TaskFlow

Gruppo Appventurers

Federico Colciago 858643 – Marco Brambilla 856428 – Lorenzo Mauro 869782

Architettura

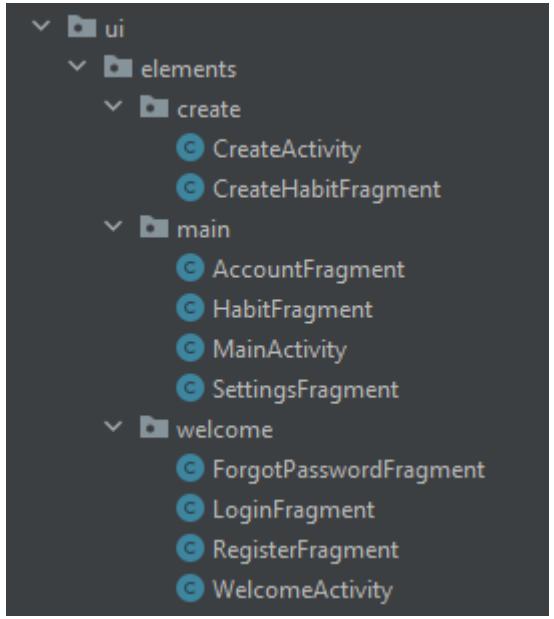
L'architettura utilizzata è la **MVVM (Model-View-ViewModel),** con LiveData e Navigation Component, è composta principalmente da tre strati :

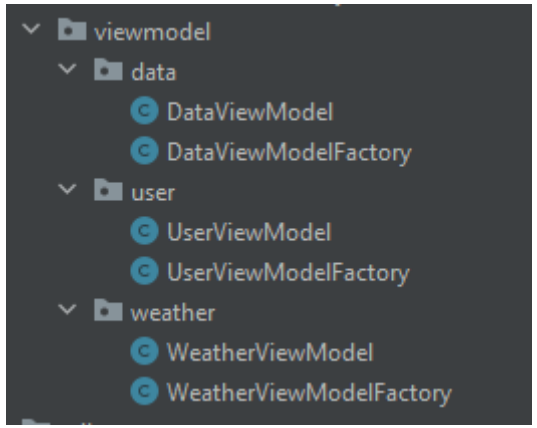
- **Activity – Fragment** : questa parte gestisce la UI, popola le view e tramite il NavHostGraph e il NavController associato gestisce la navigazione tra i vari Fragment.

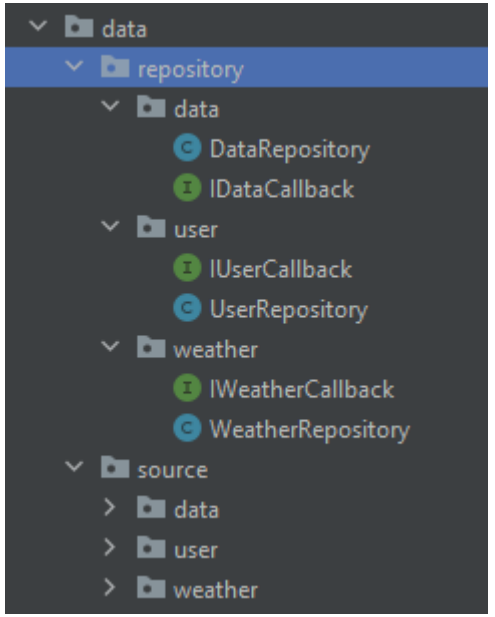
- **ViewModel** : è la parte intermedia tra la UI e il Repository, gestisce i dati, sotto forma di LiveData o MutableLiveData, che devono essere visualizzati. Ne abbiamo definiti 3:
  - DataViewModel : si occupa delle operazione delle abitudini, come il salvataggio, l'aggiornamento e l'ottenimento di tutte le abitudine salvate
  - UserViewModel : si occupa di tutte le operazioni relative all'utente, come la cancellazione dell'account, il login, il logout, ecc…
  - WeatherViewModel : per le operazione relative alla sezione del meteo

- **Repository** : la parte dei repository interagisce con i ViewModel e contiene la definizione di tutti i metodi per la gestione dei dati. Anche qui abbiamo definito 3 Repository principali, una per i dati, uno per l'utente e un'altra per il meteo. Tutti e 3 i Repository gestiscono i dati localmente con Room e in remoto con Firebase.

