazionale è stato progettato per rappresentare in maniera chiara la relazione tra utenti e le challenge da essi create. Le principali scelte progettuali includono:

- Una relazione 1:N (uno a molti) tra le tabelle `user_profile` e `challenge`;
- tabelle autonome per temi (`concept`) e vincoli (`art_constraint`) usati nelle generazioni casuali;
- uso di viste (`challenge_vw` e `user_profile_vw`) per aggregare informazioni complesse per la UI, come negli elenchi delle challenge e degli utenti.



## 2.5.2. Struttura delle tabelle in Supabase



### Tabella user\_profile

```
create table public.user_profile (  
  id uuid not null,  
  username text not null,  
  profile_pic text null,  
  bio text null,  
  constraint user_profile_pkey primary key (id),  
  constraint user_profile_username_key unique (username),  
  constraint user_profile_id_fkey foreign KEY (id)  
  references auth.users (id) on delete CASCADE  
) TABLESPACE pg_default;
```

Collega ogni utente autenticato, dalla tabella auth.users di supabase, al suo profilo personalizzato. Include:

- username univoco
- immagine del profilo e biografia (opzionali).



## Tabella challenge

```
create table public.challenge (  
  id serial not null,  
  user_profile_id uuid not null,  
  title text not null,  
  concept text not null,  
  art_constraint text not null,  
  description text null,  
  result_pic text null,  
  inserted_at timestamp with time zone null default now(),  
  updated_at timestamp with time zone null default now(),  
  constraint challenge_pkey primary key (id),  
  constraint challenge_user_profile_id_fkey foreign KEY (user_profile_id)  
  || || || || || references user_profile (id) on delete CASCADE  
) TABLESPACE pg_default;  
  
create trigger set_updated_at_challenge BEFORE  
update on challenge for EACH row  
execute FUNCTION update_updated_at_column ();
```

Contiene le sfide create dagli utenti, con i seguenti campi:

- title, concept, art\_constraint (obbligatori);
- description, result\_pic (facoltativi);
- user\_profile\_id: chiave esterna verso il profilo.

## Tabelle concept e art\_constraint

```
create table public.concept (  
  id serial not null,  
  concept text not null,  
  constraint subject_pkey primary key (id),  
  constraint subject_subject_key unique (concept)  
) TABLESPACE pg_default;  
  
create table public.art_constraint (  
  id serial not null,  
  art_constraint text not null,  
  constraint challenge_constraint_pkey primary key (id),  
  constraint challenge_constraint_challenge_constraint_key unique (art_constraint)  
) TABLESPACE pg_default;
```

Tabelle scollegate, usate per selezionare casualmente temi e vincoli artistici durante la creazione e la modifica delle challenge.

### 2.5.3. Viste per aggregazione dati

#### Vista challenge\_vw

```
create view challenge_vw as
select
  c.id,
  u.id as user_id,
  c.title,
  c.concept,
  c.art_constraint,
  c.description,
  c.result_pic,
  c.inserted_at,
  c.updated_at,
  u.username,
  u.profile_pic,
  u.bio
from challenge c
join user_profile u
  on c.user_profile_id = u.id
```

Unisce i dati della challenge con quelli dell'autore. Serve principalmente a popolare le cards degli elenchi delle challenge presenti nella Home e nei profili degli utenti.

#### Vista user\_profile\_vw

```
create view user_profile_vw as
select
  u.id,
  u.username,
  u.profile_pic,
  u.bio,
  count(c.id) as created_count,
  count(c.result_pic) filter (where c.result_pic is not null) as completed_count
from user_profile u
left join challenge c
  on c.user_profile_id = u.id
group by
  u.id,
  u.username,
  u.profile_pic,
  u.bio;
```

Unisce i dati dell'utente (username, bio, ecc.) ai valori calcolati del numero totale di challenge create e del numero di challenge completate (con result\_pic presente). Serve principalmente a popolare le cards dell'elenco di utenti presenti nella sezione Artists.



Queste due viste ottimizzano le query nella UI evitando di scrivere e ripetere direttamente le join nel codice.

#### 2.5.4. Autenticazione con Supabase Auth

Inkspire usa il provider email/password di Supabase. Gli utenti vengono registrati in `auth.users`, ma i dati visibili pubblicamente sono gestiti nella tabella `user_profile`, collegata tramite `id`. Questo consente di separare i dati sensibili da quelli pubblici e applicare policies diverse a seconda del contesto.

#### 2.5.5. Policies di sicurezza

Per garantire la sicurezza a livello riga (Row Level Security), sono state attivate su tutte le tabelle le **RLS policies** con le seguenti regole principali:

- **Lettura pubblica:** per `concept`, `art_constraint`, `user_profile` e `challenge` (solo lettura)
- **Scrittura protetta:** solo l'autore può creare, modificare o eliminare le proprie `challenge` e modificare lo `user_profile`. Tutte le policies usano la condizione `auth.uid() = user_profile_id` per garantire che ogni utente possa agire solo sui propri dati.

Esempio: policy di SELECT e DELETE challenge

```
alter policy "Authenticated can read all challenges"
on "public"."challenge"
to authenticated
using (true);

alter policy "Users can delete their own challenge"
on "public"."challenge"
to authenticated
using (user_profile_id = auth.uid());

alter policy "Users can insert their own challenge"
on "public"."challenge"
to authenticated
with check (user_profile_id = auth.uid());

alter policy "Users can update their own challenge"
on "public"."challenge"
to authenticated
using (user_profile_id = auth.uid())
with check (user_pr);
```

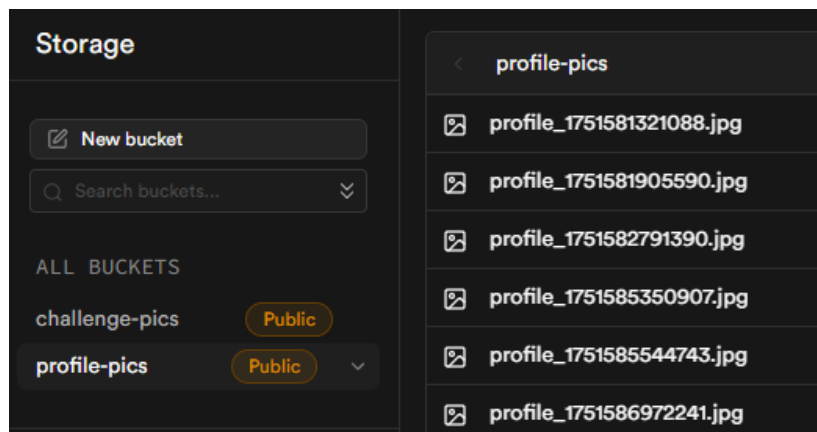
##### challenge

SELECT	<b>Authenticated can read all challenges</b> Applied to: authenticated role
DELETE	<b>Users can delete their own challenge</b> Applied to: authenticated role
INSERT	<b>Users can insert their own challenge</b> Applied to: authenticated role
UPDATE	<b>Users can update their own challenge</b> Applied to: authenticated role

