



UNIVERSITÀ  
POLITECNICA  
DELLE MARCHE

FACOLTÀ DI INGEGNERIA  
CORSO DI LAUREA IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

---

# Progetto di Programmazione Mobile

**Applicazione HabitConnect kotlin-flutter**

Autore:  
**Massimo Mecarelli**

# Contents

<b>1 Descrizione In Linguaggio Naturale</b>	<b>1</b>
<b>2 Glossario</b>	<b>2</b>
<b>3 Kotlin App</b>	<b>3</b>
3.1 Analisi dei Requisiti . . . . .	3
3.1.1 REQUISITI FUNZIONALI . . . . .	4
3.1.2 REQUISITI NON FUNZIONALI . . . . .	4
3.2 Casi d'Uso . . . . .	5
3.2.1 UC: Gestione Tasks . . . . .	6
3.2.2 UC: Gestione Focus . . . . .	7
3.2.3 UC: Gestione Community . . . . .	9
3.2.4 UC: Gestione Info (Tutorial) . . . . .	11
3.2.5 UC: Gestione Reminders . . . . .	12
3.3 Mappa dell'Architettura . . . . .	14
3.3.1 Architettura della Home di Avvio . . . . .	16
3.3.2 Architettura Gestione Tasks . . . . .	17
3.3.3 Architettura Gestione Reminders . . . . .	18
3.3.4 Architettura Gestione Focus . . . . .	19
3.3.5 Architettura Tutorial . . . . .	19
3.3.6 Architettura Gestione Community . . . . .	20
3.4 Data Storage . . . . .	21
3.4.1 Database . . . . .	21
3.4.2 Shared Preferences . . . . .	21
3.4.3 Firebase . . . . .	22
3.5 MockUp UI . . . . .	23
3.5.1 NavBar . . . . .	23
3.5.2 Sezione Tasks . . . . .	23
3.5.3 Sezione Reminders . . . . .	25
3.5.4 Sezione Community . . . . .	27
3.5.5 Sezione Focus . . . . .	30
3.5.6 Sezione Tutorial . . . . .	32
3.6 Rappresentazione Grafica dei Casi d'Uso . . . . .	35
3.6.1 Inserimento Nuovo Reminder . . . . .	35
3.6.2 Inserimento Nuovo Task . . . . .	36
3.6.3 Accesso Community . . . . .	36
3.6.4 Inserimento Nuovo Commento Community . . . . .	37
3.6.5 Utilizzo Timer Focus . . . . .	37
3.7 Approccio e Problematiche di Sviluppo . . . . .	38
3.7.1 Bug Risolti . . . . .	38
3.7.2 Bug . . . . .	38
3.7.3 Librerie di Terze Parti . . . . .	38
3.8 Descrizione Componenti . . . . .	38
3.8.1 MainActivity . . . . .	38
3.8.2 AppDatabase . . . . .	38

3.8.3	TaskDao . . . . .	39
3.8.4	ReminderDao . . . . .	39
3.8.5	Task . . . . .	39
3.8.6	Reminder . . . . .	39
3.8.7	TASKS: FragmentTask . . . . .	39
3.8.8	TaskAdapter . . . . .	39
3.8.9	FragmentTaskViewModel . . . . .	40
3.8.10	NewTaskSheet . . . . .	40
3.8.11	TUTORIAL: TutorialActivity . . . . .	40
3.8.12	FOCUS: TimerActivity . . . . .	40
3.8.13	PrefUtil . . . . .	41
3.8.14	REMINDERS: FragmentReminders . . . . .	42
3.8.15	NewReminderSheet . . . . .	42
3.8.16	ReminderAdapter . . . . .	43
3.8.17	FragmentRemindersViewModel . . . . .	43
3.8.18	COMMUNITY: CommunityActivity . . . . .	43
3.8.19	NewCommentSheet . . . . .	43
3.8.20	Comment . . . . .	44
3.8.21	Response . . . . .	44
3.8.22	Checker . . . . .	44
3.8.23	ActivityCommunityViewModel . . . . .	44
3.8.24	CommentAdapter . . . . .	45
3.8.25	SignInActivity . . . . .	45
3.8.26	SignUpActivity . . . . .	45
3.9	Testing . . . . .	45
3.9.1	DatabaseTest . . . . .	45
3.9.2	MainActivityTest . . . . .	46
3.9.3	CheckerTest . . . . .	46
3.9.4	PrefUtilTest . . . . .	46
3.9.5	TutorialActivityTest . . . . .	46
3.9.6	TimerActivityTest . . . . .	46
<b>4</b>	<b>Flutter App</b>	<b>47</b>
4.1	Introduzione . . . . .	47
4.2	Analisi dei Requisiti . . . . .	47
4.2.1	REQUISITI FUNZIONALI . . . . .	48
4.2.2	REQUISITI NON FUNZIONALI . . . . .	48
4.3	Casi d'Uso . . . . .	49
4.3.1	UC: Gestione Tasks . . . . .	50
4.3.2	UC: Gestione Community . . . . .	52
4.3.3	UC: Gestione Info (Tutorial) . . . . .	54
4.4	Mappa dell'Architettura . . . . .	55
4.5	Database . . . . .	55
4.5.1	Firebase . . . . .	55
4.6	Mockup UI . . . . .	56
4.6.1	Home . . . . .	56

4.6.2	Community . . . . .	59
4.6.3	Tutorial . . . . .	61
4.7	Rappresentazione Grafica dei Casi d'Uso . . . . .	63
4.7.1	Inserimento Nuovo Task . . . . .	63
4.7.2	Accesso Community . . . . .	63
4.7.3	Inserimento Nuovo Commento Community . . . . .	64
4.8	Commenti sull'Approccio di Sviluppo . . . . .	64
4.8.1	Utilizzo Immagini Tutorial . . . . .	64
4.8.2	Componenti Importate . . . . .	64
4.9	Descrizione delle Componenti . . . . .	64
4.9.1	Main . . . . .	64
4.9.2	TASKS : Tasks . . . . .	65
4.9.3	New Task Dialog . . . . .	65
4.9.4	New Task Dialog . . . . .	65
4.9.5	COMMUNITY : SignIn Or SignUp Page . . . . .	65
4.9.6	Login Page . . . . .	66
4.9.7	Register Page . . . . .	66
4.9.8	Custom Button . . . . .	66
4.9.9	Community . . . . .	66
4.9.10	Comment . . . . .	66
4.9.11	TUTORIAL . . . . .	67

# Progetto Programmazione Mobile

Massimo Mecarelli

September 2023

## 1 Descrizione In Linguaggio Naturale

HabitConnect è un'app che si occupa di monitorare todo Lists in modo tale da avere dei task a cui viene associato un obiettivo numerico da poter decrementare manualmente, e di fornire dei reminders, in cui c'è la possibilità di annotarsi anche la data e l'orario. Un'ulteriore funzionalità è la "modalità focus", una particolare modalità di utilizzo che quando entra in funzione è ben visibile attraverso una UI dedicata, il suo scopo è quello di aiutare la concentrazione silenziando notifiche e attivando il non disturbare, impostando un timer che regola la durata delle sessioni di lavoro e/o delle pause. Esso si mette in pausa se l'utente cambia schermata, in questo modo si cerca di evitare l'uso del dispositivo durante le sessioni di lavoro. Infine la sezione Community prevede una sezione di commenti, in cui gli utenti registrati si possono scambiare idee e consigli per lo studio. L'accesso al sistema deve avvenire attraverso delle credenziali che comprendono un indirizzo email e una password di almeno 6 caratteri. I commenti vengono visualizzati in una lista in cui è visibile il nome dell'utente che ha postato, la data, l'ora e il testo.

L'app quindi risulta flessibile e modulare nel proprio utilizzo a seconda delle particolari necessità, le macro categorie di utilizzo sono:

- Gestione delle proprie abitudini con tasks e reminders
- Modalità focus, pensata inizialmente per gli studenti, ma utile anche in diversi contesti
- Sezione community in cui gli utenti possono confrontarsi attraverso commenti scambiandosi idee e consigli

## 2 Glossario

Glossario

Termino	Definizione	Tipo	Sinonimi
Reminder	Indica un particolare testo che viene inserito associandolo a una data e un orario in cui può tornare utile	Specifico dell'app	
Task	Indica un breve testo associato ad un obiettivo numerico che decrementa fino a zero	Specifico dell'app	
Modalità focus	Indica una modalità in cui viene attivato il do not disturb e messa la suoneria silenziosa	Specifico dell'app	
Todo List	Lista di tasks	Specifico dell'app	Liste di tasks
Commenti		Specifico dell'app	Consigli di studio, idee
Do not disturb	Modalità che silenzia le notifiche	Caratteristica del dispositivo	

Figure 1: Glossario

### 3 Kotlin App

#### 3.1 Analisi dei Requisiti

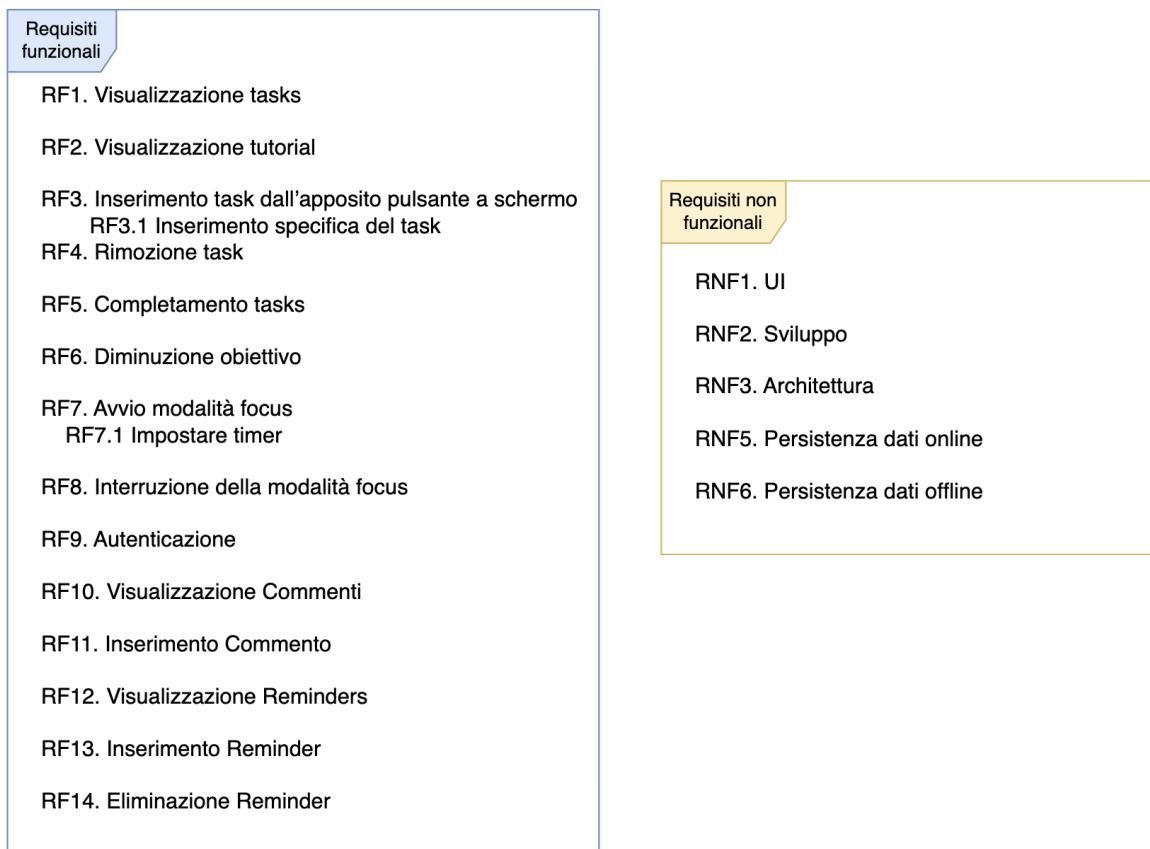


Figure 2: Requisiti di Sistema

### **3.1.1 REQUISITI FUNZIONALI**

- RF1. Visualizzazione tasks** : L'utente deve poter visualizzare i propri tasks nella home
- RF2. Visualizzazione tutorial** : L'app deve fornire all'utente informazioni sulle modalità di utilizzo delle proprie funzionalità
- RF3. Inserimento task dall'apposito pulsante a schermo** : L'utente deve poter inserire un nuovo task
- RF3.1 Inserimento specifica del task** : L'utente deve poter specificare i parametri del task inserendo manualmente il testo e l'obiettivo
- RF4. Rimozione task** : L'utente deve poter rimuovere un task con uno swipe e il sistema deve permettere un UNDO
- RF5. Completamento tasks** : L'utente deve poter visualizzare i tasks completati
- RF6. Diminuzione obiettivo** : L'utente deve poter diminuire l'obiettivo presente su ciascun task
- RF7. Avvio modalità focus** : L'utente deve poter avviare la modalità focus
- RF7.1 Impostare timer** : L'utente deve poter impostare il timer della modalità focus
- RF8. Interruzione della modalità focus** : L'utente deve poter interrompere la modalità focus mettendola in pausa o annullandola
- RF9. Autenticazione** : L'utente deve poter autenticarsi al momento dell'accesso nella sezione community
- RF10. Visualizzazione Commenti** : L'utente, una volta fatto l'accesso, deve poter visualizzare i commenti degli utenti
- RF11. Inserimento Commento** : L'utente, una volta fatto l'accesso, deve poter inserire un suo commento
- RF12. Visualizzazione Reminders** : L'utente deve poter visualizzare una lista di Reminders in cui sia ben visibile data, testo e orario
- RF13. Inserimento Reminder** : L'utente deve poter inserire un reminder specificando nome, data, orario
- RF14. Eliminazione Reminder** : L'utente deve poter rimuovere un reminder facendo uno swipe

### **3.1.2 REQUISITI NON FUNZIONALI**

- RNF1. UI** : L'interfaccia deve essere semplice ed intuitiva per migliorare la user experience
- RNF2. Sviluppo** : Il linguaggio di sviluppo è Kotlin
- RNF3. Architettura** : Il pattern architettonico utilizzato è MVVM
- RNF4. Persistenza dati online** : L'attività di community è gestita con server Firebase
- RNF5. Persistenza dati offline** : La gestione di task e reminder è eseguita offline attraverso un db che deve essere gestito attraverso Room

## 3.2 Casi d'Uso

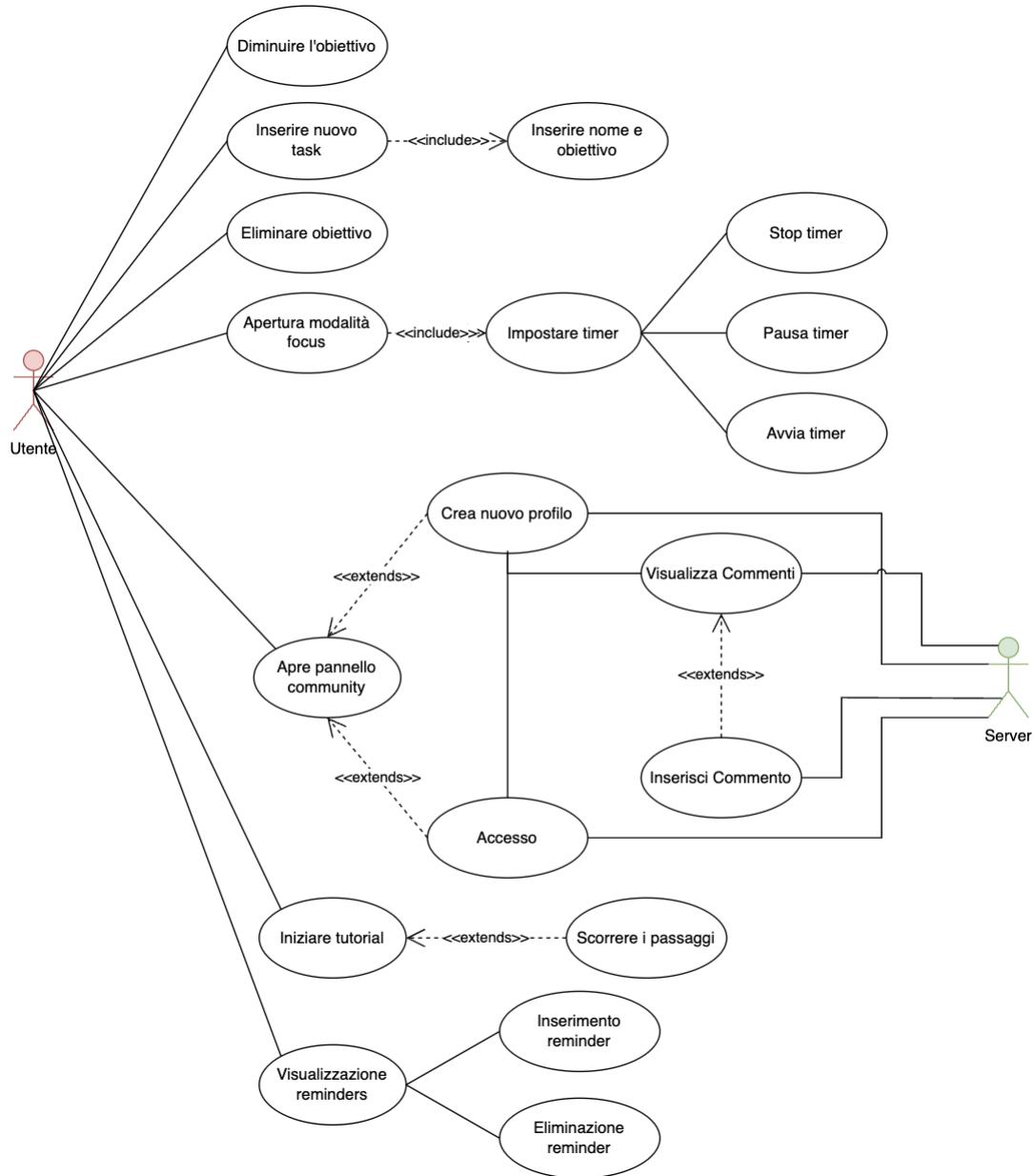


Figure 3: Casi d'Uso

### 3.2.1 UC: Gestione Tasks

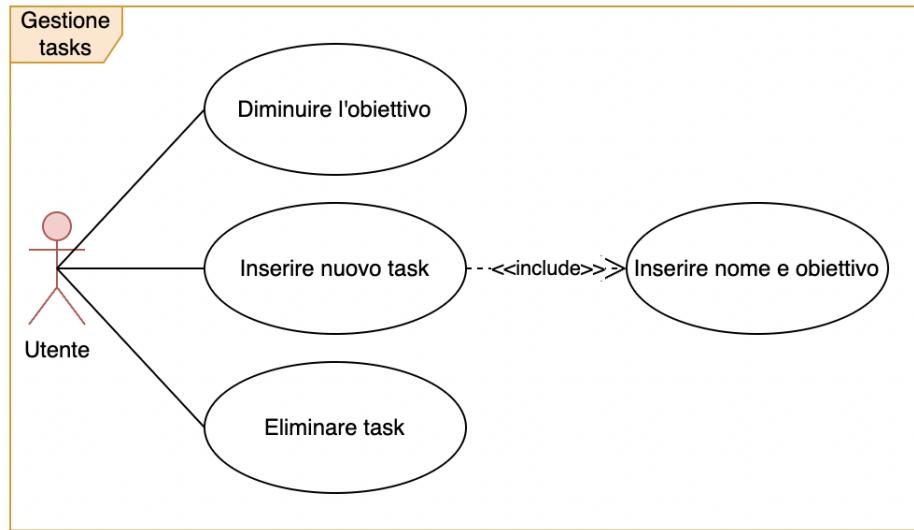


Figure 4: Gestione Tasks

Questi casi d'uso riguardano l'area di funzionalità per la gestione dei tasks

#### Casi d'uso: Inserire nuovo task

Questo caso d'uso si verifica qualora l'utente volesse inserire un nuovo task

**Pre-condizioni:** Nessuna

**Post-condizioni:** Inserimento del task nel sistema

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di inserire un nuovo task nella lista di obiettivi