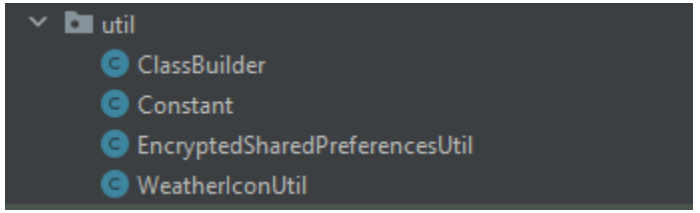
E' presente un'interfaccia di callback per ogni repository, che definisce i metodi di callback che saranno invocati per notificare i risultati delle operazioni asincrone.







Util



Nella cartella Util sono presenti tutte quelle classi che non appartengo alle macroaree elencate prima. Sono presenti:

- **ClassBuilder**: questa classe è implementata come un singleton, quindi ne verrà creata una sola istanza. Essa andrà a creare e a configurare le istanze per le repository corrispondenti, infatti sono presenti i metodi **getUserRepository(), getDataRepository()** e **getWeatherRepository().** Mentre col metodo **getWeatherApiService()** la classe configura e fornisce accesso a servizi API esterni utilizzando Retrofit.

- **Constant:** sono delle stringhe che andranno a contenere le preferenze impostate.

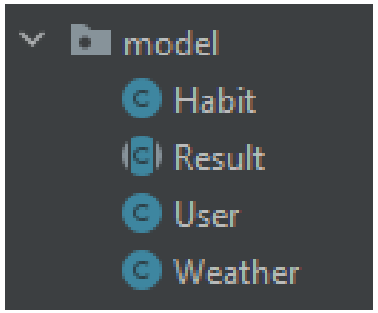
- **EncriptedSharedPreferencesUtil**: contiene i metodi per cancellare leggere e aggiornare le informazioni inserite all'interno delle preferenze.

- **WeatherIconUtil**: contiene tutte le icone del meteo per il giorno e per la notte in base al code ritornato dall'API.

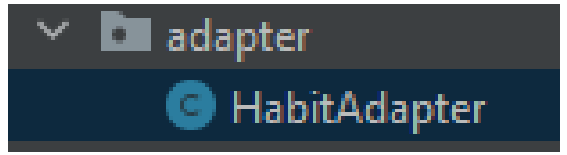
Model

- Classi che rappresentano gli oggetti del progetto

  - **Habit :** contiene tutte le informazioni relative alle abitudini, il titolo, le note, la difficoltà, la polarità e se è «syncato» con il database remoto (offline first)
  - **Result :** classe astratta che viene estesa da HabitSuccess, UserSuccess, WeatherSuccess e Fail
  - **User :** contiene tutte le informazioni relative all'utente, mail, password, livello, vita, ID e logout
  - **Weather :** contiene tutte le informazioni relative al meteo, la città, i gradi, la temperatura, i codici (relativi alle icone) e il giorno/notte



Adapter



Contiene la classe HabitAdapter utilizzata per gestire una RecyclerView dove ogni elemento è un'abitudine.

E' presente anche un'interfaccia OnItemClickListener che contiene i metodi per:

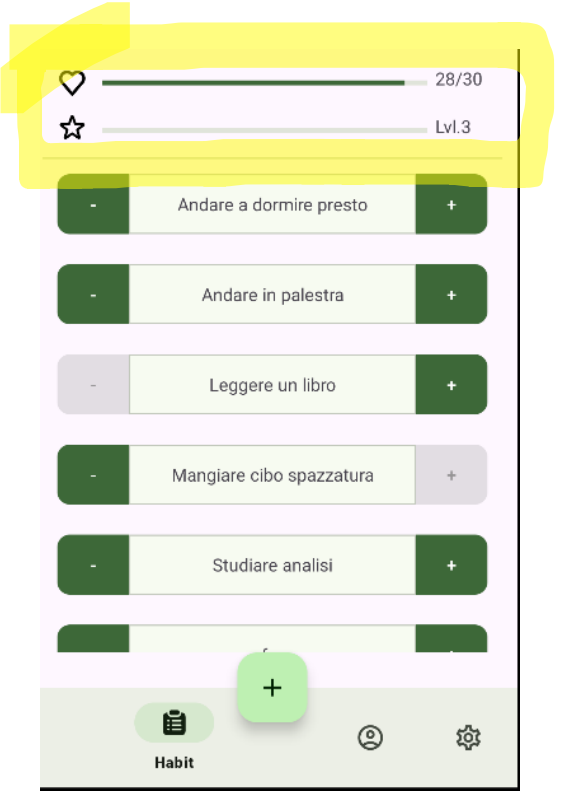
- la pressione del bottone positivo
- la pressione del bottone negativo
- la pressione dell'intera Cardview, che verranno invocati alla rispettiva pressione. Questi metodi sono "overraidati" nell'HabitFragment quando si inizializza l'Adapter.