## 2.5.6. Supabase Storage

Sono stati creati due bucket:

- **profile-pics**: immagini profilo utente
- **challenge-pics**: immagini delle challenge completate



Policies impostate:

- Upload: consentito solo agli utenti autenticati
- Lettura: disponibile agli utenti autenticati

Le immagini vengono salvate tramite Supabase Storage API e referenziate nel database via URL nei campi `profile_pic` o `result_pic`.

user_profile			
id	user...	profile_pic	
1222437d-ccd2-...	Morena	NULL	
1c2a526d-df48-...	supermario	https://wbwyqjhafxfkprjmvwfj.supabase.co/storage/v1/object/public/profile-pics/profile_1751933251242.jpg	
4265de6c-915d-...	Fedeee21	NULL	
5356e082-1e28-...	alecap	https://wbwyqjhafxfkprjmvwfj.supabase.co/storage/v1/object/public/profile-pics/profile_1751934869965.jpg	
baf18025-9796-...	test1	https://wbwyqjhafxfkprjmvwfj.supabase.co/storage/v1/object/public/profile-pics/profile_1751741815488.jpg	
c4b93f58-8326-...	teacap	https://wbwyqjhafxfkprjmvwfj.supabase.co/storage/v1/object/public/profile-pics/profile_1751741773082.jpg	
d2b848fa-bd26-...	robcap	https://wbwyqjhafxfkprjmvwfj.supabase.co/storage/v1/object/public/profile-pics/profile_1751587472956.jpg	
d8e3caea-f2ab-...	test2	https://wbwyqjhafxfkprjmvwfj.supabase.co/storage/v1/object/public/profile-pics/profile_1751586972241.jpg	
edf0d39d-0bf1-...	venafe	NULL	

## 2.6. Architettura

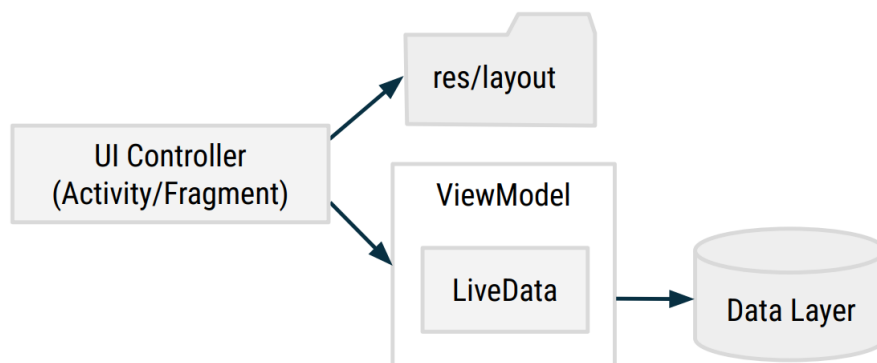
L'applicazione Inkspire è stata progettata secondo il principio della **Separation of Concerns**, ovvero la separazione delle responsabilità tra i vari componenti del software, al fine di garantire modularità, manutenibilità e facilità di test.

È stato adottato il pattern architetturale **MVVM** (Model–View–ViewModel), integrato con il **Repository Pattern** e l'utilizzo di **Supabase** come backend completo per autenticazione, database e storage. La comunicazione REST con il backend è gestita tramite il client **Ktor**.

### 2.6.1. MVVM Pattern

L'approccio MVVM prevede una chiara separazione tra:

- **Model:** classi dati che rappresentano le entità logiche dell'app. In questo progetto rispecchiano anche la struttura del database relazionale su Supabase;
- **View:** responsabile della visualizzazione dei dati e dell'interazione utente. Include i layout XML, gli Adapter, le Activity e i Fragment. Le View sono collegate ai rispettivi ViewModel tramite Data Binding e osservano lo stato tramite LiveData;
- **ViewModel:** mediatore tra View e Model. Espone dati osservabili (LiveData) e gestisce lo stato e la logica della UI in modo reattivo, indipendentemente dal ciclo di vita della View. I ViewModel invocano i metodi dei Repository ed eseguono operazioni asincrone tramite Kotlin Coroutine.