1. include(Inserire nome e obiettivo)
2. Il sistema visualizza nella home il nuovo task

#### Casi d'uso: Inserire nome e obiettivo

Questo caso d'uso si verifica durante l'inserimento del nuovo obiettivo a sistema

**Pre-condizioni:** L'utente deve aver iniziato la procedura di inserimento

**Post-condizioni:** Inserimento dei dettagli del task

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide il nome e i dettagli dell'obiettivo

1. Il sistema permette l'inserimento del nome e del goal
2. L'utente conferma
3. Il sistema salva i dati

### **Sequenza degli eventi alternativi**

La sequenza alternativa inizia al punto 2

1. L'utente annulla l'operazione cliccando fuori dalla dialog o tornando indietro
2. Il sistema torna alla home

### **Casi d'uso: Eliminare task**

Questo caso d'uso si verifica quando l'utente vuole eliminare un task dalla lista

**Pre-condizioni:** Il task deve essere già registrato a sistema

**Post-condizioni:** Il task non è più presente a sistema

### **Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di rimuovere un task

1. L'utente fa uno swipe sul task a destra o a sinistra
2. Il sistema rimuove dalla memoria il task

### **Casi d'uso: Diminuire obiettivo**

Questo caso d'uso si verifica quando l'utente clicca su un task nella lista

**Pre-condizioni:** Il task deve essere già registrato a sistema

**Post-condizioni:** L'obiettivo del task diminuisce di uno

### **Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di ridurre l'obiettivo di un task

1. L'utente preme sul task nella lista
2. Il sistema riduce il numero che indica l'obiettivo

### **3.2.2 UC: Gestione Focus**

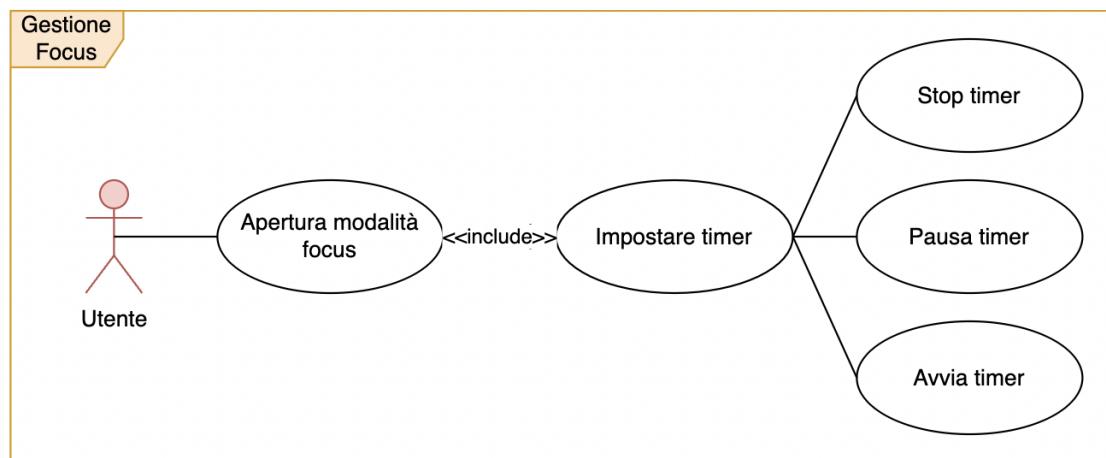


Figure 5: Gestione Focus

Questi casi d'uso riguardano l'area di funzionalità per la modalità focus

**Casi d'uso: Apertura modalità focus**

Questo caso d'uso si verifica quando l'utente vuole impostare la modalità focus

**Pre-condizioni:** Nessuna

**Post-condizioni:** Inizio modalità focus

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di utilizzare la modalità focus attraverso l'apposita icona

1. Il sistema mostra la pagina di configurazione
2. include(Impostare il timer)

**Casi d'uso: Stop Timer**

Questo caso d'uso si verifica quando l'utente vuole interrompere la modalità focus

**Pre-condizioni:** La modalità focus è iniziata

**Post-condizioni:** Modalità focus terminata

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di interrompere la modalità focus

1. L'utente preme il pulsante per interrompere il timer
2. Il sistema interrompe il do not disturb e toglie la suoneria silenziosa

**Sequenza degli eventi alternativi**

La sequenza degli eventi alternativi inizia dal punto 1

1. L'utente chiude l'app
2. il sistema interrompe il timer

**Casi d'uso: Avvia Timer**

Questo caso d'uso si verifica quando l'utente vuole impostare la modalità focus

**Pre-condizioni:** Nessuna

**Post-condizioni:** Modalità focus iniziata

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di avviare la modalità focus

1. L'utente preme play
2. Il sistema attiva il do not disturb e mette la suoneria silenziosa

**Casi d'uso: Pausa Timer**

Questo caso d'uso si verifica quando l'utente vuole mettere in pausa la modalità focus

**Pre-condizioni:** La modalità focus è iniziata

**Post-condizioni:** Modalità focus terminata

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di interrompere la modalità focus

1. L'utente preme il pulsante per mettere in pausa il timer

- Il sistema interrompe il do not disturb e toglie la suoneria silenziosa

**Casi d'uso: Impostare il timer**

Questo caso d'uso si verifica quando l'utente è appena entrato nella modalità focus

**Pre-condizioni:** Apertura modalità focus

**Post-condizioni:** Inizio modalità focus

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di utilizzare la modalità focus attraverso l'apposita icona

- Il sistema permette di selezionare la durata del timer
- L'utente conferma
- Il sistema fa partire il timer

### 3.2.3 UC: Gestione Community

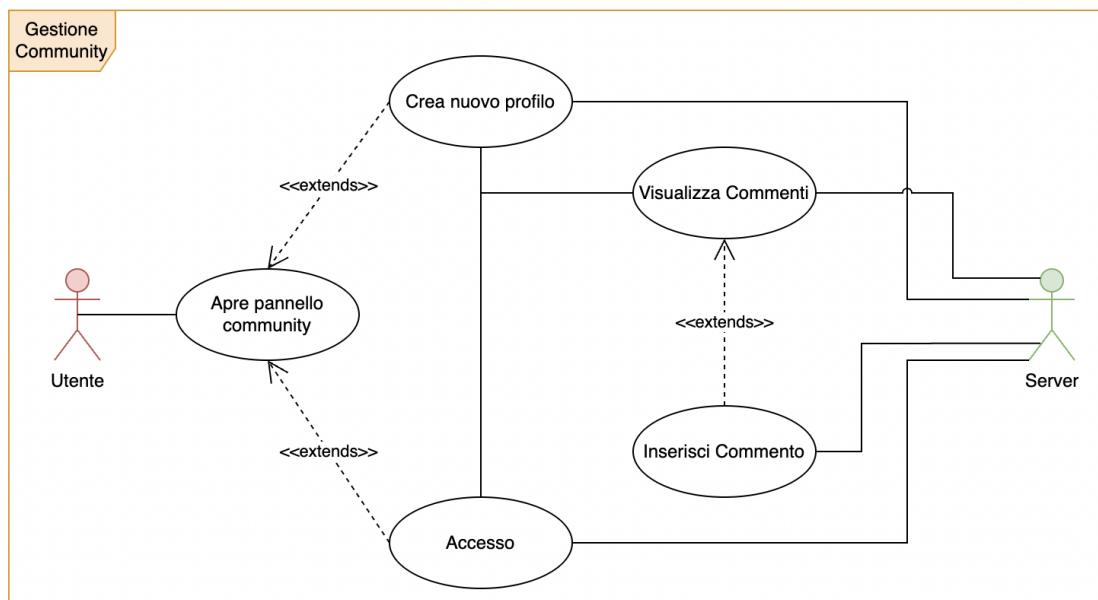


Figure 6: Gestione Community

Questi casi d'uso riguardano l'area di funzionalità dedicate alla community

**Casi d'uso: Aprire pannello community**

Questo caso d'uso si verifica quando l'utente preme sull'icona di Community

**Pre-condizioni:** Nessuna

**Post-condizioni:** Nessuna

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di entrare nell'area community

1. L'utente entra nella schermata di community
2. include(Accesso)

#### **Sequenza degli eventi alternativi**

La sequenza degli eventi alternativi si verifica se l'utente non è già registrato

*Inizia dal punto 2*

1. include(Crea nuovo profilo)
2. Il sistema carica i dati dal server

#### **Casi d'uso: Accesso**

Questo caso d'uso si verifica quando l'utente vuole accedere alla community e possiede un profilo

**Pre-condizioni:** Profilo già registrato

**Post-condizioni:** Nessuna

#### **Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente entra nella schermata di accesso

1. Il sistema permette di inserire le credenziali
2. L'utente accede

#### **Sequenza degli eventi alternativi**

La sequenza degli eventi alternativi inizia al punto 2

1. L'utente inserisce credenziali errate
2. Il sistema chiede nuovamente i dati di accesso

#### **Casi d'uso: Crea nuovo profilo**

Questo caso d'uso si verifica quando l'utente vuole registrarsi nel sistema

**Pre-condizioni:** Nessuna

**Post-condizioni:** Profilo registrato nel server

#### **Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di registrarsi

1. L'utente preme il pulsante per la registrazione
2. Il sistema permette di inserire delle credenziali per l'accesso
3. L'utente accede

#### **Casi d'uso: Visualizza commenti**

Questo caso d'uso si verifica quando l'utente entra nell'area community per visualizzare i commenti degli utenti

**Pre-condizioni:** Accesso effettuato

**Post-condizioni:** Nessuna

#### **Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente termina le fasi di accesso

- Il sistema carica i commenti dal server

**Casi d'uso: Inserisci commento**

Questo caso d'uso si verifica quando l'utente entra nell'area community per inserire un commento

**Pre-condizioni:** Accesso effettuato

**Post-condizioni:** Commento inserito nel sistema

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente termina le fasi di accesso

- L'utente preme sull'icona apposita
- Il sistema permette di inserire un testo
- L'utente conferma
- Il commento viene registrato a sistema

### 3.2.4 UC: Gestione Info (Tutorial)

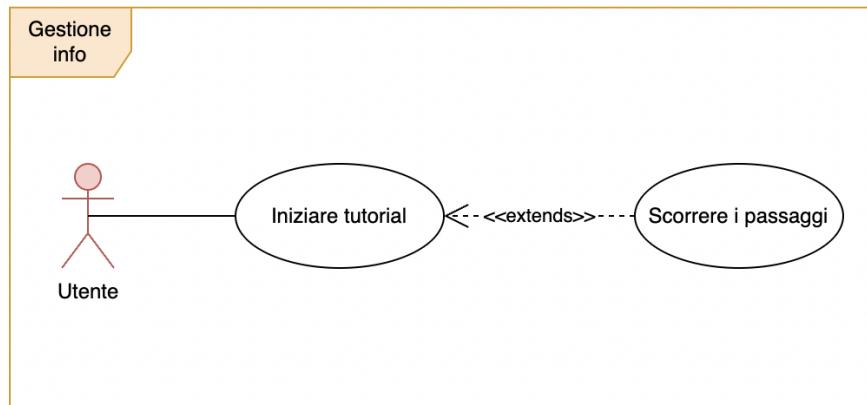


Figure 7: Gestione Info Tutorial

Questi casi d'uso riguardano le funzionalità di tutorial per spiegare il funzionamento dell'app

**Casi d'uso: Iniziare tutorial**

Questo caso d'uso si verifica quando l'utente vuole visualizzare le specifiche di funzionamento della UI

**Pre-condizioni:** Nessuna

**Post-condizioni:** Nessuna

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente termina le fasi di accesso

- L'utente preme sull'icona apposita
- Il sistema mostra la sequenza di immagini del tutorial

3. include (Scorre i passaggi)

**Casi d'uso: Scorre i passaggi**

Questo caso d'uso si verifica quando l'utente è dentro il tutorial

**Pre-condizioni:** Inizio tutorial

**Post-condizioni:** Nessuna

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente entra nel tutorial

1. L'utente scorre le fasi di tutorial avanti e indietro
2. L'utente termina il tutorial

### 3.2.5 UC: Gestione Reminders

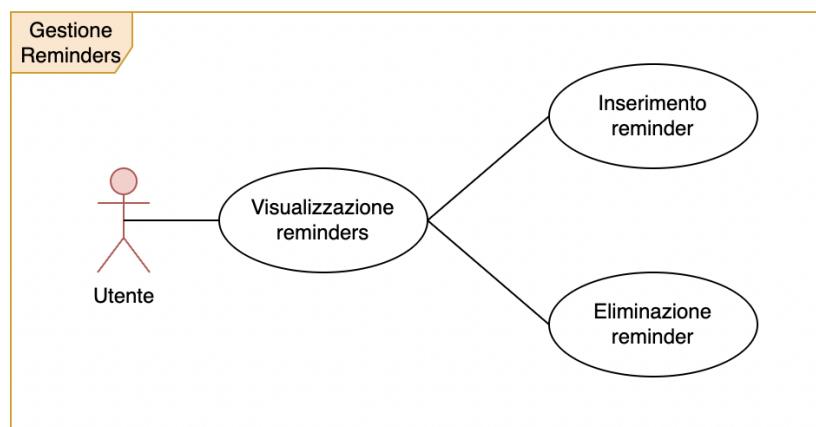


Figure 8: Gestione Reminders

Questi casi d'uso riguardano le funzionalità di gestione di reminders

**Casi d'uso: Visualizzare reminders**

Questo caso d'uso si verifica quando l'utente vuole visualizzare la lista dei reminders

**Pre-condizioni:** Nessuna

**Post-condizioni:** Nessuna

**Sequenza di eventi principali:**

Il caso d'uso si verifica quando l'utente entra nel menu reminders

1. L'utente preme sull'icona apposita
2. Il sistema carica la lista dal db

**Casi d'uso: Inserimento reminder**

Questo caso d'uso si verifica quando l'utente vuole inserire un nuovo reminder

**Pre-condizioni:** L'utente è nel menu Reminders

**Post-condizioni:** Il nuovo reminder è inserito nel db

**Sequenza di eventi principali:**

Il caso d'uso si verifica quando l'utente entra nel menu reminders e preme sul bottone '+'

1. L'utente preme sull'icona apposita
2. Inserisce Testo, data, orario e salva
3. Il sistema salva i dati

**Casi d'uso: Eliminazione reminder**

Questo caso d'uso si verifica quando l'utente vuole eliminare un nuovo reminder

**Pre-condizioni:** L'utente è nel menu Reminders

**Post-condizioni:** Il reminder è eliminato dal db

**Sequenza di eventi principali:**

Il caso d'uso si verifica quando l'utente entra nel menu reminders e fa uno swipe a destra o sinistra su un item della lista dei reminders

1. L'utente fa swipe a destra o sinistra
2. Il sistema lo rimuove dal db

### **3.3 Mappa dell'Architettura**

Per l'architettura è stato usato il pattern architettonico MVVM che si basa sul concetto di “Separation of concerns”, ovvero la separazione della logica dalla UI. Questo permette all'app di essere più modulare semplificando soprattutto la fase di manutenzione, in questo modo le modifiche ad alcune componenti possono rimanere isolate e non intaccare altre strutture dell'architettura. Inoltre rende più lineare la fase di programmazione, permettendo anche di procedere con metodologie agile in quanto ciascuna componente è rappresentativa di un requisito e possono essere integrate in modo incrementale.