All'interno dell'adapter:

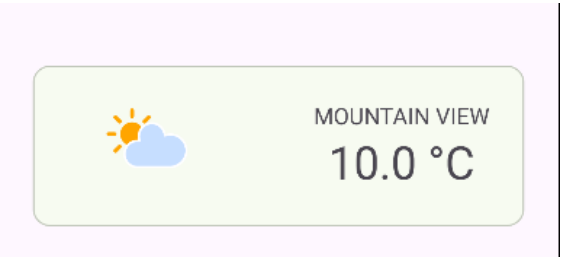
```
public interface OnItemClickListener {

    1 usage  1 implementation  Federico Colciago
    void onPositiveButtonClick(int position);
    1 usage  1 implementation  Federico Colciago
    void onNegativeButtonClick(int position);
    1 usage  1 implementation  Federico Colciago
    void onCardViewClick(int position);
}
```

Funzionalità

TaskFlow è un'app di produttività che utilizza **HP** e **XP** per motivare gli utenti a compiere buone abitudini. Completa attività per guadagnare XP, altrimenti potresti perdere punti vita e l’esperienza guadagnata!



L’utente può inoltre controllare il **meteo** prima di compiere le sue abitudini in una cardview con un design accattivante e che dispone di tutte le informazioni necessarie.