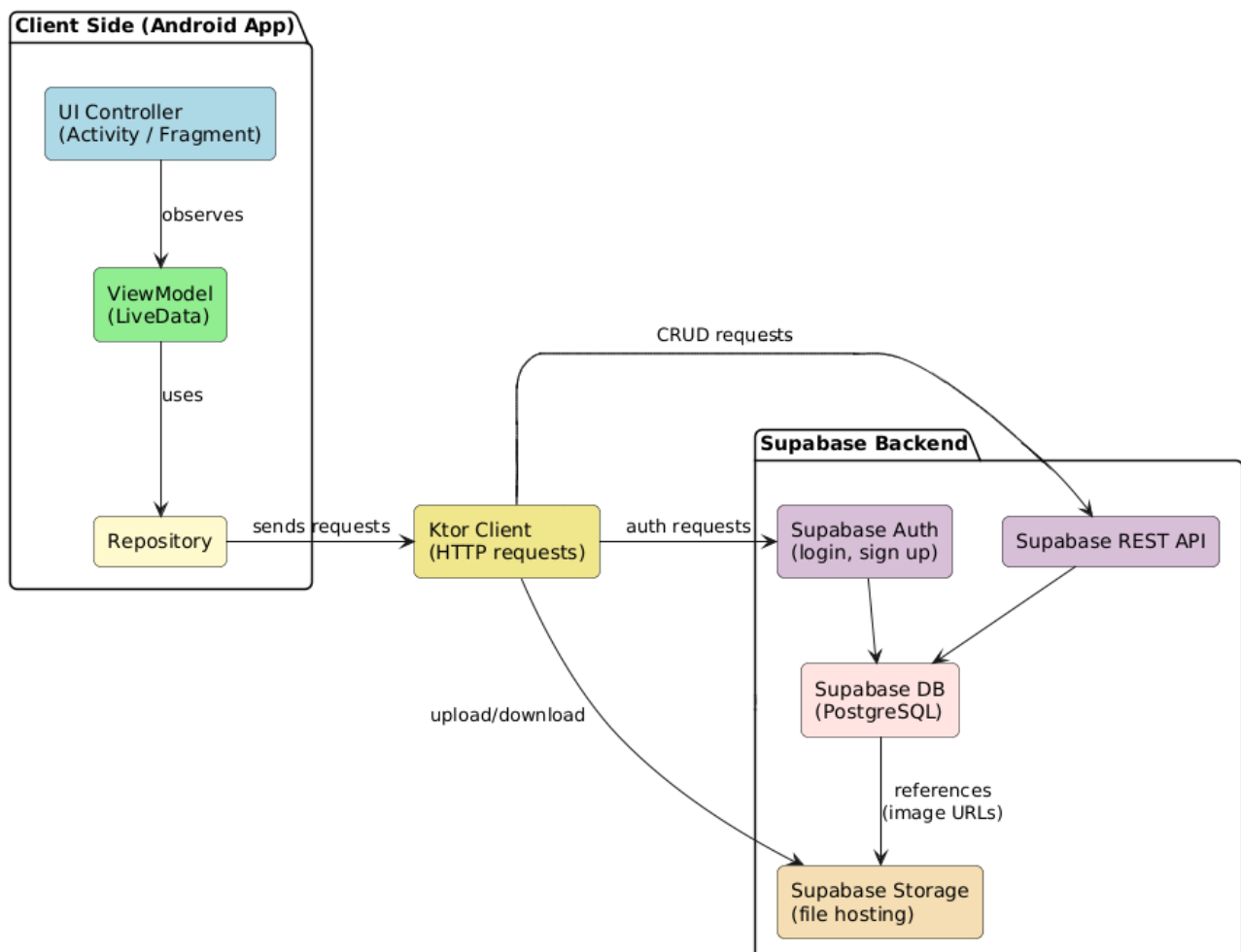
### 2.6.2. Repository Pattern

Il Repository è l'unico punto di accesso ai dati per il ViewModel. Ha il compito di astrarre l'origine dei dati (che può essere remota o locale) e incapsulare la logica di business necessaria per recuperarli o modificarli.

Nel caso di Inkspire, i Repository comunicano con il backend Supabase attraverso richieste HTTP gestite dalla libreria **Ktor**.

Questo approccio garantisce:

- maggiore testabilità dei singoli componenti;
- disaccoppiamento dalle librerie di rete o backend;
- facilità di estensione (es. aggiunta di cache o sorgenti dati alternative).



## 2.7. Sviluppo

### 2.7.1. Panoramica dello sviluppo

Il processo di sviluppo è partito dalla progettazione dei layout xml, che ha permesso di definire in anticipo la struttura visiva dell'app e fornire un riferimento concreto per l'implementazione successiva della logica.

Parallelamente sono stati definiti i Model, in modo da rispecchiare la struttura delle tabelle e delle viste definite sul database Supabase.

Dopo aver identificato le operazioni necessarie per interagire con i dati, sono stati sviluppati i Repository, contenenti le funzioni per la comunicazione col database, e a tal fine è stata utilizzata la libreria open-source Supabase Kotlin, che si appoggia su Ktor e offre un'interfaccia tipizzata per l'accesso ai servizi PostgREST, Auth e Storage di Supabase.

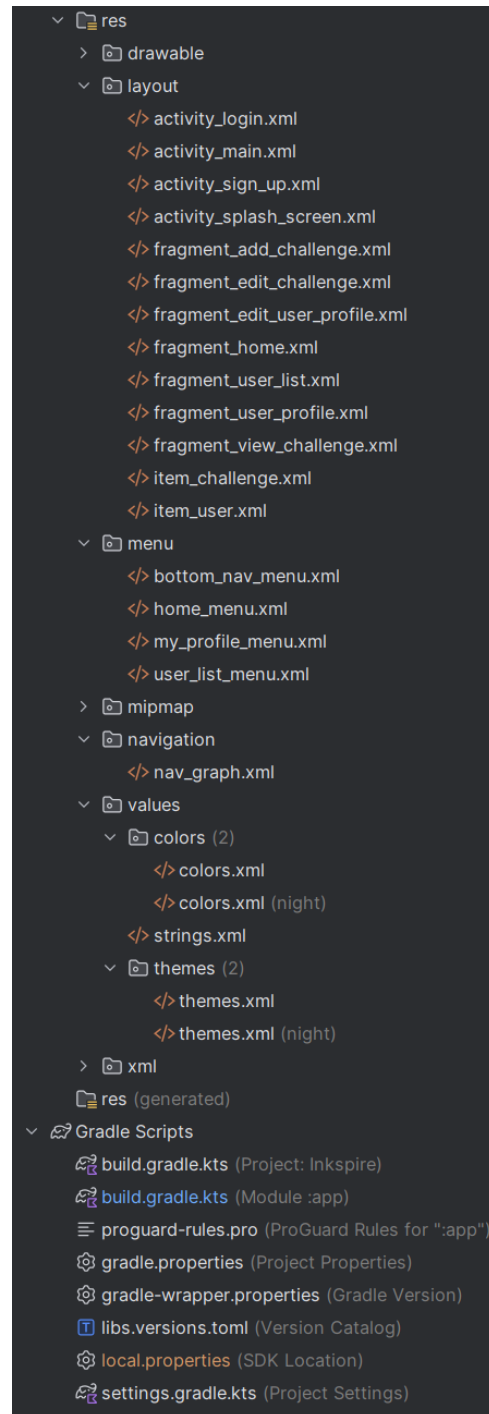
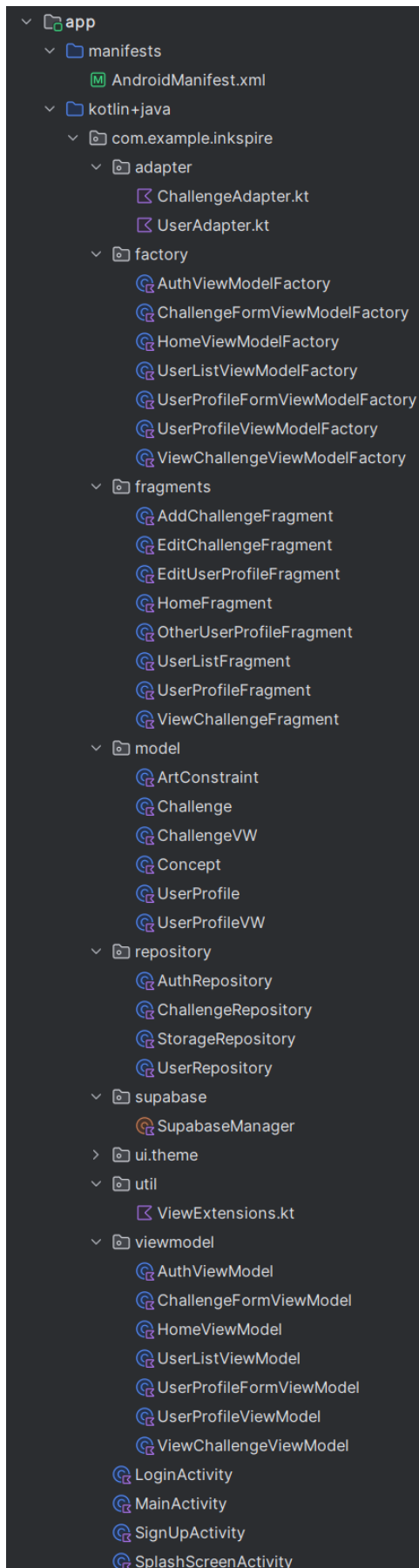
I ViewModel sono stati creati per ciascuna schermata che richiedesse logica o gestione dello stato, agendo da ponte tra repository e UI, per esporre in modo reattivo dati e operazioni verso i fragment.

Successivamente si è passati alla definizione degli Adapter per le liste dinamiche, all'implementazione dei Fragments e delle Activity, e infine all'integrazione del sistema di navigazione con Navigation Component e Safe Args.