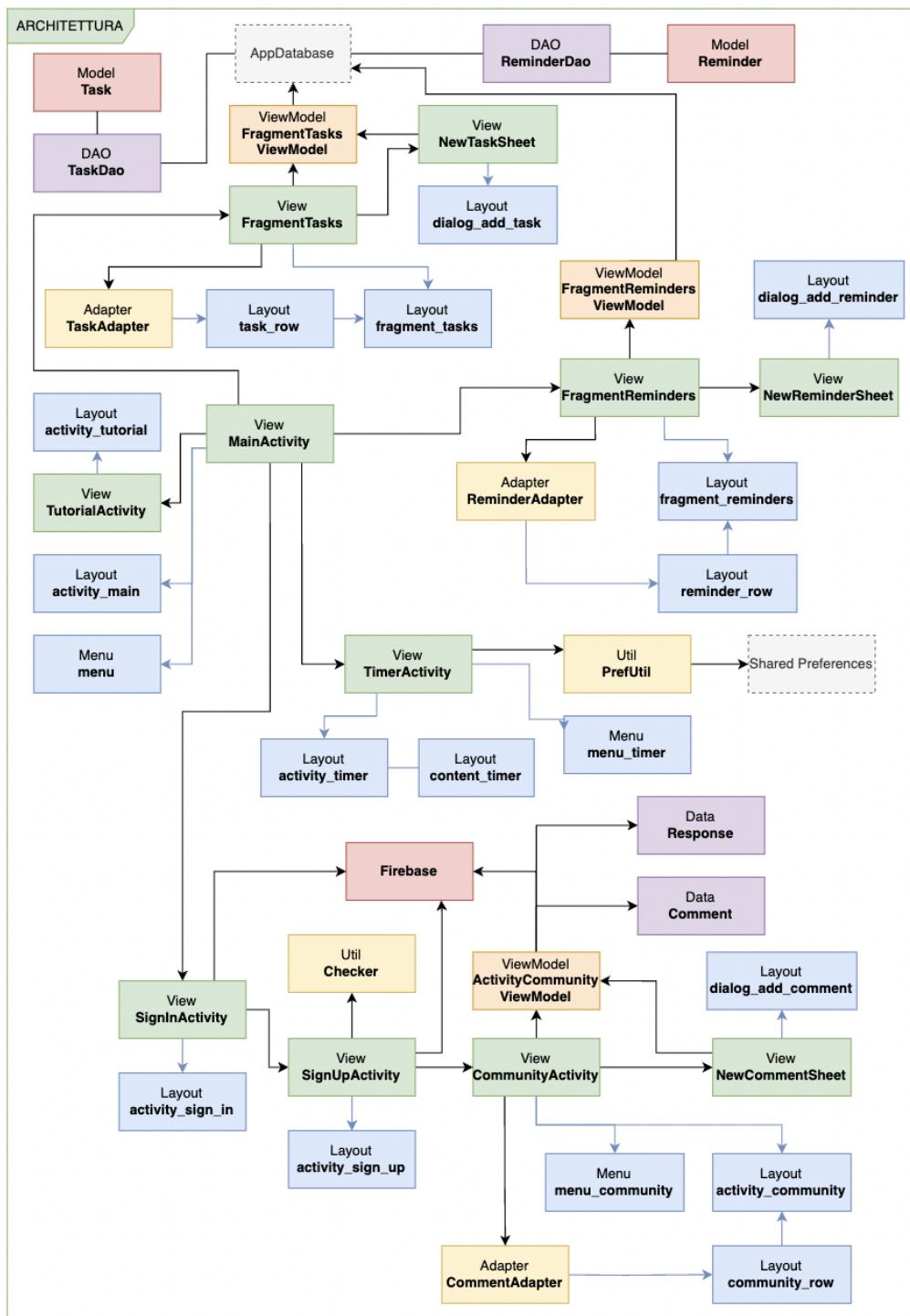


Figure 9: Mappa dell'Architettura

### 3.3.1 Architettura della Home di Avvio

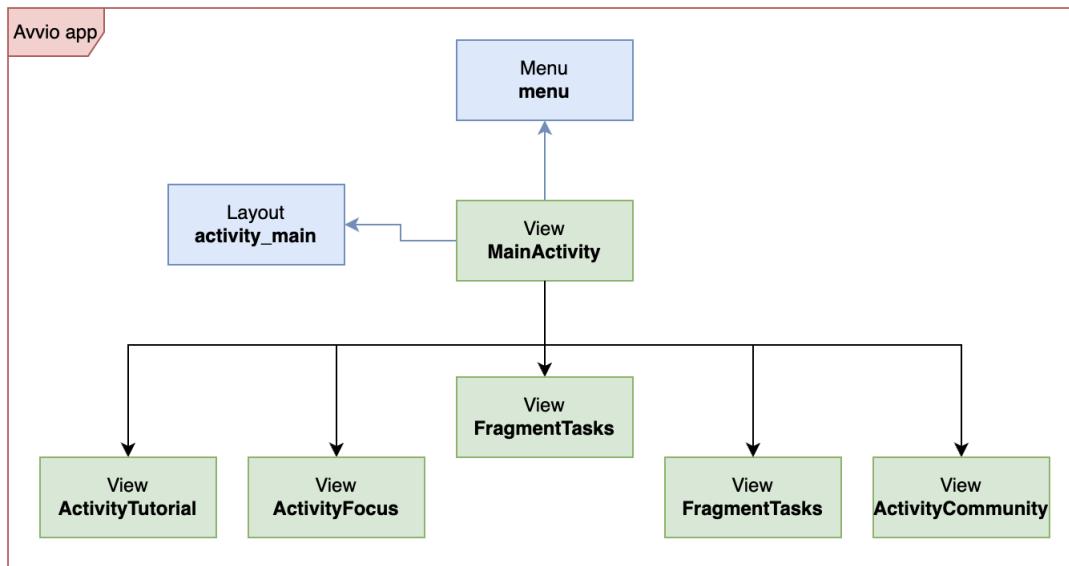


Figure 10: Mappa dell'Architettura d'Avvio

### 3.3.2 Architettura Gestione Tasks

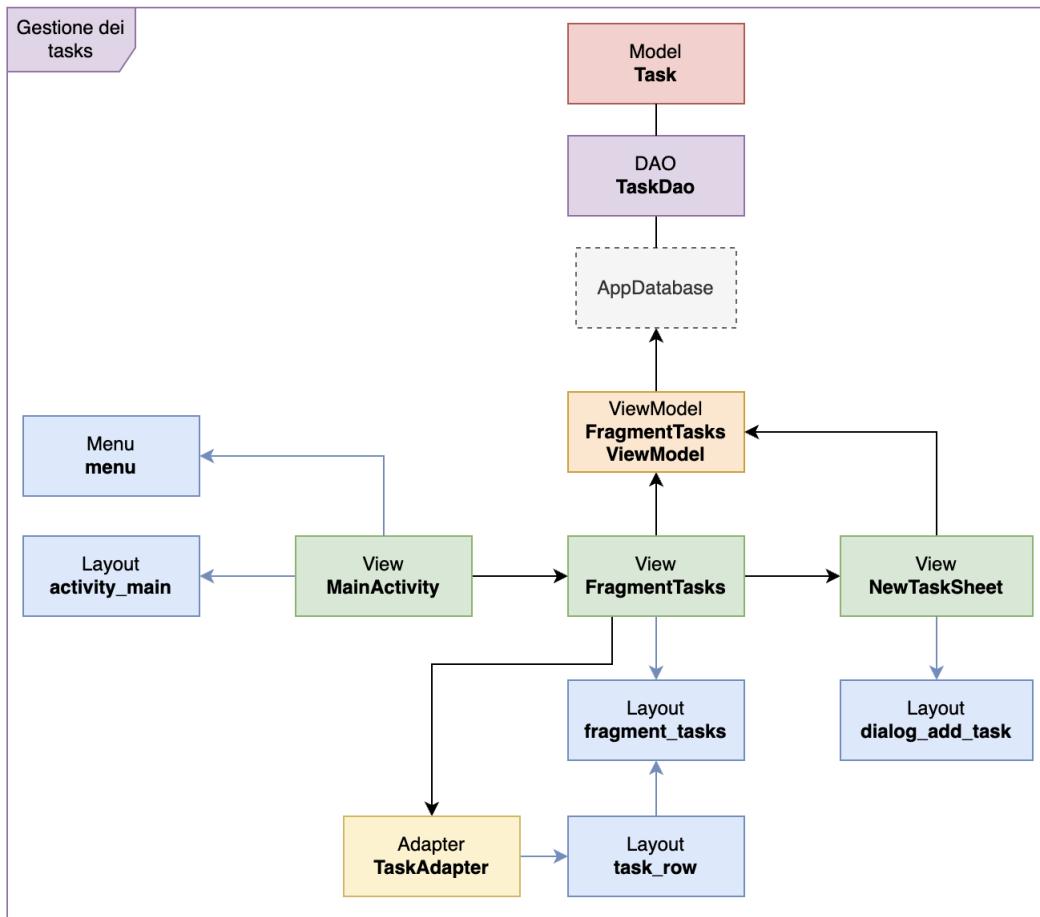


Figure 11: Mappa dell'Architettura di Gestione dei Tasks

### 3.3.3 Architettura Gestione Reminders

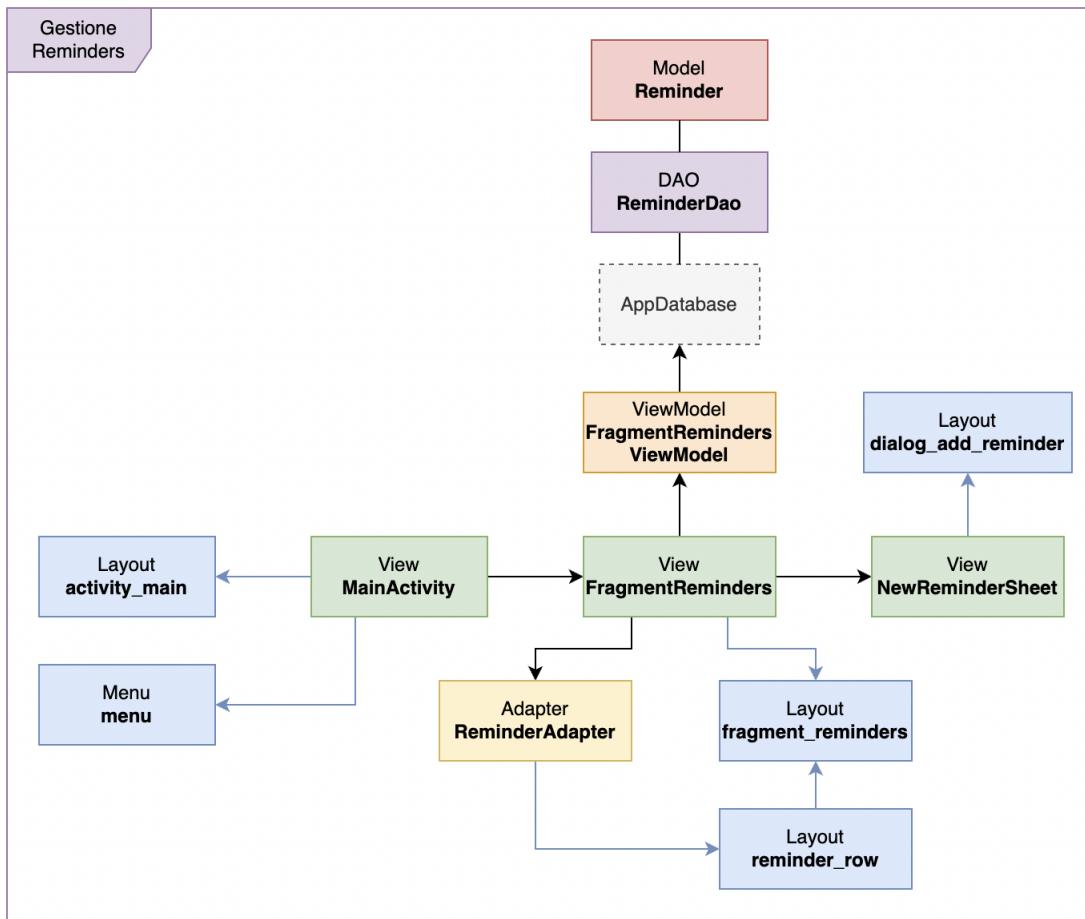


Figure 12: Mappa dell'Architettura di Gestione dei Reminders

### 3.3.4 Architettura Gestione Focus

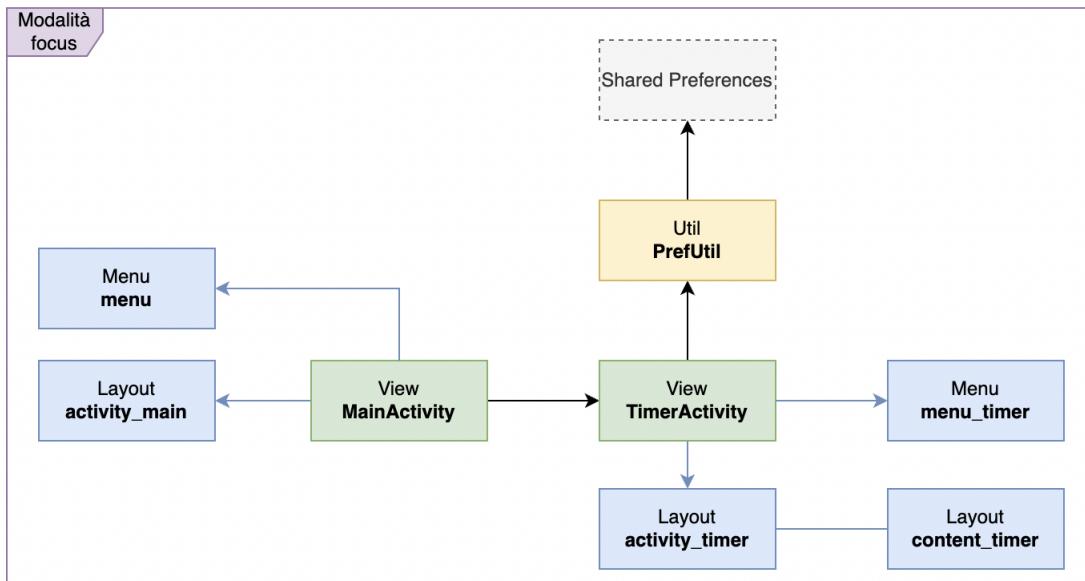


Figure 13: Mappa dell'Architettura di Gestione del Focus

### 3.3.5 Architettura Tutorial

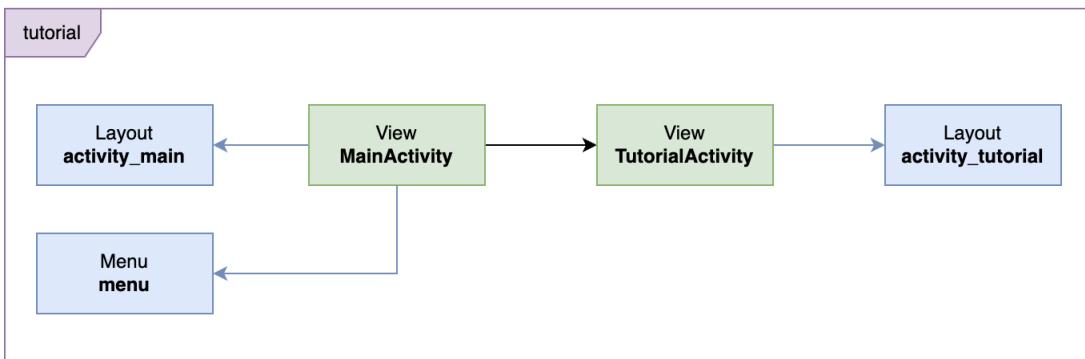


Figure 14: Mappa dell'Architettura del Tutorial

### 3.3.6 Architettura Gestione Community

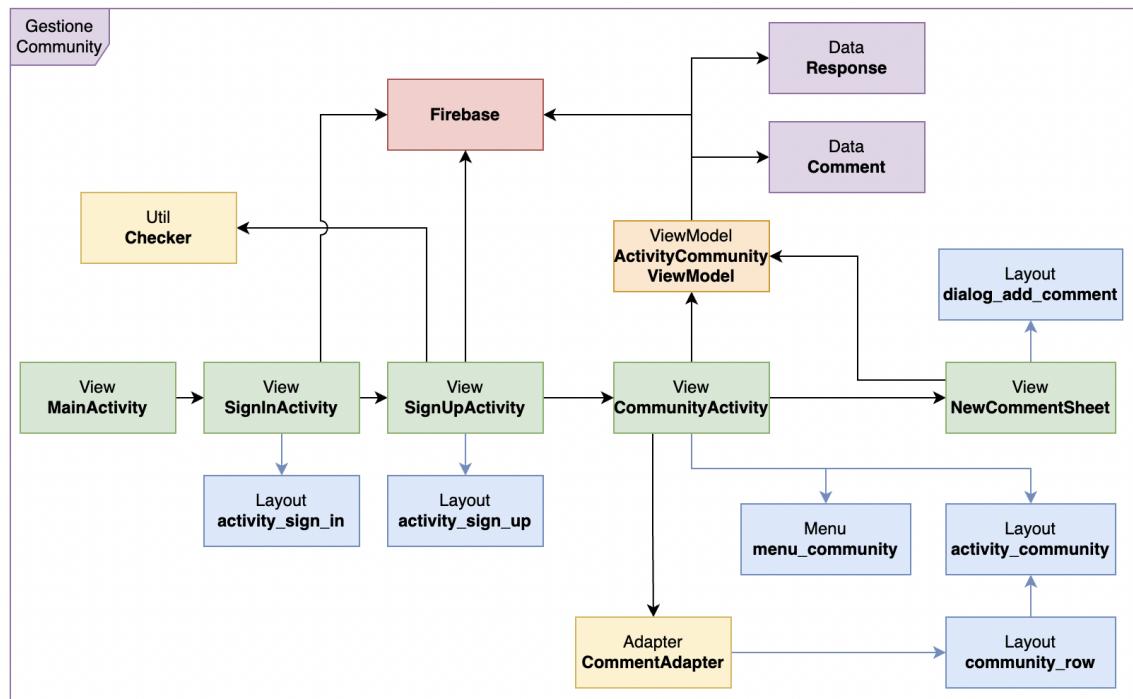


Figure 15: Mappa dell'Architettura di Gestione della Sezione Community

## 3.4 Data Storage

### 3.4.1 Database

Viene utilizzato come Storage il DB *SQLite* di Android per memorizzare i tasks e i reminders dell'utente, si evita di memorizzare questi su server remoto per permettere l'utilizzo di tutte le funzionalità, al di fuori di quelle di community, anche offline. Per interagire con il db ci si è serviti di *Room*.