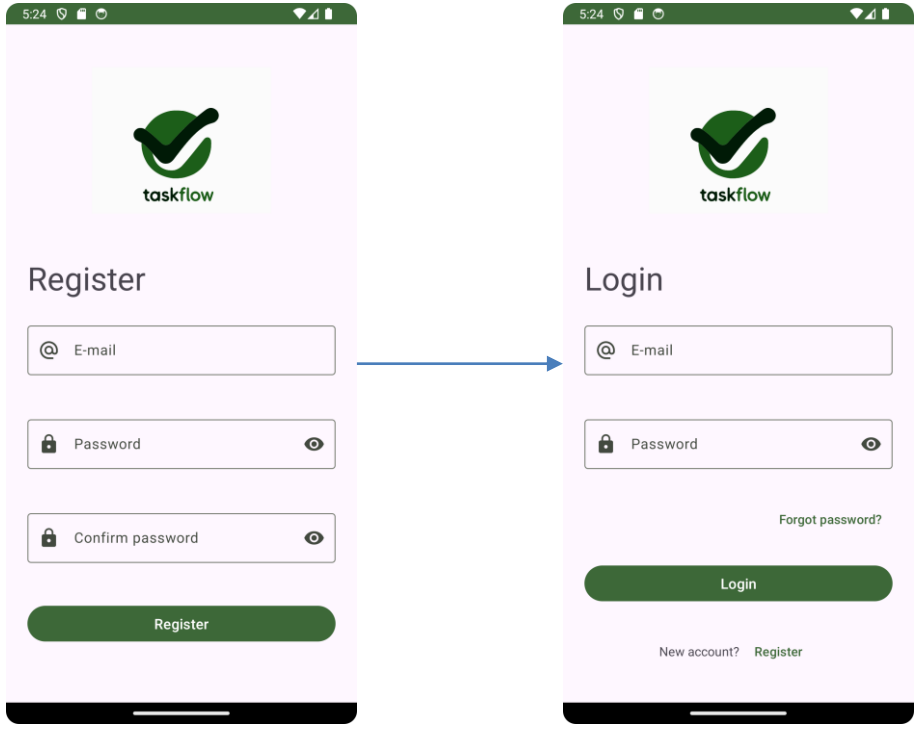


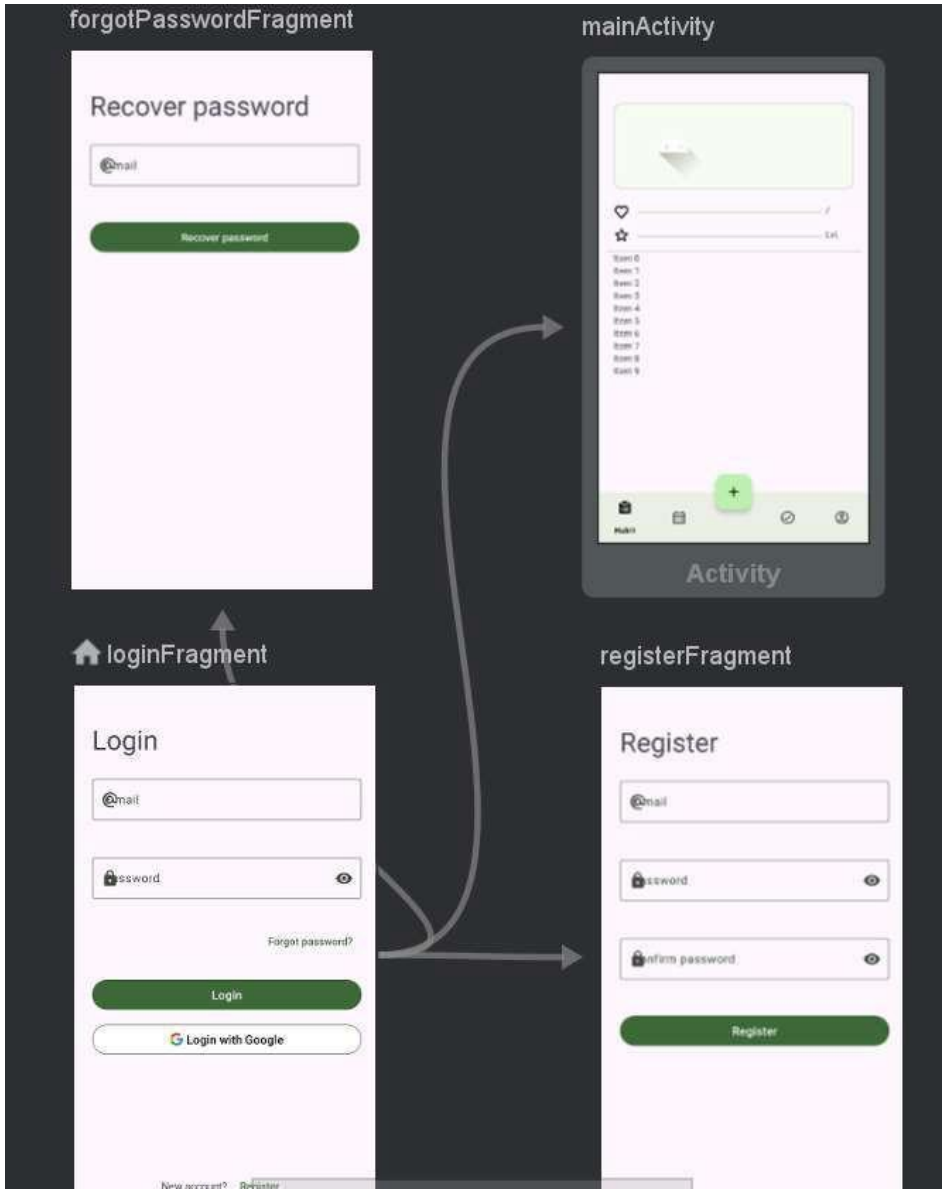
Login e registrazione :

- La prima Activity avviata all'apertura dell'app è la **WelcomeActivity**, associata al **layout activity_welcome**, che controlla lingua e tema del dispositivo per salvarle nelle **encryptedSharedPreferences**, per garantire coerenza al riavvio dell'app. Utilizza poi il **NavController** per navigare verso la schermata di login, se l'utente è già registrato, controllando se esistono i dati nel database remoto

- Arrivati alla registrazione l'utente inserisce i propri dati, ne viene controllata la correttezza, si salvano in remoto e in locale (**encrypted**) e poi si ritorna alla schermata di login per inserire i dati

- Arrivati al **LoginFragment**, associato al **layout fragment_login**, l'utente può loggarsi mettendo la propria mail e password, viene chiamato il metodo signIn, e se tutto è andato a buon fine le informazioni dell'utente già salvate in remoto vengono salvate anche in locale tramite le encryptedSharedPreferences, se non sono state già salvate. Se l'utente non è ancora registrato può premere sull'apposito pulsante.