L'autenticazione è stata una delle prime funzionalità implementate, e ha richiesto la configurazione iniziale di Supabase, il collegamento sicuro dell'app e la definizione dei relativi repository e viewmodel per la gestione della registrazione e dell'accesso utente. Il ciclo di sviluppo è poi proseguito con la gestione delle challenge, l'implementazione delle sezioni dedicate al profilo utente e alla visualizzazione della community, e infine la gestione delle immagini attraverso Supabase Storage e Glide.

Proteggono la privacy di un utente Android. Vengono dichiarati con il tag <uses-permission> nel file AndroidManifest.xml

## 2.7.2. Struttura del progetto



Il codice è stato suddiviso in file e package distinti, ognuno con una responsabilità ben definita:

- **AndroidManifest.xml**: file xml che descrive i componenti principali dell'app al sistema Android prima che venga eseguita. Contiene:
  - permessi Android di connessione e interazione con storage esterno, dichiarati con il tag `<uses-permission>`;
  - dichiarazioni delle activity, specificando quali sono esportabili (avviabili direttamente dall'esterno) e quali interne, e quale tra queste avvia l'app (launcher);
  - configurazione e impostazioni globali app (nome, icona, tema, targetApi, ecc.)
- **build.gradle**: file di configurazione del progetto, in cui è stata disattivata la libreria Jetpack Compose in favore dei layout xml classici. Contiene:
  - dichiarazioni dei plugin utilizzati (kotlin-parcelize, serialization, navigation.safeargs);
  - caricamento delle chiavi segrete SUPABASE\_URL e SUPABASE\_KEY dal file .gitignore local.properties, evitando di esporle nel codice pubblico;
  - configurazione del modulo Android (namespace, compileSdk, ecc.) con l'abilitazione di dataBinding e viewBinding;
  - dipendenze utilizzate (ui classica, supabase, ktor, glide, navigation).
- **res**: sezione contenente i package con i file xml relativi ai layout delle schermate (layout), i temi (values/themes.xml) con i colori e gli stili, le immagini (drawable) e i menu di navigazione (menu);
- **navigation**: package contenente il file nav\_graph.xml, definito con Android Navigation Component, che rappresenta la mappa di navigazione tra i vari fragment dell'app e gestisce anche il passaggio dei parametri in modo sicuro tramite Safe Args;
- **model**: package contenente le classi dati (data class) che rappresentano le entità dell'app. Ogni model rispecchia fedelmente la struttura delle tabelle e delle viste definite su Supabase e alcuni includono annotazioni utili per la serializzazione con Kotlinx Serialization, in quanto Supabase utilizza JSON come formato di comunicazione;
- **repository**: package contenente le classi che fungono da intermediari tra i viewmodel e il backend Supabase. Le funzioni all'interno dei repository sono

implementate utilizzando le API fornite dalla libreria postgrest-kt di Supabase, e permettono operazioni CRUD sulle challenge, autenticazione, gestione profili e accesso ai dati delle viste;

- **viewmodel**: package contenente i ViewModel associati ai diversi fragment. Ogni viewmodel incapsula lo stato e la logica della relativa schermata e interagisce esclusivamente con il repository per recuperare e modificare i dati. Sono utilizzati LiveData per gestire lo stato in modo reattivo;
- **factory**: package contenente le classi ViewModelFactory necessarie per l'inizializzazione dei ViewModel con parametri personalizzati;
- **adapter**: package contenente gli Adapter utilizzati per le RecyclerView, in particolare per visualizzare l'elenco delle challenge e l'elenco degli utenti. In particolare gli adapter gestiscono anche il binding dei dati ai layout delle singole card;
- **manager**: package contenente una singola classe SupabaseManager, che centralizza la configurazione e l'accesso all'istanza di Supabase. Questo file inizializza i moduli auth, postgrest e storage, rendendoli disponibili a tutta l'applicazione tramite singleton;
- **utils**: raccoglie estensioni riutilizzabili per migliorare l'interattività dei componenti UI (EditText, View) e la loro integrazione con i LiveData.

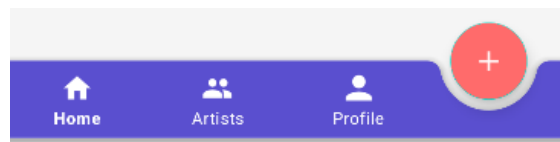


### 2.7.3. Layout e UI

L'interfaccia di Inkspire è stata progettata con l'obiettivo di garantire semplicità, coerenza visiva e un'esperienza utente intuitiva. Tutti i layout sono stati implementati tramite file xml e resi dinamici attraverso il collegamento con i ViewModel e i LiveData, sfruttando il binding dei dati e gli observer per aggiornare la UI in tempo reale.

#### Layout di Activity e Fragment

- **activity\_splash\_screen.xml**: layout per lo splash screen iniziale di ingresso all'app, contiene solo una ImageView dell'icona di Inkspire e una ProgressBar;
- **activity\_main.xml**: layout principale con NavHostFragment per la gestione della navigazione, una BottomAppBar con BottomNavigationView per la navigazione tra schermate principali (Home, Artists, Profile) e un FloatingActionButton centrale per l'aggiunta di una nuova challenge;



- **fragment\_login.xml** e **fragment\_signup.xml**: schermate di autenticazione con i relativi campi.
- **fragment\_home.xml**: contiene una RecyclerView per visualizzare le challenge e un'ImageView (empty\_bg) da mostrare quando l'elenco è vuoto;
- **fragment\_add\_challenge.xml** e **fragment\_edit\_challenge.xml**: form di creazione e modifica di una challenge, con campi EditText, di cui quelli del concept e art\_constraint dotati di drawable per lo shuffle, ImageView per l'immagine e Button per il salvataggio/eliminazione;

- **fragment\_view\_challenge.xml**: mostra i dettagli di una challenge, inclusi titolo, concetto, vincolo artistico, descrizione e immagine. Nel caso la challenge sia dell'utente corrente non c'è il link al profilo dell'autore ed è invece presente il pulsante per passare alla schermata di modifica;

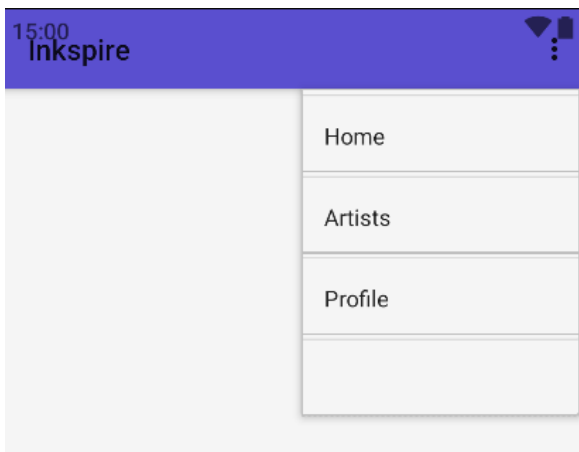
- **fragment\_user\_profile.xml** e **fragment\_edit\_profile.xml**: rispettivamente per la visualizzazione e la modifica del profilo utente. Il file di layout è lo stesso per il profilo dell'utente corrente e per quello di un altro utente, con la differenza che nel secondo caso non è presente il pulsante per passare alla schermata di modifica del profilo;
- **fragment\_user\_list.xml**: contiene una RecyclerView per visualizzare tutti gli utenti e un'ImageView (empty\_bg) da mostrare quando l'elenco è vuoto.

