



UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA
Dipartimento di Informatica, Sistemistica e
Comunicazione
Corso di Laurea in Informatica

Il ruolo degli LLM nello sviluppo del software: un'analisi sul supporto di ChatGPT nel processo di realizzazione dell'app "Communimib"

Relatore: Prof. Daniela Micucci

Correlatore: Dott. Maria Teresa Rossi

Tesi di Laurea di:
Luca Pincioli
Matricola 885969

Anno Accademico 2023-2024

Indice

Introduzione	4
1 Analisi dei requisiti	5
1.1 Come nasce Communimib	5
1.2 Analisi dei casi d'uso	7
1.2.1 Diagramma dei casi d'uso	7
1.2.2 Attori	8
1.2.3 Descrizione dei casi d'uso in formato dettagliato	9
1.3 Organizzazione dei requisiti	20
1.4 Modello di dominio	23
2 Descrizione dell'implementazione	24
2.1 Analisi architettonicale	24
2.1.1 Clean Architecture	25
2.1.2 Model-View-ViewModel (MVVM)	26
2.1.3 Diagramma dei package	28
2.2 Implementazione	29
2.2.1 Autenticazione	29
2.2.2 Bacheca	35
2.2.3 Segnalazioni	41
2.2.4 Profilo	48
2.2.5 Dettagli relativi all'uso di Firebase	51
3 Progetto di testing	53
3.1 Il ruolo dei test nello sviluppo Android	53
3.2 Test di unità	53
3.3 Test UI	54
4 Utilizzo di ChatGPT	56
4.1 Scopo dell'analisi sperimentale	56
4.2 Modalità di esecuzione e risultati ottenuti	57
4.2.1 Analisi dei requisiti	57
4.2.2 Implementazione	58
4.2.3 Testing	65
4.2.4 Sintesi dei dati raccolti	69
Conclusioni	71
Ringraziamenti	73
Sitografia	75

Elenco delle figure

1.1	Diagramma dei casi d'uso	7
1.2	Diagramma del modello di dominio	23
2.1	Clean Architecture	25
2.2	Architettura a tre layer	26
2.3	Diagramma dell'architettura	27
2.4	Diagramma dei package	28
2.5	Classe Validation	29
2.6	Login	30
2.7	Registrazione	30
2.8	Conferma mail	31
2.9	Reset password	31
2.10	Diagramma classi autenticazione - UI layer	33
2.11	Diagramma classi autenticazione - Data layer	34
2.12	Bacheca	36
2.13	Avviso nuovo post	36
2.14	Creazione post	37
2.15	Commenti	37
2.16	Diagramma classi bacheca - UI layer	39
2.17	Diagramma classi bacheca - Data layer	40
2.18	Segnalazioni	42
2.19	UI dipendente	42
2.20	Selezione preferiti	44
2.21	Selezione filtri	44
2.22	Segnalazione dettagliata	45
2.23	Creazione segnalazione	45
2.24	Diagramma classi segnalazioni - UI layer	46
2.25	Diagramma classi segnalazioni - Data layer	47
2.26	Profilo	48
2.27	Modifica profilo	48
2.28	Diagramma classi profilo - UI layer	50
4.1	Thread polling per la verifica dell'e-mail	58
4.2	Scelta del nome per uno dei tanti MutableLiveData	60
4.3	Estrazione di una classe ViewHolder	61
4.4	Posizionamento corretto della RecyclerView	63
4.5	Configurazione GitHub Actions	65
4.6	Implementazione del metodo getUserByEmail	66
4.7	Distribuzione delle interazioni con ChatGPT nelle diverse fasi di sviluppo	69

4.8	Utilità di ChatGPT durante la fase di implementazione	70
4.9	Utilità di ChatGPT durante la fase di testing	70

Elenco delle tabelle

1.1	Attore - Utente	8
1.2	Attore - Dipendente universitario	8
1.3	Caso d'uso - scegli edifici preferiti	9
1.4	Caso d'uso - visualizza segnalazioni	10
1.5	Caso d'uso - crea segnalazione	11
1.6	Caso d'uso - chiudi segnalazione	12
1.7	Caso d'uso - visualizza bacheca	13
1.8	Caso d'uso - crea post	14
1.9	Caso d'uso - visualizza profilo	15
1.10	Caso d'uso - modifica il proprio profilo	16
1.11	Caso d'uso - Elimina un proprio post	17
1.12	Caso d'uso - visualizza contenuti di un utente	18
1.13	Caso d'uso - Scrivi un commento	19
1.14	Requisiti funzionali	20
1.15	Requisiti non funzionali	22

Introduzione

La nascita dei Large Language Models ha dettato un punto di svolta nel progresso dell'intelligenza artificiale e nella sua applicazione in contesti quotidiani: l'impiego di questa tipologia di strumenti consente di eseguire con maggiore facilità una vasta gamma di operazioni che possono spaziare nelle discipline più disparate.

Tra i diversi LLM emersi negli anni, ChatGPT è quello che dal punto di vista mediatico ha messo maggiormente in luce le grandi potenzialità di cui tale famiglia di modelli dispone; grazie alla sua capacità di comprendere e generare testo in modo naturale e coerente, fin dal suo esordio ha spesso trovato occasione di essere utilizzato non solo per compiere operazioni semplici e ripetitive, ma anche per portare a termine compiti più complessi e articolati.

Tra le varie aree in cui i Large Language Models hanno trovato applicazione, l'ambito di maggiore interesse per questo elaborato è ovviamente quello informatico, in particolare lo sviluppo software; le caratteristiche che hanno reso ChatGPT particolarmente interessante per gli sviluppatori sono molteplici, una su tutte la capacità di comprendere, spiegare e generare codice sorgente.