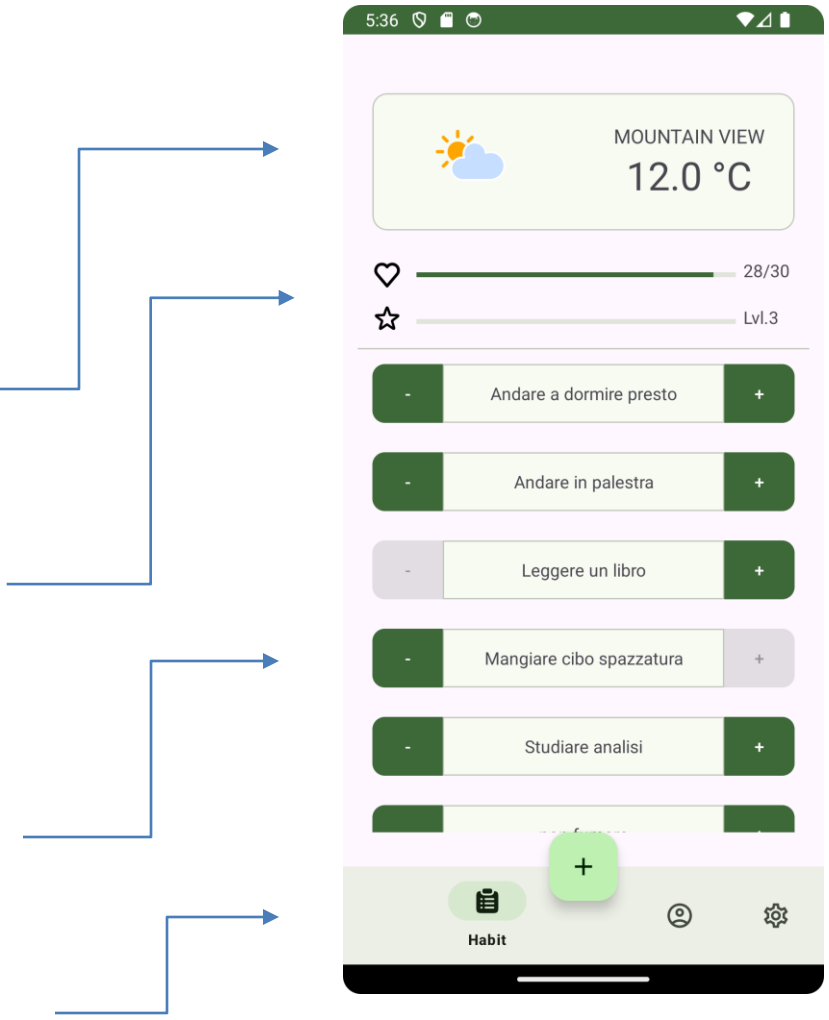


- Se l'utente si dimentica la password può cliccare sul pulsante **password dimenticata?** per passare al **ForgotPasswordFragment**

che invierà una mail all'utente, quella inserita, per resettare la password. Il metodo di recupero password è gestito tutto da Firebase

Home :

- Una volta effettuato l'accesso all'app, ci si ritrova nella schermata principale, la **MainActivity**, associata al **layout activity_main**.
- Nella parte superiore abbiamo tutte le informazioni relative al **meteo locale**, gestite tutte con una **Weather API**.
- Subito sotto abbiamo le informazioni relative al **livello** e alla **salute** dell'utente recuperate grazie al **DataViewModel**, che verranno poi osservate per essere cambiate dinamicamente alla pressione dei tasti + e – delle abitudini.
- Al centro della schermata è presente un **fragment container**, contenente i vari fragment da "switchare", gestito sempre dal **NavController**. Nella parte inferiore si trova la **mainBottomNavigationView** che in base al pulsante premuto porta ai fragment corrispondenti.