Entità			
Nome	Dettagli	Attributi	Identificatori
<b>Task</b>	Memorizza i tasks inseriti dall'utente	<b>Id</b> Tipo: Int, autoincrementale <b>Nome</b> Tipo: String <b>Obiettivo</b> Tipo: Int <b>Completo:</b> Tipo: Boolean	Id
<b>Reminder</b>	Memorizza i dettagli dei reminder inseriti dall'utente	<b>Id</b> Tipo: Int, autoincrementale <b>Testo:</b> Tipo: String <b>Data:</b> Tipo: String <b>Orario:</b> Tipo: String	Id

Figure 16: Struttura delle Entità nel DB

### 3.4.2 Shared Preferences

Per quanto riguarda la modalità focus, i dati per la personalizzazione della durata del timer, essendo di dimensioni molto ridotte, sono salvati attraverso delle key nelle Shared Preferences in modo tale da renderli persistenti anche alla chiusura dell'app. Per interagire con esse, viene utilizzata la classe PrefUtil.kt.

### 3.4.3 Firebase

L'intera sezione community utilizza **Firebase Firestore** per memorizzare i commenti della community, inoltre con **Firebase Authentication** si gestiscono gli account utente.

In firebase viene usata una raccolta “commenti” contenente documenti, ciascuno dei quali è un singolo commento. Ogni documento possiede tre attributi:

- **User:** (tipo: String) colui che ha scritto il commento
- **Testo:** (tipo: String) il corpo del commento
- **Timestamp:** (tipo: timestamp) rappresentazione della data e orario (prese lato server)

 (default)	 commenti	 T3QjdNFOvbOHP2vGJFuF
<a href="#">+ Avvia raccolta</a>	<a href="#">+ Aggiungi documento</a>	<a href="#">+ Avvia raccolta</a>
commenti >	T3QjdNFOvbOHP2vGJFuF >	+ Aggiungi campo  te... "pro tip: use Active Learning: Engage with the material actively by s... timestamp: 26 agosto 2023 alle ore 00:00:00 UTC+2 user: "Utente1"