## Layout degli items

- **Item\_challenge.xml** e **item\_user.xml**: layout per le CardView di challenge e utente da visualizzare nella RecyclerView.



## Menu



Il menu principale è gestito tramite il file **bottom\_nav\_menu.xml**, che definisce le voci di navigazione visualizzate nella BottomNavigationView (sezione vuota lasciata per il FAB).

Sono presenti anche altri menu differenziati in base al fragment: **home\_menu** e **fragment\_user\_list** con l'icona per la ricerca e **my\_profile\_menu** con l'icona di logout.

## Gestione temi e colori

L'app supporta tema chiaro e tema scuro, definiti nel file **themes.xml** e **themes-night.xml**. I colori principali sono personalizzati e centralizzati nel file **colors.xml**.

## Risorse grafiche

Le immagini statiche, le icone personalizzate e gli sfondi sono contenuti nella cartella **res/drawable**. Vengono per lo più utilizzate icone vettoriali per garantire una buona scalabilità e qualità visiva su tutti i dispositivi.

## 2.7.4. Autenticazione, Storage e accesso al DB

L'intero back-end di Inkspire è ospitato su Supabase, scelto perché combina:

- un **database PostgreSQL** gestito (accessibile via PostgREST);
- un modulo **Auth** pronto all'uso;
- uno **Storage** per file multimediali;
- **SDK Kotlin** (supabase-kt): una libreria specifica che usa internamente Ktor per comunicare con il database, offrendo un'interfaccia più comoda e tipizzata.

### Auth: Registrazione e login e-mail/password

Per l'autenticazione viene utilizzata la tabella di sistema auth.users gestita da Supabase. Il flusso di tale processo è il seguente:

1. **Sign-up:** l'AuthRepository controlla che lo username sia libero (user\_profile), poi invoca auth.signUpWith(Email)
2. **Login/Logout:** tramite auth.signInWith(Email) e auth.signOut()
3. **Sessione:** il client è configurato con `autoLoadFromStorage = true` e `alwaysAutoRefresh = true`, per cui il token JWT viene rinnovato in background e l'utente resta loggato tra un avvio e l'altro.

Il codice chiave risiede nel SupabaseManager, dove il modulo Auth viene installato contestualmente alla creazione del client:

```
object SupabaseManager {  
  
    val client: SupabaseClient by lazy {  
        createSupabaseClient(  
            supabaseUrl = BuildConfig.SUPABASE_URL,  
            supabaseKey = BuildConfig.SUPABASE_ANON_KEY  
        ) {  
            install(Auth) {  
                autoLoadFromStorage = true  
                alwaysAutoRefresh = true  
            }  
            install(Postgrest)  
            install(Storage)  
        }  
    }  
}
```

**SupabaseManager.kt:** classe singleton (object) per logiche di servizio. A differenza dei ViewModel che sono legati all'UI e al suo ciclo di vita (LiveData), i Manager sono classi usate solo per incapsulare una certa logica (funzioni di basso livello, accesso API,

ecc.) che vengono istanziate una sola volta e sono accessibili globalmente. SupabaseManager contiene la funzione per inizializzare il client Supabase, e viene chiamato all'avvio dell'app:

- lazy: crea pigramente un'istanza del client, cioè un'istanza che si inizializza solo al primo accesso e non subito all'avvio dell'app se non serve immediatamente;
- install(Auth): installa il modulo di auth supabase e tenta di recuperare token/sessione salvata in locale;
- install(Postgrest): installa il modulo per interagire col database Supabase (PostgreSQL) tramite il client REST PostgREST.

## Storage: Upload delle immagini

Per le immagini dei risultati delle challenge e delle foto profilo degli utenti vengono creati i due buckets pubblici challenge-pics e profile-pics con le relative policy: gli utenti autenticati possono caricare (upload) e leggere (download) i file nel proprio bucket.

L'utilizzo dell'SDK è riportato nella funzione uploadImage() dello StorageRepository:

```
return try {
    SupabaseManager.storage.from(bucket)
        .upload(
            path = filePath,
            data = byteArray
        ) {
            upsert = true
            this.contentType = contentType
        }

    val publicUrl = "https://{SupabaseManager.client.supabaseUrl}/storage/v1/object/public/$bucket/$filePath"
```

Nella quale terminato l'upload viene costruito manualmente l'URL pubblico che viene poi salvato nel database e caricato nell'app con Glide.

## Accesso al database con la libreria Supabase-kt

Le operazioni sul database avvengono tramite chiamate REST generate dall'SDK, che serializza/deserializza JSON in oggetti Kotlin:

```
supabase.from("challenge")
    .update(updateData) { filter { eq("id", challenge.id) }
    select()
}
.decodeSingle<Challenge>()
```

### 2.7.5. Model

Tutti i Model dell'app sono stati definiti utilizzando classi dati di Kotlin annotate con **@Serializable** per consentire la serializzazione automatica da e verso JSON, necessaria per la comunicazione con il backend Supabase tramite la libreria supabase-kt. L'annotazione **@Parcelize** è stata aggiunta ai model relativi alle challenge e agli utenti per permettere il passaggio sicuro e diretto degli oggetti tra fragment tramite SafeArgs.

L'approccio seguito è stato quello di mantenere una **corrispondenza 1:1** tra i model e le tabelle (o viste) del database Supabase, così da semplificare il mapping automatico con le risposte JSON ottenute via PostgREST.

I model realizzati sono:

- **Challenge**: rappresenta una sfida creata da un utente, associata tramite foreign key al suo profilo;
- **UserProfile**: contiene le informazioni pubbliche del profilo utente (username, bio, immagine);
- **ChallengeVW**: model mappato sulla view challenge\_vw creata in Supabase per ottenere i dati aggregati tra challenge e autore, utilizzato per popolare le recycler view;
- **UserProfileVW**: model mappato sulla view user\_profile\_vw, utile per ottenere statistiche sull'attività dell'utente (challenge create e completate) da mostrare nel suo profilo e nella recycler view degli utenti;
- **Concept** e **ArtConstraint**: modelli per la generazione casuale delle challenge, corrispondenti a due tabelle distinte e indipendenti.

### 2.7.6. Repository

I repository rappresentano il livello di accesso ai dati dell'applicazione, centralizzando le operazioni di comunicazione tra l'interfaccia utente e il backend Supabase. Tutti i repository sono scritti come classi Kotlin che sfruttano le **coroutine** e la libreria **supabase-kt** per eseguire richieste asincrone in formato JSON tramite PostgREST.