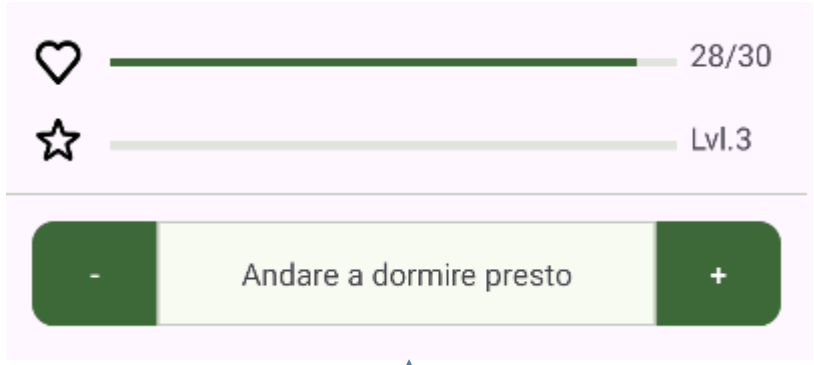


Fragment abitudine

- È associato al **layout fragment_habit**, qui viene visualizzato il **recyclerView contenete tutte le abitudini definite dall'utente**.

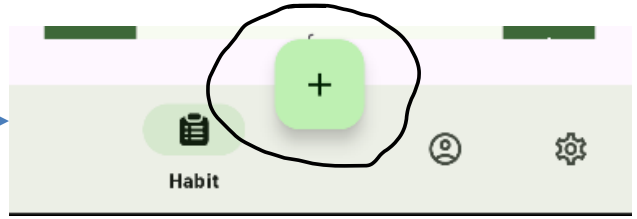
- Ai due lati degli oggetti ci sono i **pulsanti positivo o negativo**, rispettivamente per le abitudine ritenute positive e per quelle negative. Una volta premuto l'apposito pulsante si andrà nel caso del **pulsante positivo**, ad **accumulare esperienza e a recuperare della salute**, se essa è minore del massimo, e **nel caso del pulsante negativo a perdere sia esperienza che salute**. Per questi metodi viene recuperato l'utente loggato e le varie informazioni per la salute e l'esperienza in modo da settarle di conseguenza, sempre grazie allo userViewModel.

- Quando invece viene premuto l'oggetto, quindi viene cliccata la **Cardview**, si entrà nella logica di modifica dell'abitudine. Viene **creato un bundle**, contenitore dell'abitudine selezionata (recuperata dalla posizione), e un intent con la **CreateActivity** come destinazione. **All'intent viene passato il bundle** che contiene l'oggetto abitudine. **Alla fine verrà startata la nuova attività con l'intent configurato per raggiungere la CreatActivity**

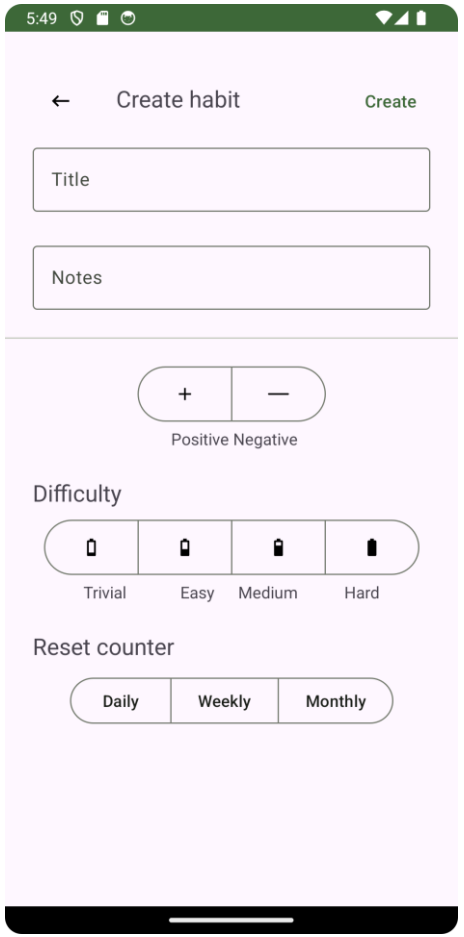


Creazione e modifica abitudine

- Nel centro della schermata principale in basso, è presente un grande pulsante con un più (+).