Figure 17: Esempio Struttura della Raccolta e dei Documenti

## 3.5 MockUp UI

### 3.5.1 NavBar



Figure 18: Navbar Light e Dark

### 3.5.2 Sezione Tasks

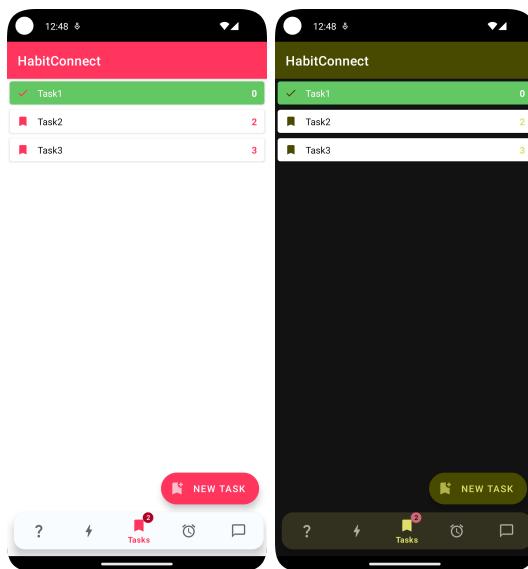


Figure 19: Tasks Light e Dark

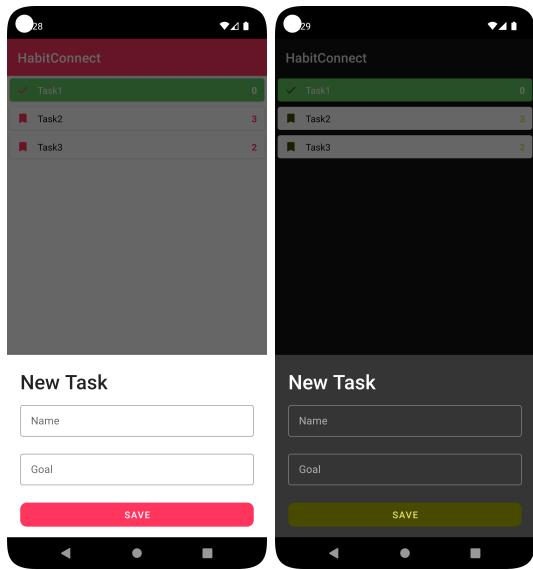


Figure 20: Inserimento Task Light e Dark

### 3.5.3 Sezione Reminders

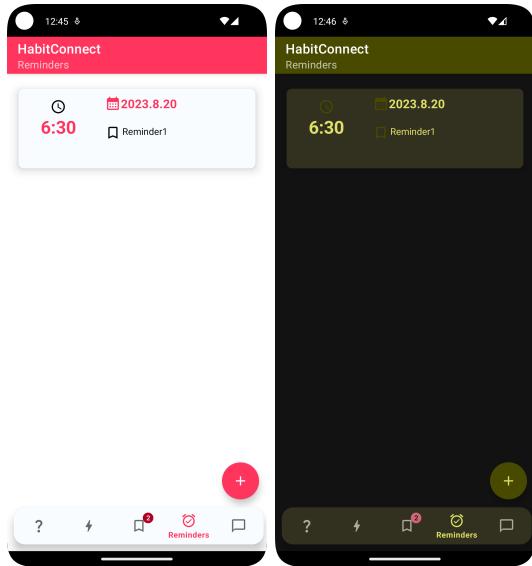


Figure 21: Sezione Reminders Light e Dark

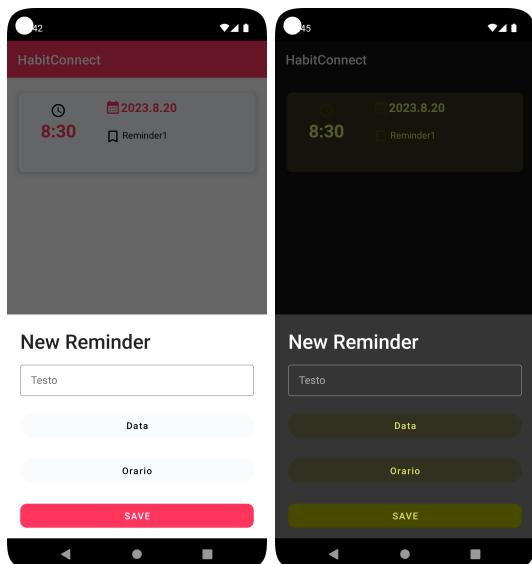


Figure 22: Inserimento Reminder Light e Dark

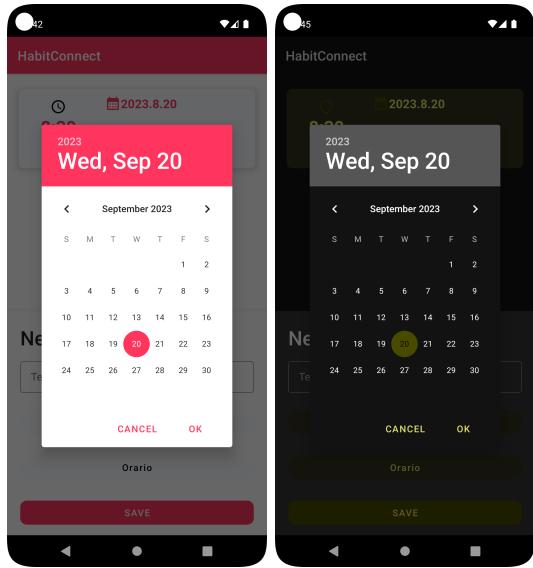


Figure 23: Inserimento Reminder Data Light e Dark

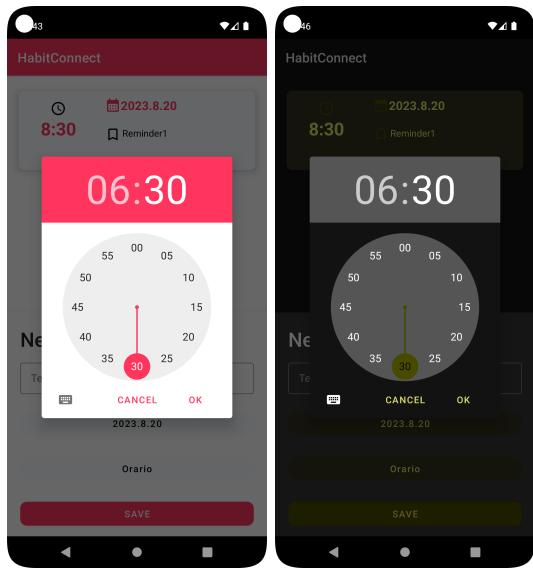


Figure 24: Inserimento Reminder Orario Light e Dark

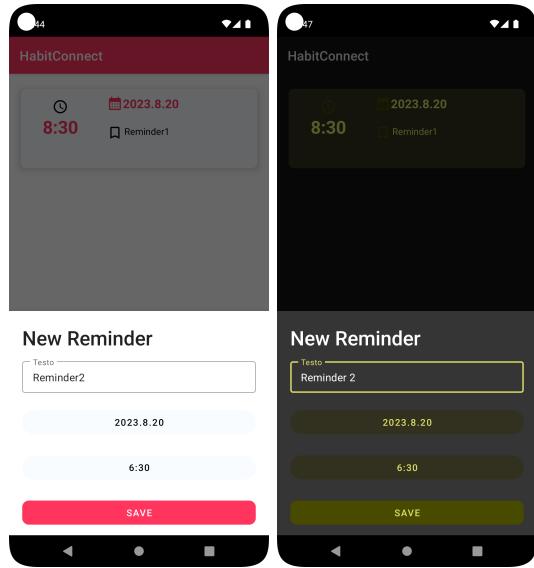


Figure 25: Dati Inseriti Light e Dark

### 3.5.4 Sezione Community

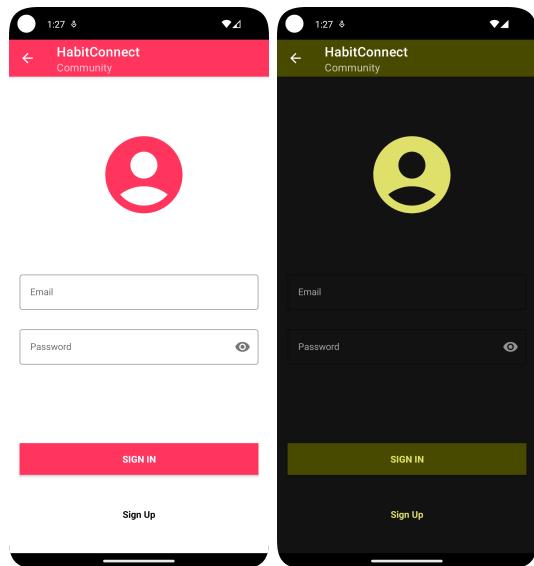


Figure 26: Accesso Light e Dark

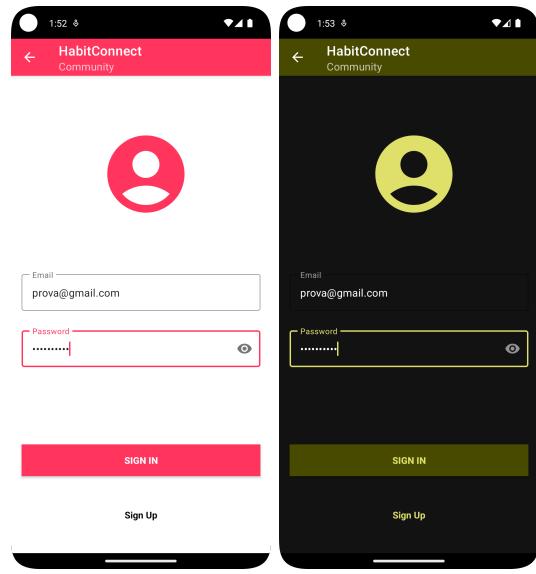


Figure 27: dati di Accesso Light e Dark

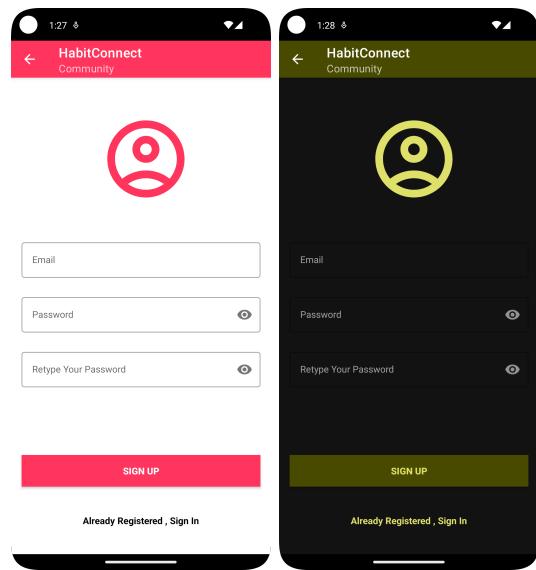


Figure 28: Registrazione Light e Dark

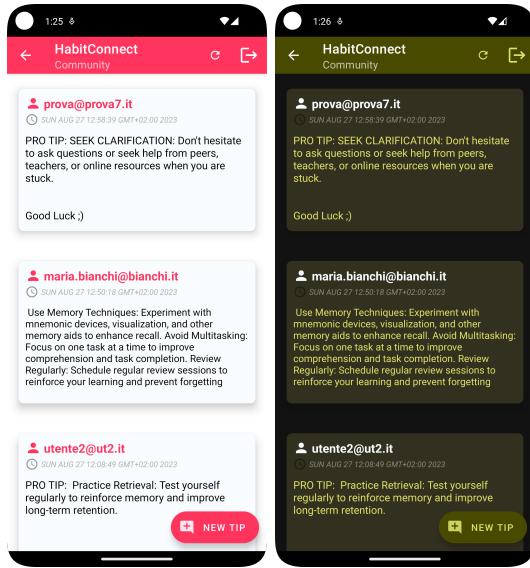


Figure 29: Community Light e Dark

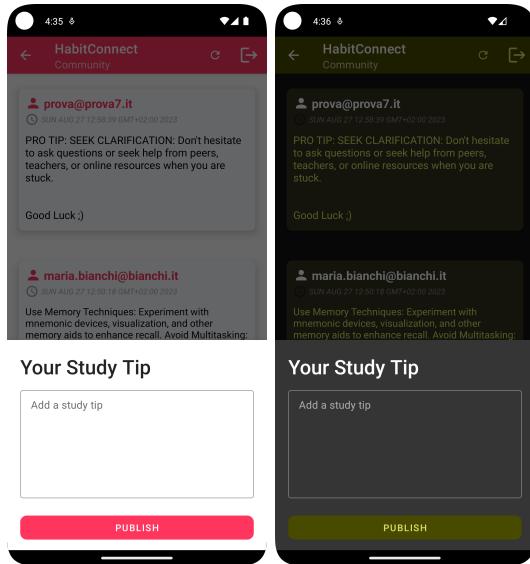


Figure 30: Nuovo Commento Light e Dark

### 3.5.5 Sezione Focus

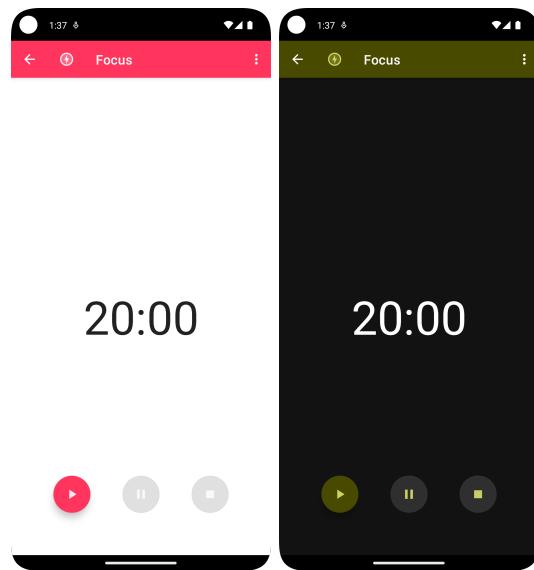


Figure 31: sezione Focus Light e Dark

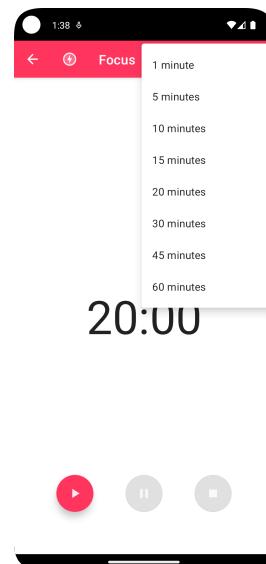


Figure 32: Impostazioni durata timer

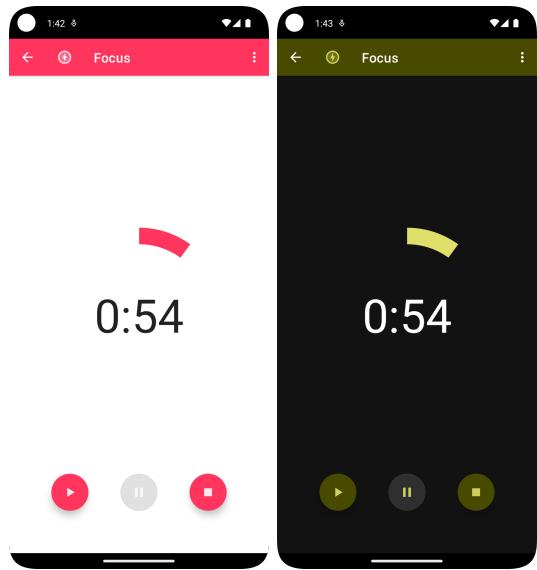


Figure 33: Timer in Pausa Light e Dark

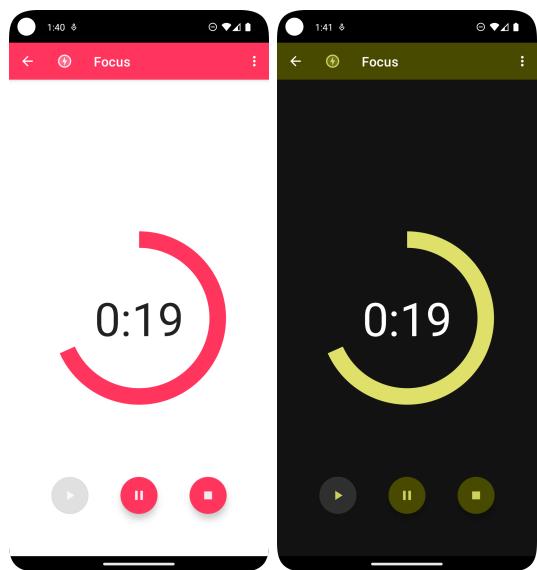


Figure 34: Timer Avviato Light e Dark

### 3.5.6 Sezione Tutorial

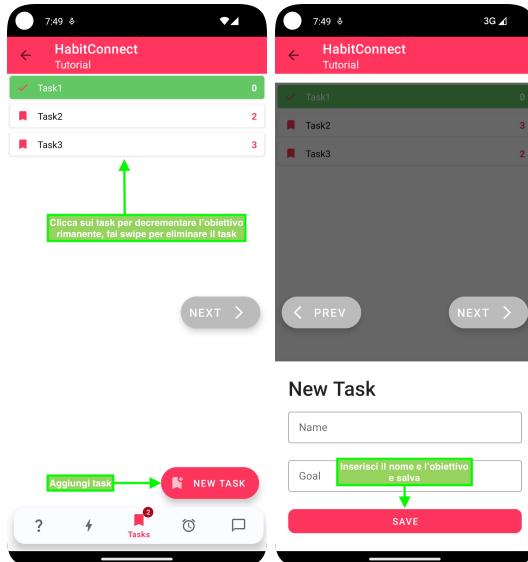


Figure 35: Tutorial 1 e 2

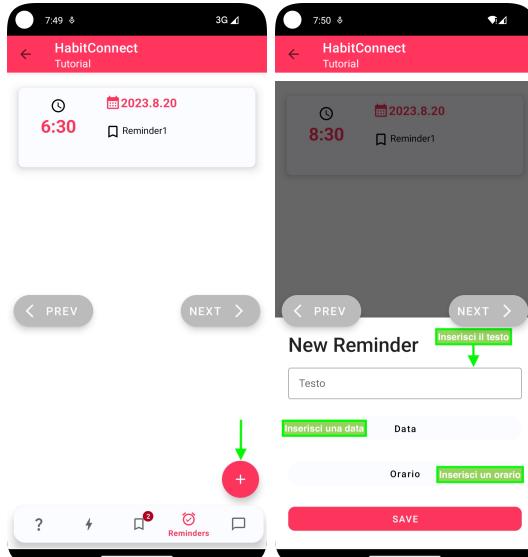


Figure 36: Tutorial 3 e 4

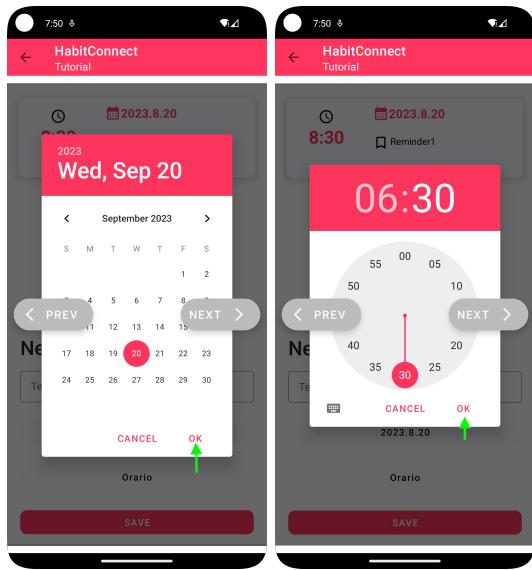


Figure 37: Tutorial 5 e 6

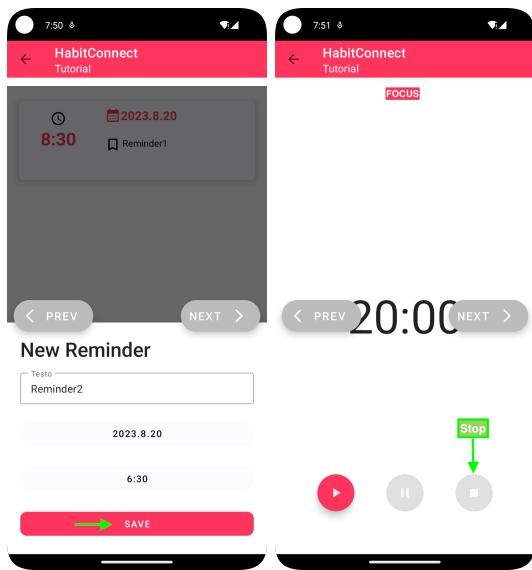


Figure 38: Tutorial 7 e 8

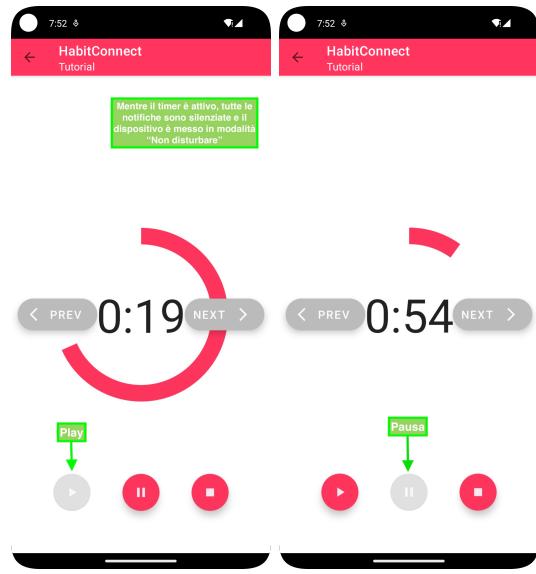


Figure 39: Tutorial 9 e 10

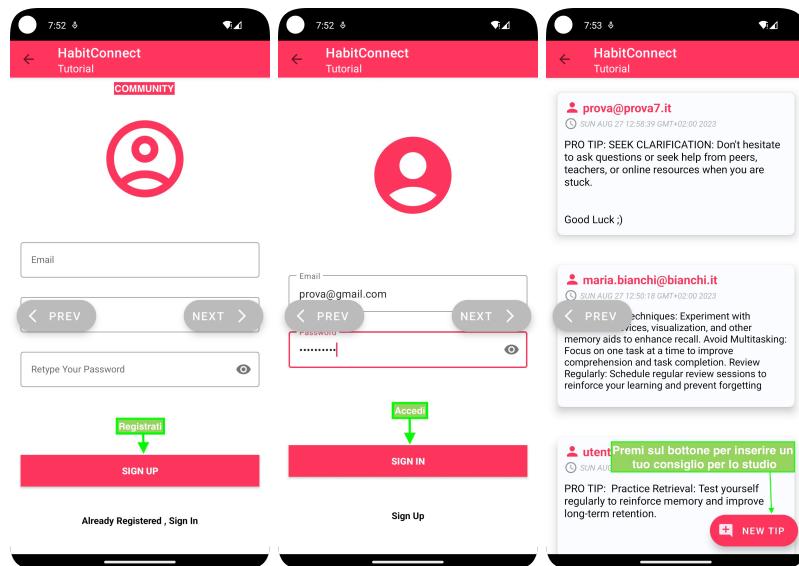


Figure 40: Tutorial 11, 12 e 13

## 3.6 Rappresentazione Grafica dei Casi d'Uso

### 3.6.1 Inserimento Nuovo Reminder

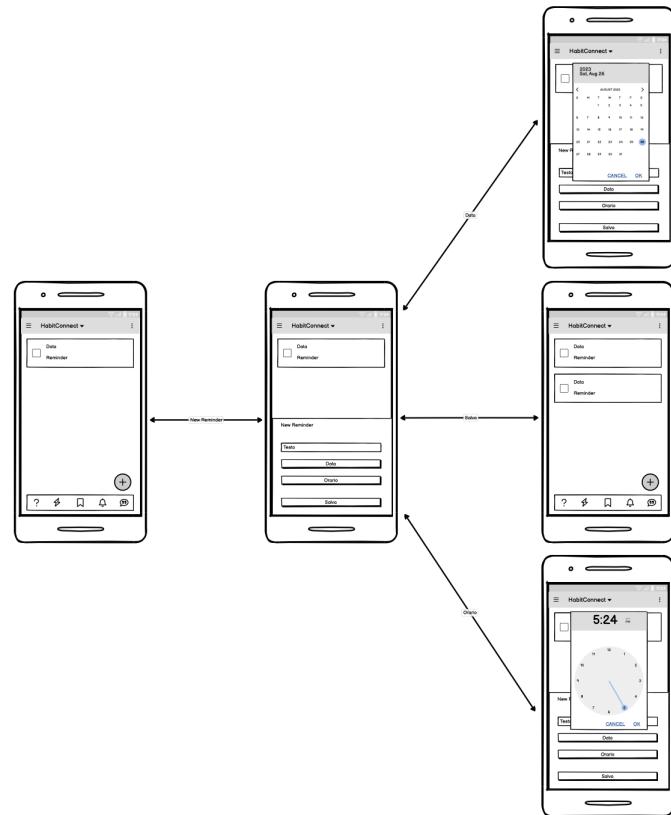


Figure 41: Nuovo Reminder UC

### 3.6.2 Inserimento Nuovo Task

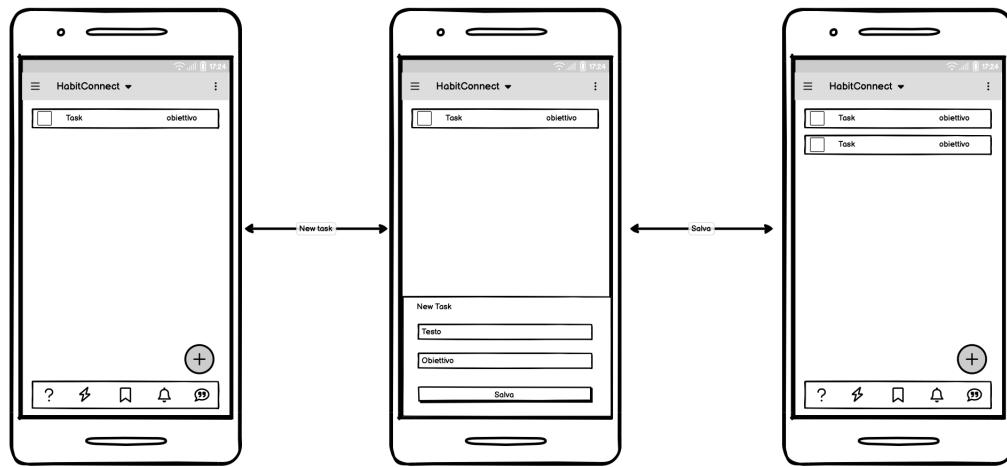


Figure 42: Nuovo Task UC

### 3.6.3 Accesso Community

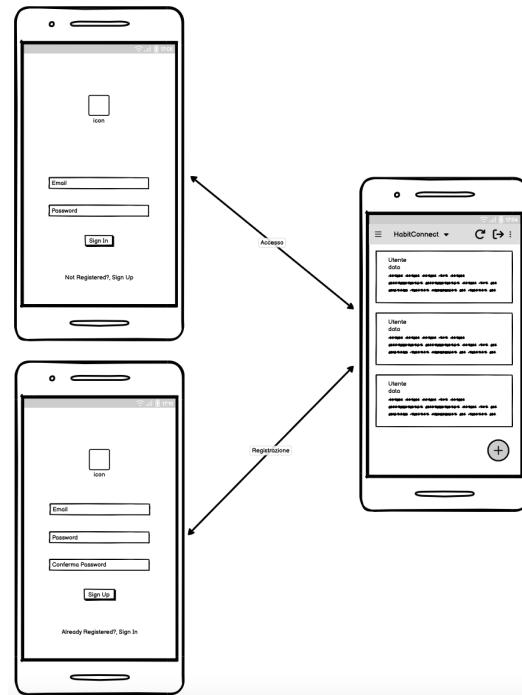


Figure 43: Accesso UC

### 3.6.4 Inserimento Nuovo Commento Community

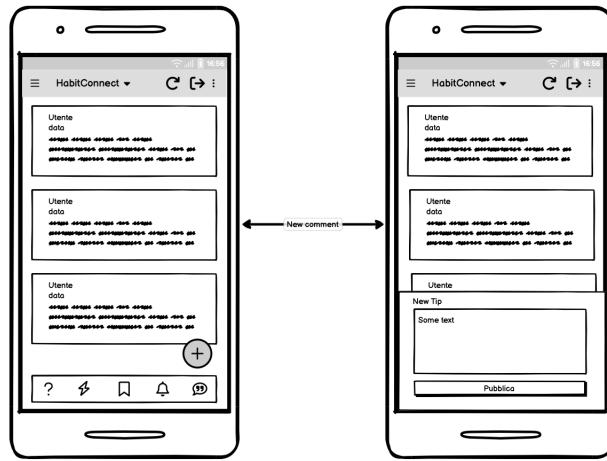


Figure 44: Nuovo Commento UC

### 3.6.5 Utilizzo Timer Focus

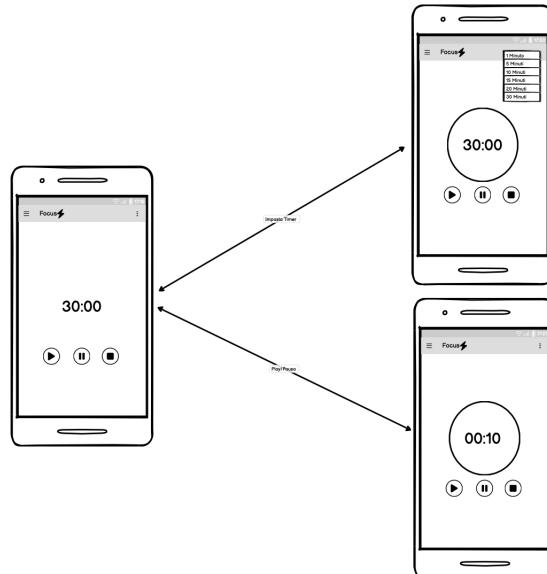


Figure 45: Timer Focus UC

## 3.7 Approccio e Problematiche di Sviluppo

### 3.7.1 Bug Risolti

- Una volta impostata una nuova durata partiva un nuovo timer oppure crashava l'app : Per risolverlo, nella funzione che inizializza il timer, si è fatto in modo che questo parta in ogni caso all'inizio (appena impostato o all'apertura della modalità). Così ogni item del menu di selezione della durata, esegue cancel() del timer attualmente istanziato, imposta una nuova durata, agendo (con la nuova key) sulle Preferences che definiscono la durata e infine inizializzano un nuovo timer.
- L'activity si creava prima di finire di scaricare i dati da Firebase : (visto usando logs) quindi alla prima apertura caricava sempre una recycler view vuota.  
Risolto usando LiveData e observe nella CommunityActivity. PERCHÈ IL FETCH DEI DATI È ASINCRONO, quindi serviva mettere un listener al completamento del fetch da firestore che restituisse la risposta, poi presa in carico dall'observer con observe, che quindi chiama, nell'adapter della recyclerview, il metodo notifyDataSetChanged().

### 3.7.2 Bug

La recyclerview dei tasks a volte modifica la UI in modo anomalo quando si fa lo scroll e ricarica gli items (testato nell'emulatore)

### 3.7.3 Librerie di Terze Parti

Utilizzata una libreria di terze parti per l'inserimento della Progress bar circolare che seguia il timer e non giri in continuazione, Android non la fornisce (solo una line Progress bar):

<https://github.com/zhanghai/MaterialProgressBar>

Dipendenze Gradle: implementation 'me.zhanghai.android.materialprogressbar:library:1.6.1'

## 3.8 Descrizione Componenti

### 3.8.1 MainActivity

```
private lateinit var bottomNavigationView: BottomNavigationView  
private lateinit var binding: ActivityMainBinding
```

**onCreate()**: inflate del layout e della bottom nav bar per navigare tra le sezioni, setta un click listener su ogni bottone che fa intent o Fragment transaction all'interno del frame layout. Viene controllato se savedInstanceState è null, ovvero se è la prima volta che l'activity viene creata, e in tal caso imposta come primo fragment nel FrameLayout della home il FragmentTask.kt. Alla creazione dell'activity aggiorno il badge della sezione task (che è la home di avvio).

**onResume()**: quando l'activity è nella fase di resume (era stato fatto un intent verso un'altra activity e si è tornati indietro) ricarica la main activity in modo che ci si trovi nella sezione Tasks con l'item Tasks nella bottom nav bar selezionato.

### 3.8.2 AppDatabase

Companion object col metodo che crea un istanza di db.