Ogni repository si occupa di un insieme logico di operazioni. Oltre all'AuthRepository e StorageRepository visti in precedenza, i due repository principali dell'app sono:

- **ChallengeRepository.kt**: gestisce tutte le operazioni CRUD sulle challenge: inserimento, aggiornamento, cancellazione e ricerca. Utilizza metodi come `decodeSingle<T>()` o `decodeList<T>()` per convertire automaticamente le risposte JSON in oggetti Kotlin, e restituisce Boolean o liste di oggetti a seconda del contesto. Come esempio si riportano le funzioni:

```
suspend fun insertChallenge(challenge: Challenge): Boolean {
    return try {
        supabase.from("challenge")
            .insert(challenge) {
                select()
            }
            .decodeSingle<Challenge>()
        true
    } catch (e: Exception) {
        false
    }
}
```

`insertChallenge()` inserisce un oggetto Challenge nel database e ritorna true in caso di successo.

```
suspend fun getAllChallenges(): List<ChallengeVW> {
    return supabase.from("challenge_vw")
        .select {
            order("updated_at", Order.DESENDING)
        }
        .decodeList<ChallengeVW>()
}
```

`getAllChallenges()` restituisce l'elenco completo delle challenge aggregate (tramite la view `challenge_vw`), ordinate per data di aggiornamento.

- **UserRepository.kt**: gestisce il recupero e l'aggiornamento dei profili utente, l'accesso ai dati aggregati tramite view (come `user_profile_vw`) e il calcolo delle

statistiche individuali (challenge totali e completate). Anche in questo caso è stato adottato un approccio robusto con try/catch per la gestione delle eccezioni e l'utilizzo di mappe (Map<String, Any?>) per aggiornare solo i campi modificabili (come bio e immagine):

```
suspend fun updateUserProfile(userProfile: UserProfile): Boolean {
    return try {
        val updateData = mapOf(
            "bio" to userProfile.bio,
            "profile_pic" to userProfile.profile_pic
        )
        client.from("user_profile")
            .update(updateData) {
                filter { eq("id", userProfile.id) }
                select()
            }
            .decodeSingleOrNull<UserProfile>() != null
    } catch (e: Exception) {
        e.printStackTrace()
        false
    }
}
```

Tutte le query sono eseguite utilizzando una sintassi fluida, definita tramite lambda per impostare i filtri o l'ordinamento.

### 2.7.7. ViewModel

Per ciascuna schermata con logica associata è stato definito un ViewModel dedicato. I ViewModel fungono da intermediari tra la UI e i repository, mantenendo separata la logica di business dalla logica di presentazione. La comunicazione tra ViewModel e Fragment avviene in modo unidirezionale, principalmente tramite LiveData, osservata all'interno della UI.

Ogni ViewModel espone:

- Dati di dominio (es. elenco challenge, profili utente, dettagli)
- Eventuali messaggi o segnali di stato (es. esito delle operazioni, caricamento in corso)
- Funzioni per attivare operazioni asincrone (es. login, inserimento, aggiornamento, eliminazione)

I ViewModel definiti sono i seguenti:

- **HomeViewModel**: gestisce il recupero e la ricerca delle challenge visibili nella Home;
- **ChallengeFormViewModel**: fornisce tutte le operazioni CRUD su una challenge, gestisce l'upload dell'immagine su Supabase Storage e suggerisce parametri random per tema e vincolo artistico;
- **UserProfileViewModel** e **UserProfileFormViewModel**: si occupano del profilo utente, rispettivamente per la visualizzazione e la modifica dei dati.
- **ViewChallengeViewModel**: carica i dati di una challenge verificando anche se l'utente corrente è l'autore della challenge selezionata, in modo da far adattare l'interfaccia di conseguenza;
- **AuthViewModel**: gestisce la logica di autenticazione e la propagazione di messaggi di feedback;
- **UserListViewModel**: mostra gli utenti della community e filtra in base a ricerche testuali.

In presenza di dipendenze da repository, i ViewModel sono stati istanziati tramite **Factory** personalizzate, al fine di supportare parametri nel costruttore. Tutte le operazioni asincrone sono gestite con `viewModelScope.launch` per evitare memory leak e rispettare il ciclo di vita dell'interfaccia.



### 2.7.8. Adapter

Per la visualizzazione delle liste dinamiche in RecyclerView, sono stati definiti due adapter principali che estendono **ListAdapter**, sfruttando `DiffUtil.ItemCallback` per ottimizzare il rendering e ridurre i ricaricamenti superflui:

- **ChallengeAdapter**: utilizzato per mostrare l'elenco delle challenge pubblicate, sia nella home che nei profili utente. Ogni elemento della lista è rappresentato dal layout `item_challenge.xml`, che mostra il titolo, il concept e l'autore della challenge, insieme all'immagine del risultato (se disponibile). Le immagini vengono caricate asincronamente tramite Glide, con supporto a placeholder e progress bar.
- **UserAdapter**: utilizzato per visualizzare la lista degli utenti. Ogni elemento (`item_user.xml`) mostra il nome utente, le statistiche delle challenge create/completate e l'immagine del profilo, anch'essa caricata tramite Glide con effetto `circleCrop()`.

Entrambi gli adapter espongono una **lambda onClick** per gestire gli eventi di selezione dell'elemento, facilitando la navigazione verso i dettagli o i profili utente.

L'organizzazione degli adapter segue il **pattern ViewHolder** interno e utilizza il binding automatico delle view tramite classi generate (`ItemChallengeBinding`, `ItemUserBinding`), favorendo la leggibilità e riduzione di codice ridondante.

```
class ChallengeAdapter(
    private val onChallengeClick: (ChallengeVW) -> Unit
) : ListAdapter<ChallengeVW, ChallengeAdapter.ChallengeViewHolder>(ChallengeDiffCallback()) {

    inner class ChallengeViewHolder(private val binding: ItemChallengeBinding) :
        RecyclerView.ViewHolder(binding.root) {

        fun bind(challengeUser: ChallengeVW) {...}

    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ChallengeViewHolder {...}
    override fun onBindViewHolder(holder: ChallengeViewHolder, position: Int) {...}
}

class ChallengeDiffCallback : DiffUtil.ItemCallback<ChallengeVW>() {
    override fun areItemsTheSame(oldItem: ChallengeVW, newItem: ChallengeVW) =
        oldItem.id == newItem.id

    override fun areContentsTheSame(oldItem: ChallengeVW, newItem: ChallengeVW) =
        oldItem == newItem
}
```

### 2.7.9. Activity e Fragment

L'architettura dell'applicazione Inkspire è basata su una singola **MainActivity** che ospita tutti i fragment dell'applicazione tramite un **NavHostFragment**. A questa struttura si affiancano due activity distinte dedicate all'autenticazione (**LoginActivity** e **SignUpActivity**) e una **SplashScreenActivity** che viene visualizzata all'avvio per verificare lo stato di login dell'utente.