- ...una volta cliccato si andrà a creare un nuovo intent con la **CreateActivity** come destinazione, e si starterà il nuovo intent per raggiungerla. Una volta raggiunta questa Activity si potrà raggiungere il Fragment di creazione delle abitudini vero e proprio (**CreateHabitFragment**), dove poter inserire, il **titolo**, le eventuali **note**, il grado di **difficoltà** della specifica abitudine. In base al grado di difficoltà si andrà a guadagnare o a perdere salute/esperienza proporzionalmente.



- Una volta premuto il pulsante **createButton** il sistema setta tutti i dati inseriti dall'utente, e, se tutto è corretto, viene chiamato il metodo **saveHabit** per salvare in remoto e locale grazie al dataViewModel (offline first)

Gestione della modifica

- La gestione della modifica viene gestita in modo diverso, ovvero:
- prima si fa un controllo per verificare che il bundle sia vuoto o meno, se non lo è significa che il fragment è stato raggiunto passandogli un bundle con dentro l'abitudine.

```
if (bundle != null) {
    currentHabit = bundle.getParcelable(HABIT);
```

In questo caso il file di layout associato viene modificato visualizzando i pulsanti delete e save, che servono appunto per cancellare/modificare l'abitudine, andando a salvare le modifiche.

```
binding.createButton.setVisibility(View.INVISIBLE);
binding.deleteButton.setVisibility(View.VISIBLE);
binding.saveButton.setVisibility(View.VISIBLE);
```

Alla **pressione del saveButton** si settano tutte le modifiche attuate dall'utente e si chiama il metodo **updateHabit** per salvare la nuova abitudine in **remoto e in locale** (offline first and synced). Allo stesso modo quando si preme il puslante **deleteButton** si chiamerà il metodo **deleteHabit** che lo cancellerà sia localmente che in remoto, in questo caso i metodi sono gestiti col **dataViewModel.**

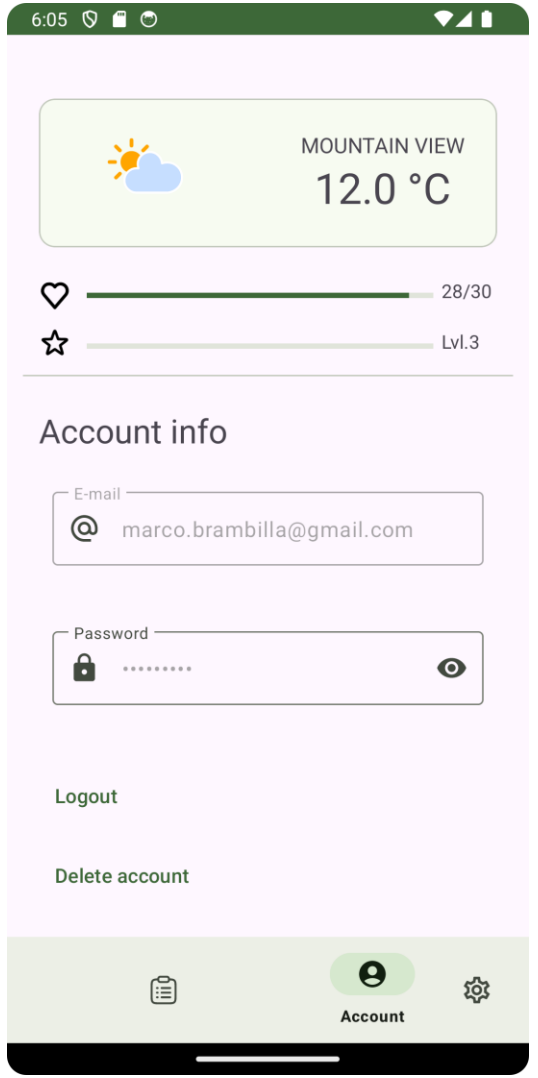
Account :

L'area **Account** presente una singola schermata contenente le **informazioni** inserite in fase di **registrazione**, **email** e **password**, e **due pulsanti**. I due pulsanti servono per effettuare il **logout** e **cancellare l'account creato**.