Classe di converters per convertire tipo Date in Long

### **3.8.3 TaskDao**

Interfaccia con metodi associate a Query SQL per l'interazione con l'entità Task

### **3.8.4 ReminderDao**

Interfaccia con metodi associate a Query SQL per l'interazione con l'entità Reminder

### **3.8.5 Task**

Modello dell'entità task

### **3.8.6 Reminder**

Modello dell'entità reminder

### **3.8.7 TASKS: FragmentTask**

```
class FragmentTask() : Fragment() , TaskAdapter.ClickListener
private lateinit var viewModel: FragmentTaskViewModel
private lateinit var adapter: TaskAdapter
private lateinit var recyclerView: RecyclerView
onCreateView(): inflate del layout, istanziazione della recyclerview, del TaskViewModel e
creazione di un observe in cui assegno il TaskAdapter alla recyclerview. Faccio l'observe dei tasks
del viewmodel, ogni volta che i tasks nel database cambiano, l'observer sarà notificato e i nuovi
task saranno aggiunti all'adapter (si mette ad osservare, setTasks poi invierà una notify). Listener
sul fab che apre la dialog per l'aggiunta di un task.
onItemClick(): click su item della recyclerview, decrementa l'obiettivo del task fino a zero e a
quel punto imposta completo=true.
onViewCreated(): inserisce una callback sugli item della recycler view, implementazione
dell'azione di swipe che elimina un task e possibilità di UNDO dell'operazione di eliminazione, poi
fa l'apply di tutto questo sulla recyclerview.
```

### **3.8.8 TaskAdapter**

Class TaskViewHolder: classe che assegna a due variabili due text view per testo e obiettivo.

**onCreateViewHolder()**: implementa TaskViewHolder, imposta una nuova variabile, faremo
linflate di qualcosa prendendo il context del parent e per ogni row fa linflate del layout task row
(è il layout per ogni row).

**onBindViewHolder()**: prende il task nella posizione passata dalla recyclerview e nel layout
assegna ai campi testo della recyclerview il testo e l'obiettivo, di seguito fa il controllo se il task è
completo, inserisce un onClickListener che esegue la funzione onItemClick che implemento nel
FragmentTask.

**getItemCount()**: dice alla recyclerview quanti records abbiamo

**setTasks()**: applica il metodo notifyDataSetChanged() per informare l'observer di una modifica.

**Interface ClickListener**: con una funzione onItemClick che prende come parametro un Task,
sarà implementato nel FragmentTask.

### 3.8.9 FragmentTaskViewModel

```
class FragmentTaskViewModel (application: Application) :  
    AndroidViewModel(application)  
  
taskDao: Variabile privata che prende un istanza di db e poi del TaskDao.  
Tasks : Variabile istanziata lazy che prende come valore il risultato della funzione getAllTasks()  
fun getAllTasks(): chiama getAllTasks() nel taskDao che ritorna una LiveData<List<Task>>; fun  
insertTask(entity: Task) : chiama insert(task) del dao per inserire un task nel db  
fun deleteTask(entity: Task) = viewModelScope.launch : elimina un task servendosi di  
un thread apposito, lanciando una coroutine.  
fun updateTask(entity: Task) : chiama metodo update(task) per aggiornare lo stato di un  
task  
fun getNonCompletati (): List<Task> : chiama metodo Dao per recuperare i task con  
completato=false
```

### 3.8.10 NewTaskSheet

```
class NewTaskSheet : BottomSheetDialogFragment()  
private lateinit var viewModel: FragmentTaskViewModel  
onCreateView() : inflate del layout della dialog per l'aggiunta di un task, mette un click  
listener al bottone per salvare.  
private fun saveAction(view: View) : funzione lanciata dal bottone di salvataggio, chiama  
metodo insert del viewmodel e poi fa il dismiss() della dialog.
```

### 3.8.11 TUTORIAL: TutorialActivity

```
class TutorialActivity : AppCompatActivity()  
private lateinit var binding: ActivityTutorialBinding  
private val listImage : viene inserito un array di interi con ciascun elemento che è un id di  
un'immagine da visualizzare in sequenza  
private var index = 0 : indice inizializzato a zero  
onCreate() : con il binding del layout ne fa linflate e setta limage switcher, poi mette dei  
listener nei bottoni di next e previous per incrementare e decrementare lindice e aggiornare  
l'immagine da visualizzare, inoltre gestisce la visibilità dei due bottoni
```

### 3.8.12 FOCUS: TimerActivity

```
class TimerActivity : AppCompatActivity()  
private lateinit var binding: ActivityTimerBinding  
private lateinit var notificationManager: NotificationManager  
private lateinit var audioManager: AudioManager
```

- proprietà del timer:

```
private lateinit var timer: CountDownTimer  
private var timerLengthSeconds: Long = 0  
private var timerState = TimerState.Stopped  
private var secondsRemaining: Long = 0
```

```

enum class TimerState : definisce gli stati del timer: Stopped, Paused, Running

OnCreate() : binding e inflating di layout e della toolbar, setta click listener sui fab di stop, play, e pause che chiamano funzioni per modificare lo stato del timer e aggiornare la UI
onResume() : chiama initTimer()
onPause() : nelle Preferences imposta il tempo rimanente
private fun initTimer() inizializza il timer con lo stato e il tempo salvati nelle preferences se lo stato è stopped lo imposta con la durata impostata la volta prima, salvata nelle preferences, altrimenti imposta la durata che mancava dalla scorsa volta, anch'essa salvata nelle preferences. Il timer inizia comunque per evitare problemi con crash cliccando sul menu per la selezione temporale.
private fun onTimerFinished() : funzione per quando si clicca sul pulsante di stop (stato di stopped). Imposta la lunghezza del timer con quella impostata in SettingsActivity se la durata è cambiata nel mentre era in stato di running, update della UI dei bottoni e del countdown.
private fun startTimer() : agganciata al pulsante di start, setta lo stato a Running, crea un oggetto di tipo CountDownTimer e fa l'override funzione onTick() così ad ogni tick del timer, aggiorna la UI; e l'override di onFinish() che così chiama la funzione onTimerFinished().
private fun setNewTimerLength() : funzione per settare la durata del timer
private fun setPreviousTimerLength() : imposta la durata "precedente" ovvero quella già salvata in precedenza nelle preferences
private fun updateCountdownUI() : aggiorna UI timer quando è in running, la stringa dei secondi è gestita in modo che visualizzi uno 0 davanti se il numero è un'unità.
private fun updateButtons() : aggiorna i bottini a seconda dello stato in cui il timer si trova al momento della chiamata abilita/disabilita i fab con conseguente modifica automatica della UI. In Running viene settato il non disturbare e la modalità silenziosa, in Stopped tutte le notifiche di nuovo attive (nessun filtro), in Paused tutte le notifiche di nuovo attive (nessun filtro).
override fun onCreateOptionsMenu(menu: Menu): Boolean : Inflate del menu; aggiunge items all'action bar se presente.
override fun onOptionsItemSelected(item: MenuItem): Boolean : gestisce la selezione dell'item nel menu, dopo il click su un item: cancella l'esecuzione attuale come se fosse uno stop, imposta una nuova durata nelle preferences, imposta lo stato di Stopped, inizializza un nuovo timer con la nuova durata (che partì in automatico)
override fun onDestroy() : chiamato setTimerState per sicurezza, perché un utente può uscire dal focus mentre il timer è in stato Running, quindi viene sempre rimesso in stato paused. Inoltre per sicurezza vengono rimesse tutte le suonerie e disattivato do not disturb
override fun onRestart() : l'utente quando è dentro focus, può andare nelle impostazioni e togliere la permission per il do not disturb. Quindi al restart dell'activity viene rifatto il controllo

```

### 3.8.13 PrefUtil

```

class PrefUtil : chiamata come companion object
private const val TIMER_LENGTH_ID = "com.example.timer.timer_length" : key per recuperare la durata del timer nelle preferences
private const val PREVIOUS_TIMER_LENGTH_SECONDS_ID =
"com.example.timer.previous_timer_length_seconds" : key usata per recuperare il dato con la durata già salvata nelle preferences

```

```

private const val TIMER_STATE_ID = "com.example.timer.timer_state" : key usata per gestire lo
stato del timer nelle preferences
private const val SECONDS_REMAINING_ID = "com.example.timer.seconds_remaining" : key
per recuperare i secondi rimanenti dalle preferences
fun getTimerLength(context: Context): Int : ritorna la durata del timer usando la key
fun setTimerLength(minutes: Int = 30, context: Context) : setta la durata del timer con
l'input passato come parametro, per il quale possiede un valore di default di 30.
fun getPreviousTimerLengthSeconds(context: Context): Long : recupera dalle
preferences la durata salvata in precedenza
fun setPreviousTimerLengthSeconds(seconds: Long, context: Context) : imposta la
durata salvata in precedenza nelle preferences
fun getTimerState(context: Context): TimerActivity.TimerState : recupera lo stato
attuale del timer
fun setTimerState(state: TimerActivity.TimerState, context: Context) : imposta lo
stato del timer andando a scrivere nelle preferences, passo la variabile di tipo TimerState che è
una enum class definita nella TimerActivity
fun getSecondsRemaining(context: Context): Long : recupera I secondi rimanenti di una
run precedentemente messa in pausa
fun setSecondsRemaining(seconds: Long, context: Context) : setta i secondi rimanenti
quando viene messo in pausa

```

### 3.8.14 REMINDERS: FragmentReminders

```

class FragmentReminders: Fragment(), ReminderAdapter.ClickListener
private lateinit var viewModel: FragmentRemindersViewModel
private lateinit var adapter: ReminderAdapter
private lateinit var recyclerView: RecyclerView
onCreateView(): inflate del layout, inizializzazione recyclerview, inizializzazione viewmodel,
inserimento di un observer in cui viene assegnato un adapter alla recyclerview e viene chiamato il
suo metodo setReminders() per chiamare il metodo di notifica di una modifica. Inserito click
listener al fab per mostrare la bottom sheet dialog per aggiungere un nuovo reminder. E ritorna la
view con cui fa linflate del fragment layout nel layout dellactivity, dentro il FrameLayout.
override fun onItemClicked(reminder: Reminder) : funzione per il click su un item della
recyclerview che mostra i dettagli del testo. Specifica un listener che permette di visualizzare il
testo di una notifica. La dialog viene automaticamente chiusa quando si clicca sul bottone OK, un
listener null permette al bottone di chiudere la dialog senza azioni.
onViewCreated() : inserisce una callback sugli item della recycler view, implementazione
dellazione di swipe che elimina un reminder con conseguente messaggio Snackbar, poi fa lapply
di tutto questo sulla recyclerview.

```

### 3.8.15 NewReminderSheet

```

class NewReminderSheet : BottomSheetDialogFragment()
private lateinit var viewModel: FragmentRemindersViewModel
private fun openDatePicker(view: View) : funzione che apre il date picker e setta il testo
nel bottone della data con la data selezionata

```

**private fun openTimePicker(view: View)** : funzione che apre il time picker e setta il testo nel bottone dell'orario con l'orario selezionato  
**private fun saveAction(view: View)** : vengono fatti controlli al momento del salvataggio, se il testo non è stato inserito, viene lasciato quello di default, se data e ora sono state inserite allora procede con l'inserimento nel db, altrimenti mostra un Toast richiedendole.

### 3.8.16 ReminderAdapter

Classe che crea il layout dell'item di recyclerview e poi fa il binding.

**fun setReminders(reminders: List Reminder)** : invia una notifica di modifica dei dati all'observer, e assegna la lista di reminders passata alla sua variabile privata.  
**interface ClickListener : fun onItemClick(reminder : Reminder)** : funzione poi implementata per gestire il click sull'item.

### 3.8.17 FragmentRemindersViewModel

Classe ViewModel che prende un'istanza di db e chiama i metodi del reminderDao

```
private val _reminderDao = AppDatabase.getInstance(application).reminderDao()
val reminders by lazy {_reminderDao.getAllReminders() }
```

### 3.8.18 COMMUNITY: CommunityActivity

```
private lateinit var binding : ActivityCommunityBinding
private lateinit var firebaseAuth: FirebaseAuth
private lateinit var adapter: CommentAdapter
private lateinit var recyclerView: RecyclerView
private lateinit var viewModel: ActivityCommunityViewModel
```

**onCreate()** : binding del layout, aggancia un adapter alla recyclerview. Scarica tutti i commenti e li salva su una mutable list, gestito con un observer che quando riceve risposta al completamento del download, setta l'adapter, lo assegna alla recyclerview e notifica il cambiamento. Listener sul fab che apre la bottom sheet dialog per inserire un nuovo commento.

**getResponseUsingLiveData()** : Richiede tutti i commenti al server gestendo la lista di risposta come live data, agganciando un observer alla funzione di get del viewmodel, questa ha un listener al completamento del fetch dei dati (ASINCRONO) che ritorna la Response dal server.

**onCreateOptionsMenu(menu: Menu): Boolean** : funzione che fa linflate del menu della community, che prevede due icone nella nav bar, una per il sign out e l'altra per ricaricare la pagina (fa una nuova richiesta al server).

**onOptionsItemSelected(item: MenuItem): Boolean** : Assegna le funzioni agli item di menu, l'icona di refresh esegue la recreate() dell'activity (quindi effettua nuovamente una richiesta), l'icona del sign out esegue signOut() in firebase Auth e fa un intent verso la pagina di accesso.

### 3.8.19 NewCommentSheet

```
private lateinit var viewModel: ActivityCommunityViewModel
```

**private fun publish(view: View)** : prende come parametro la view del layout, così recupera il riferimento al campo di EditText dal quale estrae il testo che poi passa come parametro al metodo

`insert()` del view model per l'inserimento di un commento (utente e timestamp sono generati in automatico dentro il metodo `insert`).

### 3.8.20 Comment

Data class che definisce la struttura di un commento:

```
var user : String? = null,  
var timestamp: Timestamp? = null,  
var testo: String? = null,
```

### 3.8.21 Response

Data class che mappa la struttura della risposta inviata da firebase: il prodotto di una richiesta o un'eccezione

```
var products: List<Comment>? = null,  
var exception: Exception? = null
```

### 3.8.22 Checker

Definisce un companion object con una variabile EMAIL\_ADDRESS dove viene inizializzato il pattern che devono avere le email usate per accedere.

```
fun isValidEmail(email: String): Boolean : controlla il match tra l'email inserita e il pattern  
fun passwordCheck(password: String, confirmPassword: String): Boolean : controlla  
che la password e la conferma concidano  
fun passwordLengthCheck(password: String): Boolean : controlla che la lunghezza della  
password sia almeno di 6 caratteri, perchè altrimenti firebase lancerebbe un'eccezione
```

### 3.8.23 ActivityCommunityViewModel

```
private lateinit var firebaseAuth: FirebaseAuth  
var listaCommenti:MutableList<Comment> = mutableListOf() : Crea istanza di FirebaseAuth e  
una lista vuota di Comment che poi andrà a riempire.  
fun getResponseAllComments(): MutableLiveData<Response> : Prende un'istanza di  
Firestore e fa una get per ottenere tutti i documenti nella collection 'commenti' ordinati per  
timestamp. Mette un listener in caso di successo che mappa ciascun documento della risposta in  
un oggetto di tipo Comment e poi lo inserisce nella listaCommenti.  
Listener di errore.  
Listener al completamento: se l'operazione ha avuto successo, inserisce i documenti ricevuti nella  
variabile products dell'oggetto Response mappandoli come oggetti Comment, infine questo viene  
inserito in un MutableLiveData Response e restituito.  
fun insertComment(testo:String) : crea un'istanza di FirebaseAuth per recuperare lo  
username dell'utente registrato per inserirlo come parametro User nel documento che rappresenta  
il commento. Il timestamp viene definito a livello server. Viene preso come parametro solo il testo  
del commento. Infine viene salvato su Firestore con feedback di un Toast.
```

### 3.8.24 CommentAdapter

Adapter per i singoli commenti nella recyclerview della community

```
class CommentViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) :  
    assegnamento dei valori del singolo item al layout di row di cui ho fatto linflate  
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):  
        CommentViewHolder : inflate del layout di row  
    override fun onBindViewHolder(holder: CommentViewHolder, position: Int) : bind  
        degli item della recyclerview (con le loro posizioni in lista) con un holder dell'elemento. Il  
        timestamp viene prima convertito in dateFormat e poi in String per assegnarlo all'attributo text  
        di una TextView.  
    override fun getItemCount() : restituisce la lunghezza della lista di commenti  
    fun setComments() : chiama il metodo notifyDataSetChanged() che serve a notificare un  
        observer di un certo cambiamento nel set di dati che sta osservando.
```

### 3.8.25 SignInActivity

```
private lateinit var binding: ActivitySignInBinding  
private lateinit var firebaseAuth: FirebaseAuth  
onCreate() : binding del layout e inflate della toolbar, setta listener sui bottoni e esegue  
controlli sulle credenziali inserite, poi prende la risposta del server e in base al successo o meno  
dell'operazione invia un feedback.  
onStart() : allo start dell'activity controlla se l'utente ha già fatto l'accesso e ha una sessione  
attiva, in tal caso fa un intent direttamente nella CommunityActivity.
```

### 3.8.26 SignUpActivity

```
private lateinit var binding: ActivitySignUpBinding  
private lateinit var firebaseAuth: FirebaseAuth  
onCreate() : nel listener sul bottone di sign up fa i controlli sulle credenziali: inizialmente  
verifica che i campi non siano nulli, poi utilizza la Classe Checker.kt per il controllo del pattern  
email, della lunghezza della password e della coincidenza tra la password e la riconferma. Infine se  
ha successo, fa intent verso SignIn (che poi redirezionerà l'utente registrato direttamente nella  
pagina di Community), altrimenti mostra l'eccezione del server.
```

## 3.9 Testing

### 3.9.1 DatabaseTest

**FIX :** tolta db.close() da @After perchè veniva eseguito sempre dopo un singolo test,  
quindi se si avviava il test per tutta la classe di test, solo il primo riceveva l'istanza di  
db

1. writeAndReadReminder() : test di scrittura e lettura reminder
2. writeAndReadTask() : test di scrittura e lettura task
3. After : reset() : cancella le istanze di test, controllando quali istanze di test sono state  
create (se reminder o task)

### **3.9.2 MainActivityTest**

Test UI per vedere se carica il layout correttamente.