La **MainActivity** funge da **contenitore** per tutti i fragment dell'interfaccia principale, integrando una **BottomNavigationView**, una **BottomAppBar** e un **FloatingActionButton**, la cui visibilità viene gestita dinamicamente in base alla destinazione attiva. I principali fragment dell'app si suddividono in due macro-categorie:

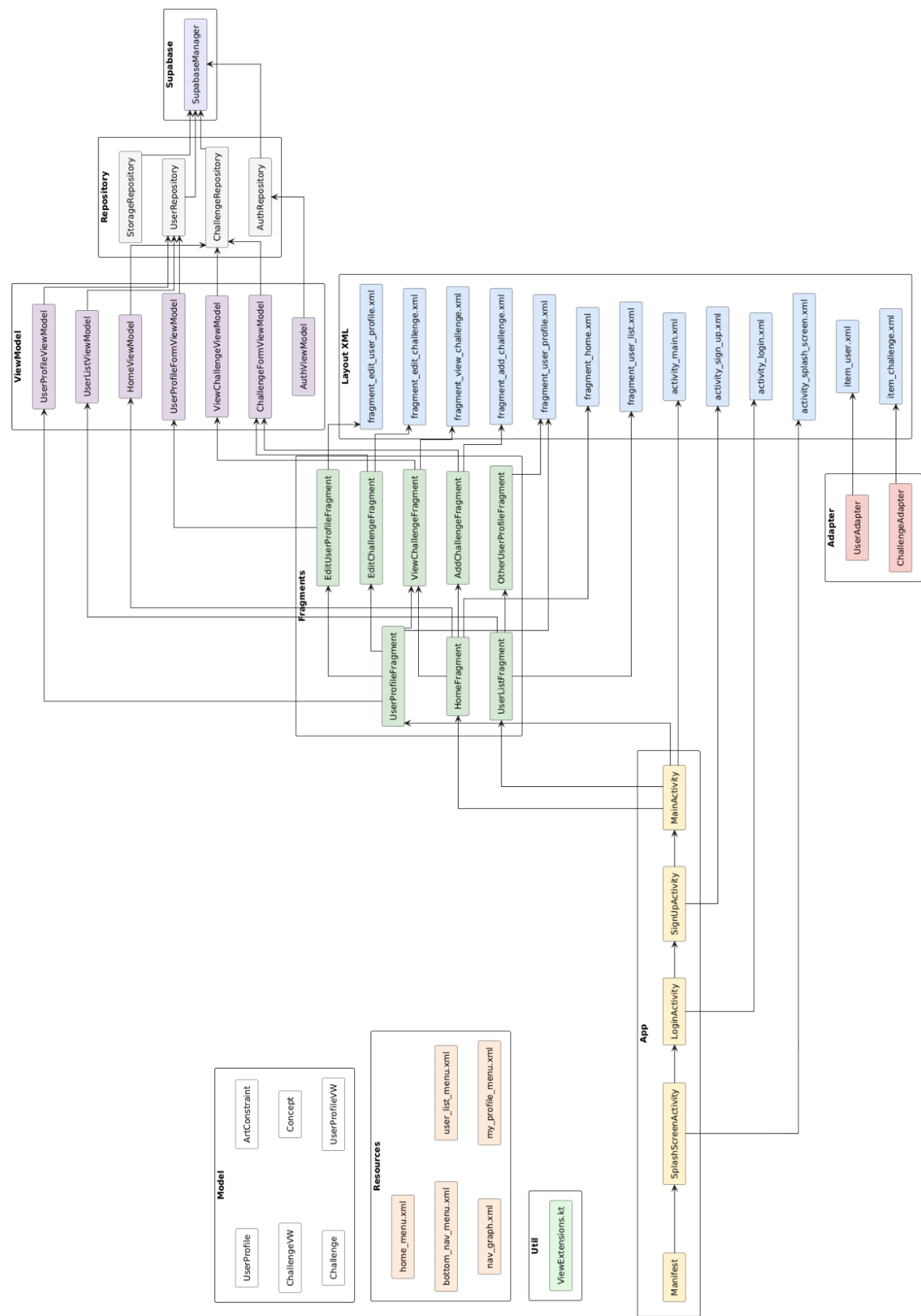
#### Area challenge e community

- **HomeFragment**: mostra l'elenco delle challenge pubblicate, tramite una **RecyclerView** alimentata da **HomeViewModel**. Supporta la ricerca tramite campo testuale e osserva lo stato di loading per aggiornare l'interfaccia;
- **AddChallengeFragment**: consente la creazione di una nuova challenge, con caricamento opzionale di un'immagine e suggerimenti random per il concept e l'art constraint;
- **EditChallengeFragment**: consente la modifica o eliminazione di una challenge esistente. I dati della challenge vengono caricati tramite ID e gestiti dal **ChallengeFormViewModel**;
- **ViewChallengeFragment**: visualizza i dettagli di una singola challenge. Se l'utente corrente è l'autore, viene mostrata l'opzione per modificare il contenuto;
- **UserListFragment**: elenca tutti gli utenti registrati, ordinati per numero di challenge completate, con possibilità di ricerca. Le informazioni sono fornite dallo **UserListViewModel**.