Nella zona delle impostazione l'utente può scegliere autonomamente che lingua e che tema dovrà avere l'applicazione.

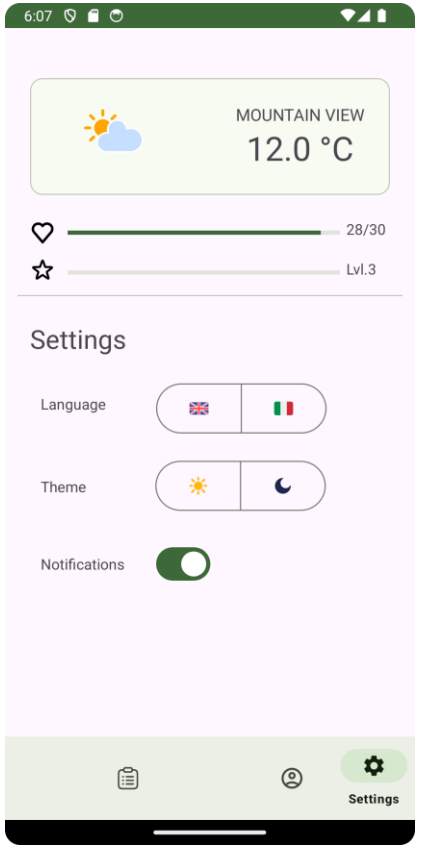
Per **visualizzare la mail e la password** dell'utente vengono lette le informazioni localmente tramite le **encryptedSharedPreferences** per poi visualizzarle.

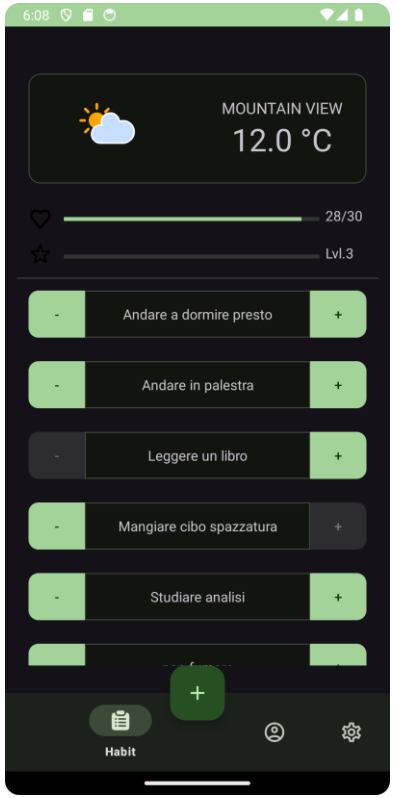
Per il **logout**, alla pressione del pulsante viene chiamato il metodo logout grazie allo **userViewModel** e si viene poi riportati alla **WelcomeActivity** tramite un intent. Per la cancellazione dell'account, alla pressione del tasto vengono cancellate le informazioni localmente tramite il metodo **deleteCredentialInformationsEncrypted**, e in remoto col metodo **deleteUser tramite lo UserViewModel.**



Impostazioni :

- Nelle **impostazioni** l'utente può modificare la **lingua** e il **tema** dell'app.
- Semplicemente alla pressione dei rispettivi pulsanti, si andrà a scrivere nelle **encryptedSharedPreferences** la preferenza di lingua o tema scelta e verrà restartata la MainActivity tramite un intent.





- La **MainActivity** andrà poi a leggere le encryptedSharedPreferences di lingua e tema per capire la preferenza e settarla di conseguenza

Sviluppi futuri :

- Nelle idee iniziali c'era quella di inserire un **sistema di notifiche**, attivabili dalla sezione impostazioni, che potesse ricordare all'utente di svolgere una determinata abitudine.
- Abbiamo anche pensato di inserire un **Reset Counter** come informazione relativa alla abitudine creata, che andrà settata su **giornaliera, settimanale o mensile**. Infatti un reset specifico permetterà all'utente di avere un numero massimo di click sui button **+** e **–** giornalmente/settimanalmente/mensilmente. **Molto utile quindi il sistema di notifiche menzionato prima.**

- **Inserimento di (Daily)** ovvero azioni non programmate da svolgere solo quando necessario/possibile.
- **Inserimento di (ToDo)** ovvero azioni che vanno fatte una sola volta.