### **3.9.3 CheckerTest**

Unit test delle funzioni della classe Checker nel package util:

1. test funzione di controllo del pattern email,
2. test funzione controllo lunghezza password,
3. test funzione che controlla l'uguaglianza tra la password e il suo retype.

### **3.9.4 PrefUtilTest**

Unit test della funzione che ritorna la durata del timer

### **3.9.5 TutorialActivityTest**

Instrumented test per:

- Fare il check che al primo click sul fab next, sia visibile il fab prev e al successivo click su prev questo torna invisibile,
- 12 click su next: verifica che next sia invisibile a seguito del dodicesimo click (ultima slide di tutorial), poi clicca su prev e verifica che next sia visibile (ritorna nella penultima slide).

### **3.9.6 TimerActivityTest**

Instrumented tests per verificare che:

- Al click su stop i bottoni start, pause, stop si abilitino e disabilitino correttamente,
- Al click su stop e poi start i bottoni start, pause, stop si abilitino e disabilitino correttamente,
- Al click su pausa i bottoni start, pause, stop si abilitino e disabilitino correttamente.

## 4 Flutter App

### 4.1 Introduzione

L'applicazione flutter presenta le stesse funzionalità di quella kotlin, escluse le sezioni di reminders e focus.

### 4.2 Analisi dei Requisiti

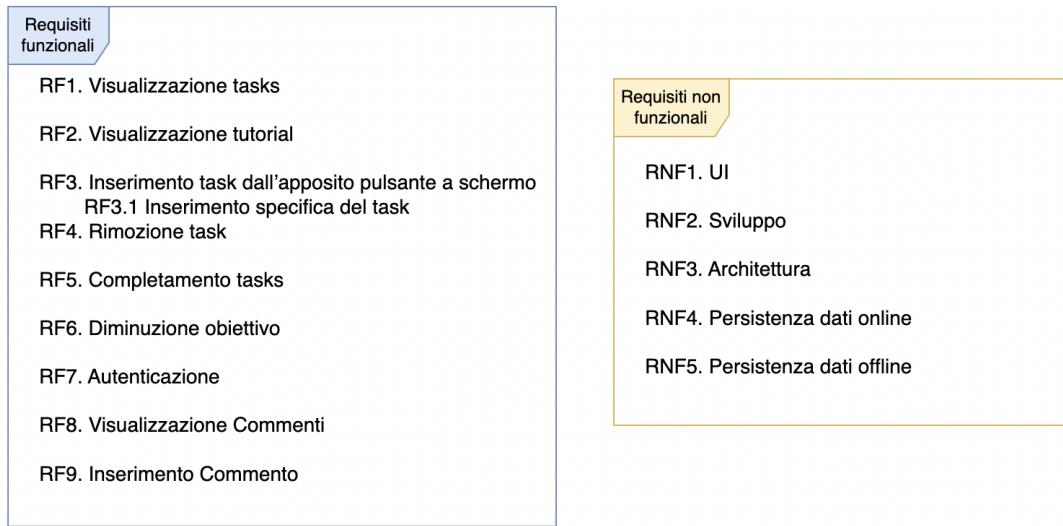


Figure 46: Requisiti Flutter

#### **4.2.1 REQUISITI FUNZIONALI**

- RF1. Visualizzazione tasks** : L'utente deve poter visualizzare i propri tasks nella home
- RF2. Visualizzazione tutorial** : L'app deve fornire all'utente informazioni sulle modalità di utilizzo delle proprie funzionalità
- RF3. Inserimento task dall'apposito pulsante a schermo** : L'utente deve poter inserire un nuovo task
- RF3.1 Inserimento specifica del task** : L'utente deve poter specificare i parametri del task inserendo manualmente il testo e l'obiettivo
- RF4. Rimozione task** : L'utente deve poter rimuovere un task con uno swipe e il click sull'apposita icona
- RF5. Completamento tasks** : L'utente deve poter visualizzare i tasks completati
- RF6. Diminuzione obiettivo** : L'utente deve poter diminuire l'obiettivo presente su ciascun task
- RF7. Autenticazione** : L'utente deve poter autenticarsi al momento dell'accesso nella sezione community
- RF8. Visualizzazione Commenti** : L'utente, una volta fatto l'accesso, deve poter visualizzare i commenti degli utenti
- RF9. Inserimento Commento** : L'utente, una volta fatto l'accesso, deve poter inserire un suo commento

#### **4.2.2 REQUISITI NON FUNZIONALI**

- RNF1. UI** : L'interfaccia deve essere semplice ed intuitiva per migliorare la user experience
- RNF2. Sviluppo** : Il linguaggio di sviluppo è Dart con framework Flutter
- RNF3. Architettura** : Le varie classi di schermate devono essere definite in file dart separati
- RNF4. Persistenza dati online** : L'attività di community è gestita con server Firebase
- RNF5. Persistenza dati offline** : La gestione di task è eseguita offline con l'ausilio di Hive, un db noSQL che gestisce delle box nei quali si definiscono gli attributi degli elementi che queste contengono.

### 4.3 Casi d'Uso



Figure 47: Casi d'Uso

#### 4.3.1 UC: Gestione Tasks

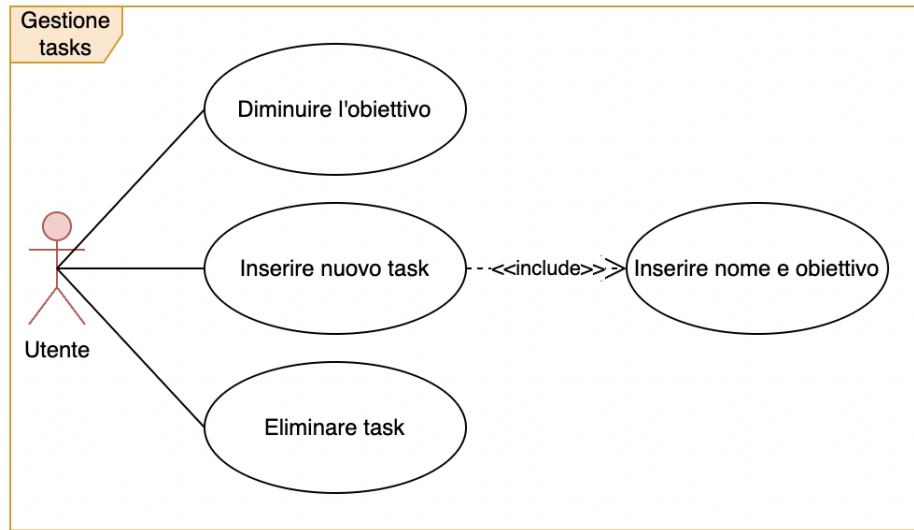


Figure 48: Gestione Tasks

Questi casi d'uso riguardano l'area di funzionalità per la gestione dei tasks

##### Casi d'uso: Inserire nuovo task

Questo caso d'uso si verifica qualora l'utente volesse inserire un nuovo task

**Pre-condizioni:** Nessuna

**Post-condizioni:** Inserimento del task nel sistema

##### Sequenza di eventi principali:

Il caso d'uso inizia quando l'utente decide di inserire un nuovo task nella lista di obiettivi

1. include(Inserire nome e obiettivo)
2. Il sistema visualizza nella home il nuovo task

##### Casi d'uso: Inserire nome e obiettivo

Questo caso d'uso si verifica durante l'inserimento del nuovo obiettivo a sistema

**Pre-condizioni:** L'utente deve aver iniziato la procedura di inserimento

**Post-condizioni:** Inserimento dei dettagli del task

##### Sequenza di eventi principali:

Il caso d'uso inizia quando l'utente decide il nome e i dettagli dell'obiettivo

1. Il sistema permette l'inserimento del nome e del goal
2. L'utente conferma
3. Il sistema salva i dati

### **Sequenza degli eventi alternativi**

La sequenza alternativa inizia al punto 2

1. L'utente annulla l'operazione cliccando fuori dalla dialog o tornando indietro
2. Il sistema torna alla home

### **Casi d'uso: Eliminare task**

Questo caso d'uso si verifica quando l'utente vuole eliminare un task dalla lista

**Pre-condizioni:** Il task deve essere già registrato a sistema

**Post-condizioni:** Il task non è più presente a sistema

### **Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di rimuovere un task

1. L'utente fa uno swipe sul task a destra o a sinistra e poi clicca sull'apposita icona
2. Il sistema rimuove dalla memoria il task

### **Casi d'uso: Diminuire obiettivo**

Questo caso d'uso si verifica quando l'utente clicca su un task nella lista

**Pre-condizioni:** Il task deve essere già registrato a sistema

**Post-condizioni:** L'obiettivo del task diminuisce di uno

### **Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di ridurre l'obiettivo di un task

1. L'utente preme sul task nella lista
2. Il sistema riduce il numero che indica l'obiettivo

#### 4.3.2 UC: Gestione Community

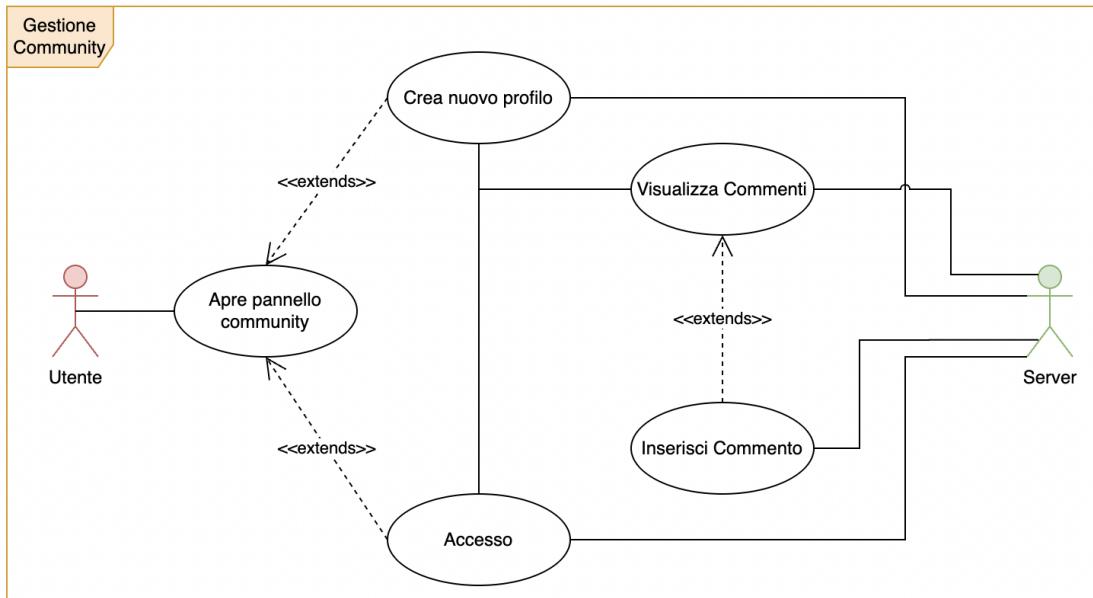


Figure 49: Gestione Community

Questi casi d'uso riguardano l'area di funzionalità dedicate alla community

##### Casi d'uso: Aprire pannello community

Questo caso d'uso si verifica quando l'utente preme sull'icona di Community

**Pre-condizioni:** Nessuna

**Post-condizioni:** Nessuna

##### Sequenza di eventi principali:

Il caso d'uso inizia quando l'utente decide di entrare nell'area community

1. L'utente entra nella schermata di community
2. include(Accesso)

##### Sequenza degli eventi alternativi

La sequenza degli eventi alternativi si verifica se l'utente non è già registrato

*Inizia dal punto 2*

1. include(Crea nuovo profilo)
2. Il sistema carica i dati dal server

##### Casi d'uso: Accesso

Questo caso d'uso si verifica quando l'utente vuole accedere alla community e possiede un profilo

**Pre-condizioni:** Profilo già registrato

**Post-condizioni:** Nessuna

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente entra nella schermata di accesso

1. Il sistema permette di inserire le credenziali
2. L'utente accede

**Sequenza degli eventi alternativi**

La sequenza degli eventi alternativi inizia al punto 2

1. L'utente inserisce credenziali errate
2. Il sistema chiede nuovamente i dati di accesso

**Casi d'uso: Crea nuovo profilo**

Questo caso d'uso si verifica quando l'utente vuole registrarsi nel sistema

**Pre-condizioni:** Nessuna

**Post-condizioni:** Profilo registrato nel server

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente decide di registrarsi

1. L'utente preme il pulsante per la registrazione
2. Il sistema permette di inserire delle credenziali per l'accesso
3. L'utente accede

**Casi d'uso: Visualizza commenti**

Questo caso d'uso si verifica quando l'utente entra nell'area community per visualizzare i commenti degli utenti

**Pre-condizioni:** Accesso effettuato

**Post-condizioni:** Nessuna

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente termina le fasi di accesso

1. Il sistema carica i commenti dal server

**Casi d'uso: Inserisci commento**

Questo caso d'uso si verifica quando l'utente entra nell'area community per inserire un commento

**Pre-condizioni:** Accesso effettuato

**Post-condizioni:** Commento inserito nel sistema

**Sequenza di eventi principali:**

Il caso d'uso inizia quando l'utente termina le fasi di accesso

1. L'utente preme sull'icona apposita
2. Il sistema permette di inserire un testo
3. L'utente conferma
4. Il commento viene registrato a sistema

#### 4.3.3 UC: Gestione Info (Tutorial)

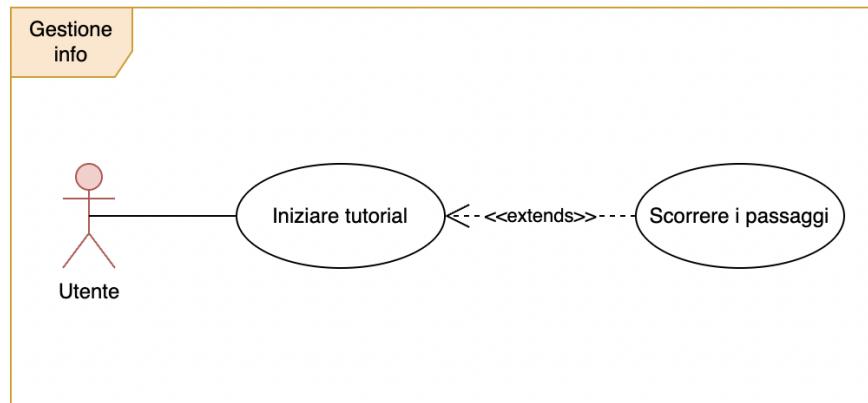


Figure 50: Gestione Info Tutorial

Questi casi d'uso riguardano le funzionalità di tutorial per spiegare il funzionamento dell'app

##### Casi d'uso: Iniziare tutorial

Questo caso d'uso si verifica quando l'utente vuole visualizzare le specifiche di funzionamento della UI

**Pre-condizioni:** Nessuna

**Post-condizioni:** Nessuna

##### Sequenza di eventi principali:

Il caso d'uso inizia quando l'utente termina le fasi di accesso

1. L'utente preme sull'apposita icona
2. Il sistema mostra la sequenza di immagini del tutorial
3. include (Scorre i passaggi)

##### Casi d'uso: Scorre i passaggi

Questo caso d'uso si verifica quando l'utente è dentro il tutorial

**Pre-condizioni:** Inizio tutorial

**Post-condizioni:** Nessuna

##### Sequenza di eventi principali:

Il caso d'uso inizia quando l'utente entra nel tutorial

1. L'utente scorre le fasi di tutorial avanti e indietro
2. L'utente termina il tutorial

## 4.4 Mappa dell'Architettura

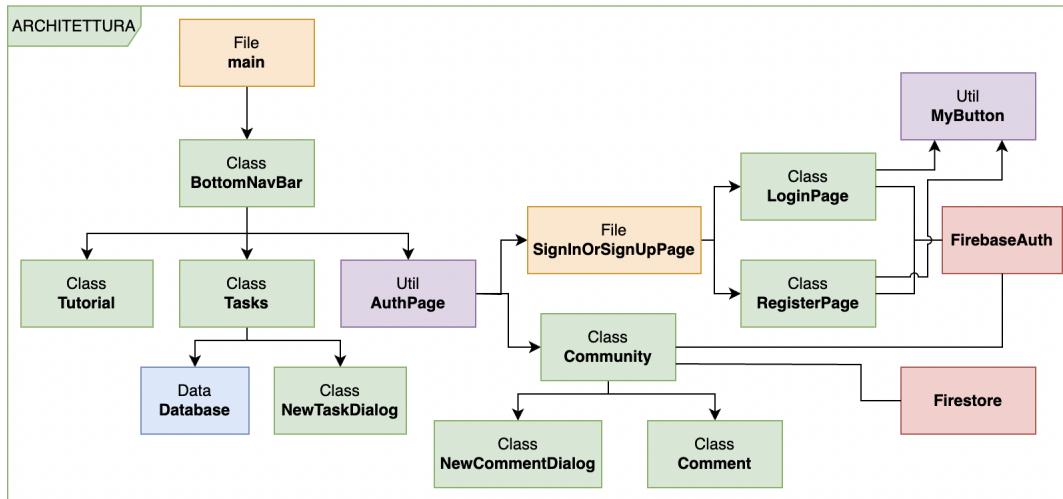


Figure 51: Mappa dell'Architettura

## 4.5 Database

Per salvare i dati riguardanti i task è stato utilizzato *Hive*, un database noSQL basato su key-value, perfetto per salvare dati di piccole dimensioni localmente. È stata creata una box 'taskbox' e per accedere ai suoi valori una key 'TASKLIST'. Ciascun elemento possiede 3 attributi: testo, obiettivo e completato. Prima si prende un riferimento alla box, poi si prendono i dati dentro con la key 'TASKLIST' e si salvano in una lista, accedo al singolo elemento con un indice, e con un ulteriore indice si accede al campo specifico (0=testo, 1=obiettivo, 2=completato).