#### Area profilo utente

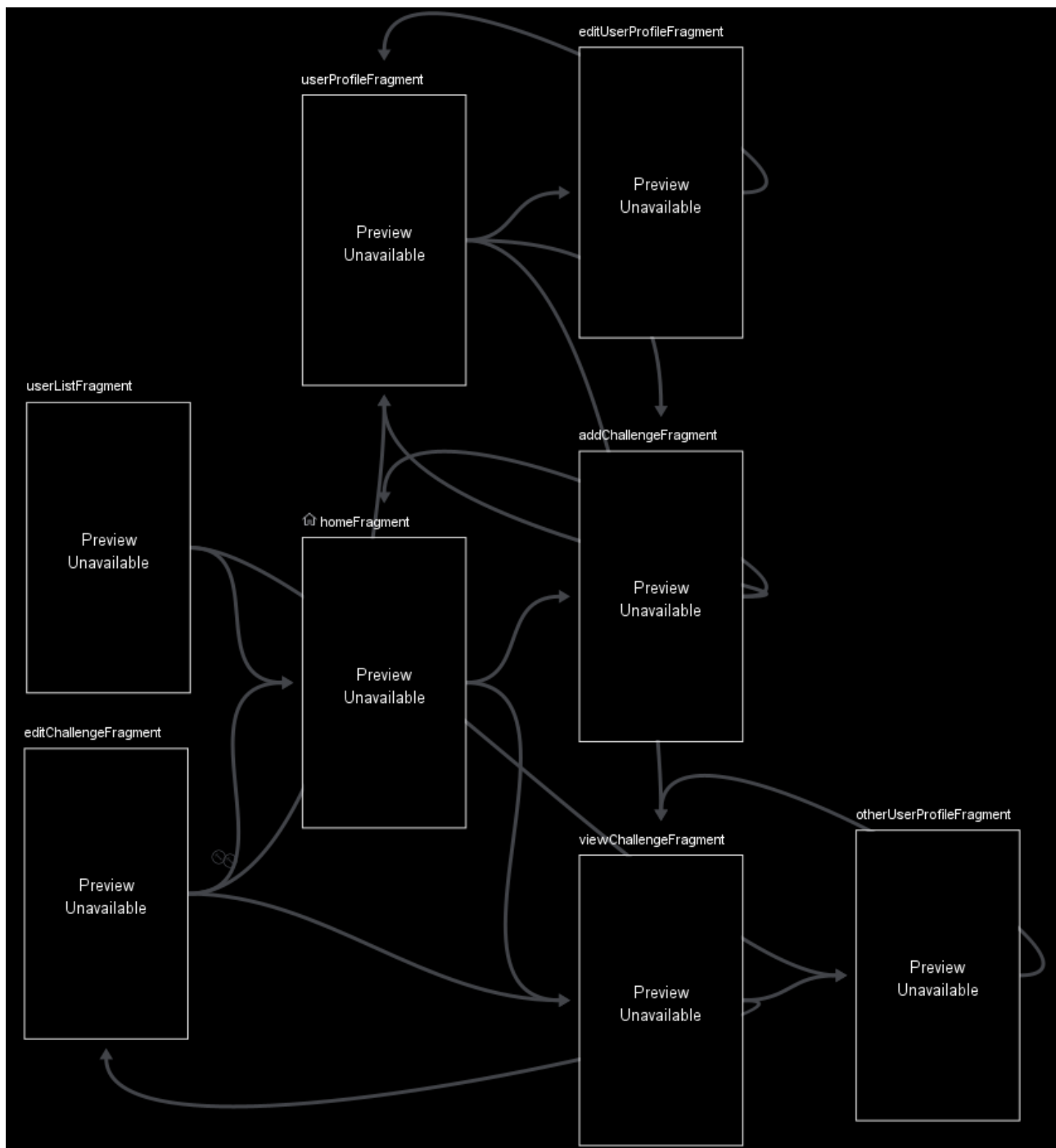
- **UserProfileFragment**: mostra i dati del profilo corrente. È possibile accedere alla modifica del profilo o effettuare il logout tramite i pulsanti dedicati;
- **EditUserProfileFragment**: permette all'utente di aggiornare bio e immagine del profilo. I dati sono gestiti dallo **UserProfileFormViewModel**;
- **OtherUserProfileFragment**: mostra il profilo di un altro utente in sola lettura, raggiungibile dai dettagli di una challenge.

Di seguito lo schema dei collegamenti tra tutti i file del progetto:



### 2.7.10. Navigazione

La navigazione tra i fragment è gestita dal **Navigation Component**, con grafo di navigazione definito nel file `nav_graph.xml`. I parametri vengono passati in modo type-safe tramite **SafeArgs**, assicurando coerenza tra fragment. La gestione del back stack e della navigation bar è centralizzata in MainActivity, in base alla destinazione corrente.



## 2.8. Problematiche e considerazioni finali

Durante lo sviluppo dell'applicazione Inkspire sono emerse alcune problematiche tecniche che hanno richiesto interventi puntuali per garantire la stabilità e il corretto funzionamento del sistema.

In fase iniziale si sono riscontrati conflitti tra le librerie di Jetpack Compose e quelle **AppCompat** utilizzate per le activity e i fragment. Per evitare incompatibilità, è stato deciso di rimuovere completamente i riferimenti a Compose dai file Gradle, mantenendo un approccio tradizionale basato su xml per la gestione dei layout.

Un ulteriore ostacolo ha riguardato le **operazioni CRUD** sul database Supabase: in assenza di policy esplicite, le query venivano bloccate per mancanza di permessi. Il problema è stato risolto configurando correttamente le Row Level Security (RLS) e le relative policy per ogni tabella coinvolta.

Per quanto riguarda la **navigazione**, si è optato per un'impostazione che ruota attorno ai fragment principali (Home, Artists, Profile), così da ridurre problematiche legate allo stack di navigazione. Le operazioni come la creazione o modifica di una challenge terminano riportando l'utente alla schermata principale, evitando stacking eccessivo di fragment secondari.

Durante il debugging è stato fatto ampio uso di **Log**, in particolare per verificare il contenuto delle richieste inviate al backend, come ad esempio i payload JSON nella fase di aggiornamento delle challenge.

L'app è stata condivisa con alcuni utenti esterni, che hanno effettuato test reali registrandosi e utilizzando le principali funzionalità. Il feedback ricevuto ha contribuito a identificare ulteriori aree di miglioramento.

Tra i possibili sviluppi futuri si segnalano:

1. Ottimizzazione della navigazione per evitare accumulo di fragment nello stack.
2. Gestione della dimensione delle immagini caricate: attualmente non sono imposti limiti, il che può rallentare il caricamento (soprattutto per immagini pesanti). Potrebbero essere integrate funzioni di **ridimensionamento** o compressione lato client.
3. Possibilità di **rendere private** le proprie challenge, visibili solo all'autore.
4. Aggiunta della funzionalità di **like** per aumentare l'interazione tra utenti e valorizzare i contenuti creati.

5. Refactoring del codice per ridurre la ridondanza, ad esempio introducendo fragment base condivisi tra schermate simili.

Nonostante le criticità iniziali, si conferma che l'applicazione è **funzionante e stabile**. Tutte le funzionalità descritte nell'analisi dei requisiti sono state implementate correttamente, garantendo all'utente la possibilità di registrarsi, creare e completare challenge artistiche, gestire il proprio profilo, visualizzare i contenuti della community e navigare fluidamente tra le sezioni principali dell'app.

## Bibliografia

Di seguito viene riportato tutto il materiale utilizzato, con documentazione, risorse e tutorial utili per lo sviluppo:

App structure (CRUD, MVVM, Activity e Fragment, Navigazione)

- The Notes App (MVVM + ROOM Database, CRUD operations)  
<https://www.youtube.com/watch?v=zCDB-OqOzfY>
- Types of Login and Signup in Android Studio (Auth + db)  
[https://www.youtube.com/watch?v=fStYN\\_\\_AmEc](https://www.youtube.com/watch?v=fStYN__AmEc)
- Navigation Component  
<https://developer.android.com/guide/navigation>

Ktor e Supabase

- Supabase Project creation and implementation on Android Studio  
[https://www.youtube.com/watch?v=\\_iXUVJ6HTHU](https://www.youtube.com/watch?v=_iXUVJ6HTHU)
- Supabase Auth with Email and Google + Login UI + Credentials Manager  
<https://www.youtube.com/watch?v=ZgYvexniGDA>
- Supabase plugins and dependencies for Android Studio  
<https://supabase.com/docs/reference/kotlin/installing>
- Ktor for API Services  
<https://ktor.io/docs/client-engines.html>
- Android Internet Connection (URL + public key)  
<https://developer.android.com/develop/connectivity/network-ops/connecting>
- Supabase Auth e GoTrue Auth Service  
<https://supabase.com/docs/guides/auth/architecture>

Editor

- Editor online per i diagrammi  
<https://www.plantuml.com/>