### 4.5.1 Firebase

L'utilizzo di firebase è lo stesso dell'app kotlin.

## 4.6 Mockup UI

### 4.6.1 Home

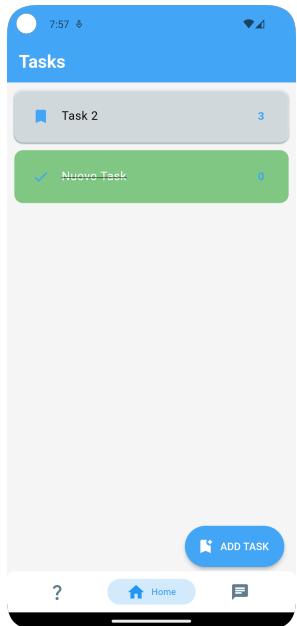
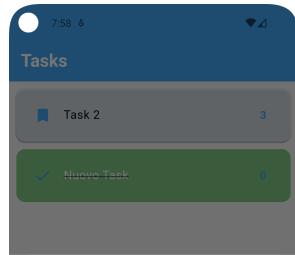


Figure 52: Home (tasks)



New Task

Task

Obiettivo

X ✓

?

Home

?

ADD TASK

Figure 53: Nuovo Task

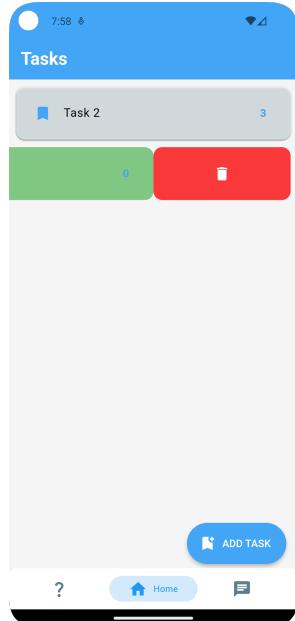


Figure 54: Elimina Task

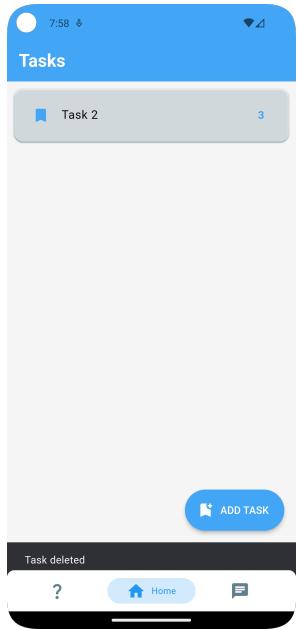


Figure 55: Elimina Task Feedback

#### 4.6.2 Community

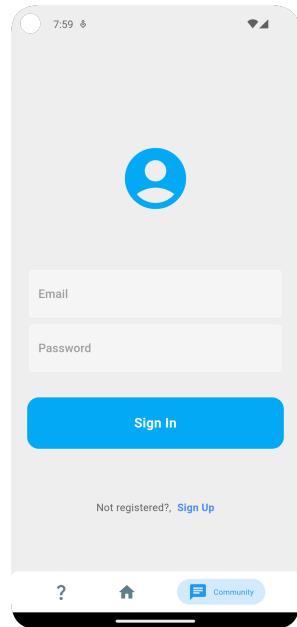


Figure 56: Accesso

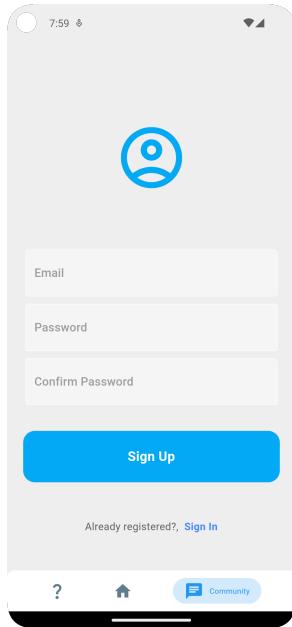
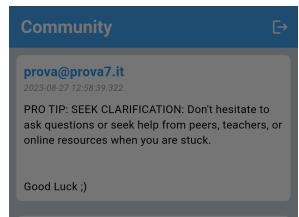


Figure 57: Registrazione



Figure 58: Community Home Page



New Tip

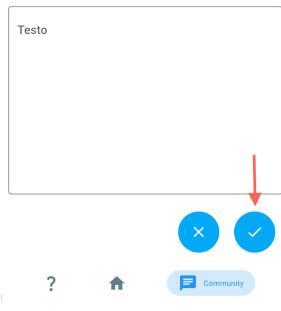


Figure 59: Nuovo Commento

#### 4.6.3 Tutorial

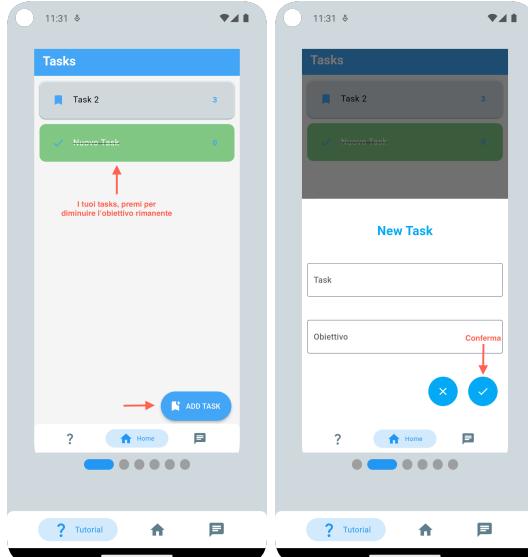


Figure 60: Tutorial 1 e 2

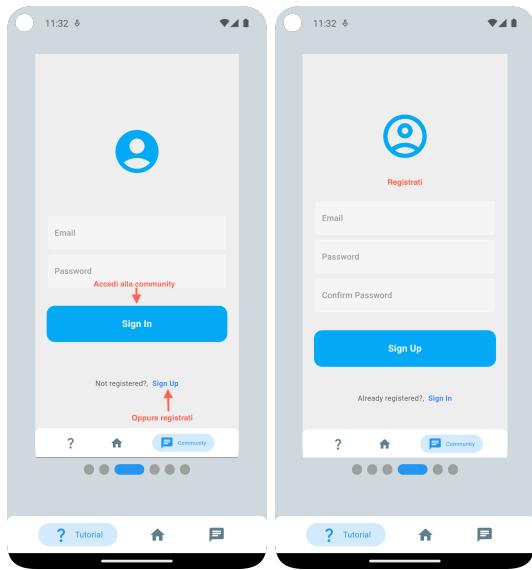


Figure 61: Tutorial 3 e 4

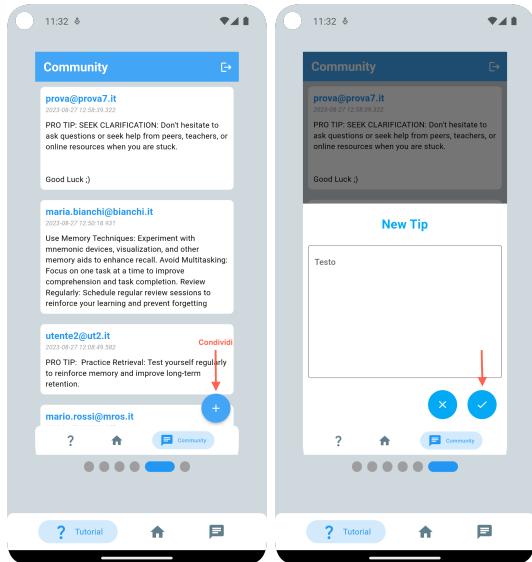


Figure 62: Tutorial 5 e 6

## 4.7 Rappresentazione Grafica dei Casi d'Uso

### 4.7.1 Inserimento Nuovo Task

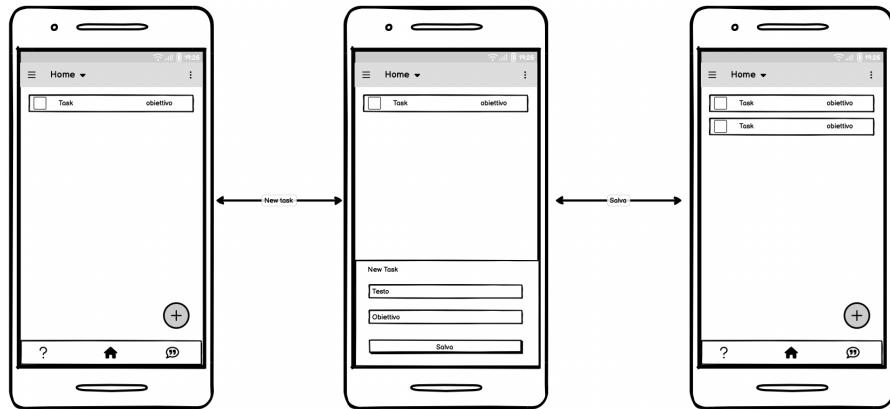


Figure 63: Nuovo Task UC

### 4.7.2 Accesso Community

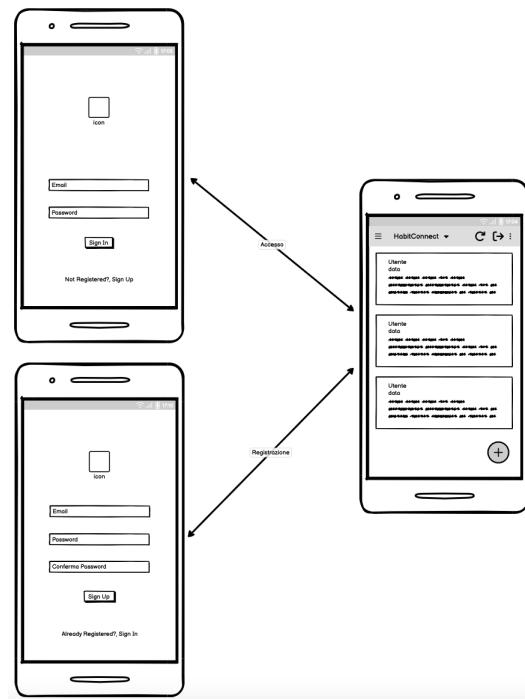


Figure 64: Accesso UC

#### 4.7.3 Inserimento Nuovo Commento Community

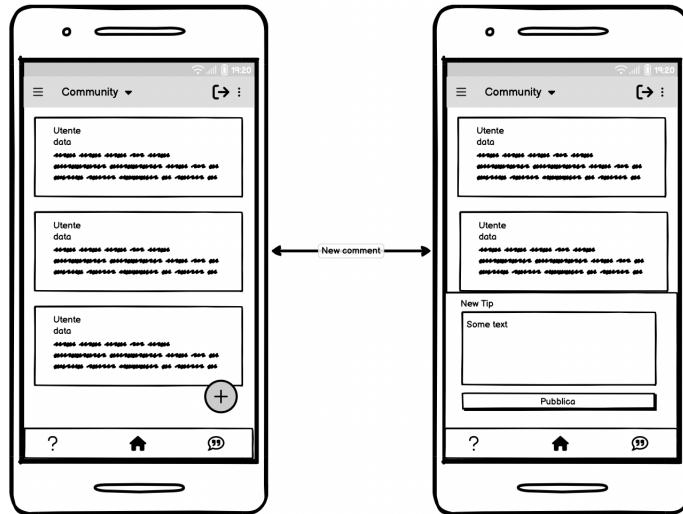


Figure 65: Nuovo Commento UC

### 4.8 Commenti sull'Approccio di Sviluppo

#### 4.8.1 Utilizzo Immagini Tutorial

Modifica nel pubspec in assets: tolto il commento e messo /images, così tutti gli elementi dentro la cartella vengono presi come asset.

#### 4.8.2 Componenti Importate

**Tutorial carousel:**

COMANDO: flutter pub add carousel slider

CHE HA IMPORTATO AUTOMATICAMENTE: dependencies: carousel\_slider: 4.1.1

**Per l'indicatore della slide del tutorial:**

COMANDO: flutter pub add smooth\_page\_indicator

CHE HA IMPORTATO AUTOMATICAMENTE: dependencies: smooth\_page\_indicator: 1.0.0+2

### 4.9 Descrizione delle Componenti

#### 4.9.1 Main

**main()** : funzione che apre la box 'taskbox' di Hive e inizializza Firebase, entrambe in modo asincrono.

**class MyApp extends StatelessWidget**

Classe che ritorna la MaterialApp dove come home è indicata la BottomNavBar().

**class BottomNavBar extends StatelessWidget**

Ha come variabile una lista di widget che rappresentano le schermate, da assegnare poi agli item della bottom nav bar.

Ritorna una bottom nav bar persistente che ad ogni indice dei suoi elementi associa una schermata, con uno screen controller persistente come la bottom nav bar, impostando anche quella iniziale (schermata tasks).

#### 4.9.2 TASKS : Tasks

```
class Tasks extends StatefulWidget
```

Nello stato prendo un riferimento alla box di Hive e chiamo la funzione per caricare i dati in una lista. Il metodo build ritorna la lista con gli item ListTile e associa uno slide gesture per eliminare i task.

```
final _controller1 = TextEditingController();  
final _controller2 = TextEditingController();
```

**void goalChanged(int index)** : funzione chiamata quando è stato cliccato l'elemento, passa l'indice per prenderlo nella lista. Se l'obiettivo non è a zero, lo decrementa, se è a zero mette a true il campo che indica 'completo'. Infine fa l'update del db.

**void saveNewTask()** : funzione chiamata per salvare nel db un nuovo task, ripulisce i controller dai dati precedentemente inseriti nei campi di input (perchè fanno parte di una classe stateful).

Prevede anche dei parametri di default in caso non venisse inserito nulla.

**void createNewTask()** : funzione per creare un nuovo task ritornando l'apposita dialog e passandogli la funzione di salvataggio e i due controller (stateful, mantengono lo stato alla chiusura della dialog).

**void deleteTask(int index)** : elimina il task che corrisponde all'indice nella box che viene passato

#### 4.9.3 New Task Dialog

```
class NewTaskDialog extends StatelessWidget
```

Classe che ritorna la dialog per l'inserimento di un nuovo task, prende in input una funzione (salvataggio) e due controller che poi assegna ai due campi TextField.

#### 4.9.4 New Task Dialog

```
class AuthPage extends StatelessWidget
```

Classe che controlla se l'utente ha fatto l'accesso, per mostrare la pagina di switch tra sign in e sign up o direttamente la home page della community. Con uno stream che sta continuamente in ascolto per vedere se l'utente si è loggato.

#### 4.9.5 COMMUNITY : SignIn Or SignUp Page

```
class SigninOrSignupPage extends StatefulWidget
```

Pagina chiamata quando si preme sul pulsante community, gestisce lo switch tra pagine di sign in e sign up dopo il click sui rispettivi pulsanti, gestito con un toggle booleano. Inizialmente mostra sempre la pagina di sign in

#### 4.9.6 Login Page

```
class LoginPage extends StatefulWidget
```

Classe statful per mantenere lo stato di controller nei campi di input degli EditText. Nel widget crea il bottone di sign in con la classe MyButton, per avere la specifica del widget separata e utilizzabile anche per il sign up.

**void signInUser()** : Metodo che gestisce l'accesso con i vari controlli sui dati inseriti, gestisce le eccezioni inviate dal server (es: credenziali errate).

#### 4.9.7 Register Page

```
class RegisterPage extends StatefulWidget
```

**void signUpUser()** : esegue i controlli sui dati inseriti e gestisce le eccezioni del server, se tutto ha successo, esegue il metodo per registrare l'utente. L'utente è informato di eventuali errori attraverso alert dialog.

#### 4.9.8 Custom Button

```
class MyButton extends StatelessWidget
```

Classe che ritorna lo widget del bottone per l'accesso e la registrazione. Richiede come parametri del costruttore il testo e una funzione da eseguire al onTap.

#### 4.9.9 Community

```
class Community extends StatefulWidget
```

Ritorna un widget con uno stream builder sempre in ascolto sul riferimento alla collection 'commenti', e chiama il costruttore di Comment per costruire l'interfaccia con la lista di widget.

Le variabili definite nello stato sono:

- l'utente attualmente loggato (preso da un'istanza di FirebaseAuth),
- un riferimento alla collection 'commenti' su Firestore,
- controller del TextField,
- un HashMap che serve per mappare i dati in input da inviare poi a firebase.

**void createNewComment()** : metodo agganciato al fab, serve a chiamare la dialog per aggiungere un nuovo commento, passandogli un controller e il metodo per il salvataggio.

**void saveNewComment()** : controlla se è stato inserito del testo, e in tal caso mappa nella variabile di mapping i dati con i tre attributi dei commenti, che poi manda a Firebase per aggiungerli alla raccolta.

#### 4.9.10 Comment

Prende come parametri il testo, l'utente, e il timestamp. Serve a mappare i commenti che vengono ricevuti da firebase e inserirli in un Widget, che poi sarà chiamato dalla classe Community per costruire l'interfaccia.

#### 4.9.11 TUTORIAL

```
class Tutorial extends StatefulWidget
```

Le variabili sono un controller per l'image carousel, una lista in cui sono inseriti i path delle immagini da visualizzare e l'indice attuale del carosello (istanziato a zero).

**Widget buildIndicator()** : Crea l'indicatore della pagina del carosello, il numero di dot è pari alla lunghezza della lista delle immagini e l'indice attivo che viene modificato con set state in onPageChanged delle CarouselOptions.

**void animateToSlide(int index)** : restituisce il metodo per l'animazione di cambio pagina.

**Widget buildImage** : restituisce un container con dentro l'immagine. Perchè per visualizzare le immagini, le devo trasformare in widget (Container) da chiamare nella build.