In questo contesto, lo scopo del presente elaborato è quello di esaminare il ruolo assunto da ChatGPT in materia di sviluppo software, valutandone l'effettiva utilità all'interno del processo di realizzazione di un'applicazione complessa. Tale obiettivo è stato perseguito mediante la conduzione di un'analisi sperimentale basata sullo sviluppo di un'applicazione volta a migliorare la comunicazione tra i diversi soggetti che compongono l'ateneo: il suo nome trae origine dai termini "Community" e "Unimib", la cui unione da vita a *Communimib*.

Communimib dovrà essere in grado di rendere l'università più connessa, mettendo a disposizione due funzionalità principali:

- una **bacheca**, che permetta a studenti e personale universitario di pubblicare liberamente pensieri, notizie e annunci riguardanti l'ateneo
- un **sistema di segnalazione**, che consenta di segnalare agli addetti ai lavori i problemi che si verificano all'interno degli edifici, favorendone una rapida risoluzione

Durante il processo di realizzazione dell'applicazione, ChatGPT ha ricoperto un importante ruolo nello svolgimento di mansioni quali scrittura del codice, refactoring e risoluzione dei malfunzionamenti; il suo impiego è stato proficuo per la buona riuscita del progetto, avvalorando la tesi che lo descrive come uno strumento efficace in ambito di sviluppo.

Capitolo 1

Analisi dei requisiti

1.1 Come nasce Communimib

Communimib, come la maggior parte delle applicazioni mobili più diffuse al giorno d'oggi, nasce da un desiderio dei suoi sviluppatori: rendere più semplice ed efficace la comunicazione all'interno dell'ateneo.

La comunicazione tra i diversi soggetti di un'organizzazione rappresenta un fattore fondamentale per la creazione di un ambiente lavorativo sano e piacevole; per questo motivo Communimib ha l'ambizione di rappresentare uno strumento di comunicazione, immediato e di facile utilizzo, per tutti coloro che quotidianamente vivono l'ambiente universitario.

Le diverse idee su cui l'applicazione si fonda trovano riscontro concreto in due funzionalità chiave, che insieme compongono l'essenza di Communimib: il sistema di segnalazione e la bacheca universitaria.

La prima funzionalità consiste in un sistema che permette agli utenti di segnalare eventuali problematiche riscontrate negli edifici, come ad esempio il guasto di un erogatore d'acqua o la mancanza di materiale nelle aule. Il sistema di segnalazione svolge un ruolo importante nella collaborazione tra studenti e personale dell'ateneo: con il suo utilizzo gli studenti si impegnano a segnalare tutti gli inconvenienti che possono disturbare la normale prosecuzione delle attività universitarie, consentendo così al personale una rapida risoluzione.

La seconda funzionalità si configura in una piattaforma in cui gli utenti possono pubblicare post atti a raggiungere, in base alle proprie esigenze, gli altri membri della comunità universitaria. Tale piattaforma consente di ammodernare l'anacronistico uso delle bacheche fisiche presenti negli edifici, i cui annunci spesso non raggiungono i destinatari sperati: non è raro trovarne di estremamente inattuali ed obsoleti. Inoltre, qualora si fosse alla ricerca di un annuncio specifico, spesso è necessario visitare fisicamente la bacheca di ogni plesso; per questo motivo, la bacheca di Communimib rappresenta uno spazio virtuale in cui condividere post facilmente accessibili ai fruitori dell'intero ateneo.

Al fine di costruire un'applicazione dinamica e capace di adattarsi allo stile di chi la utilizza, è necessario mettere a disposizione degli utenti la possibilità di personalizzare il proprio profilo e i propri interessi. A tal proposito, si può pensare ad un semplice esempio: ogni studente, in base alla facoltà a cui è iscritto e alle lezioni che decide di seguire, tende a frequentare spesso lo stesso insieme (solitamente ristretto) di edifici, pertanto sarà probabilmente interessato a conoscere soltanto le segnalazioni attive in tali edifici. Contestualmente, è probabile che non tutti gli utenti facciano uso della bacheca allo stesso modo: a titolo esemplificativo, si pensi agli studenti che intendono prendere lezioni private per un superare un esame difficile, a coloro che sono alla ricerca di una casa in affitto, a chi ha semplicemente smarrito la giacca o chi invece vuole promuovere un'iniziativa benefica.

Per soddisfare adeguatamente gran parte delle possibili modalità di utilizzo di Communimib, sia il sistema di segnalazione che la bacheca devono quindi consentire all'utente di individuare e visualizzare soltanto i contenuti di suo interesse.

1.2 Analisi dei casi d'uso

Al fine di garantire la buona riuscita dell'intero progetto, la prima fase dello sviluppo è stata dedicata all'analisi e definizione dei casi d'uso dell'applicazione. Questo passo è stato di fondamentale importanza per porre delle solide basi all'interno del team di sviluppo, permettendo l'acquisizione di una visione comune sul sistema da implementare.

Il processo che ha portato alla definizione dei casi d'uso ha inoltre consentito di approfondire ed analizzare in maniera critica ciascuna funzionalità dell'applicativo, identificando dettagli e proprietà aggiuntive che sono state successivamente implementate.

1.2.1 Diagramma dei casi d'uso

Il principale elaborato prodotto in questa fase è il diagramma dei casi d'uso (si veda Figura 1.1), il quale sintetizza visivamente i casi di utilizzo che descrivono l'applicazione e le relazioni che sussistono tra essi.

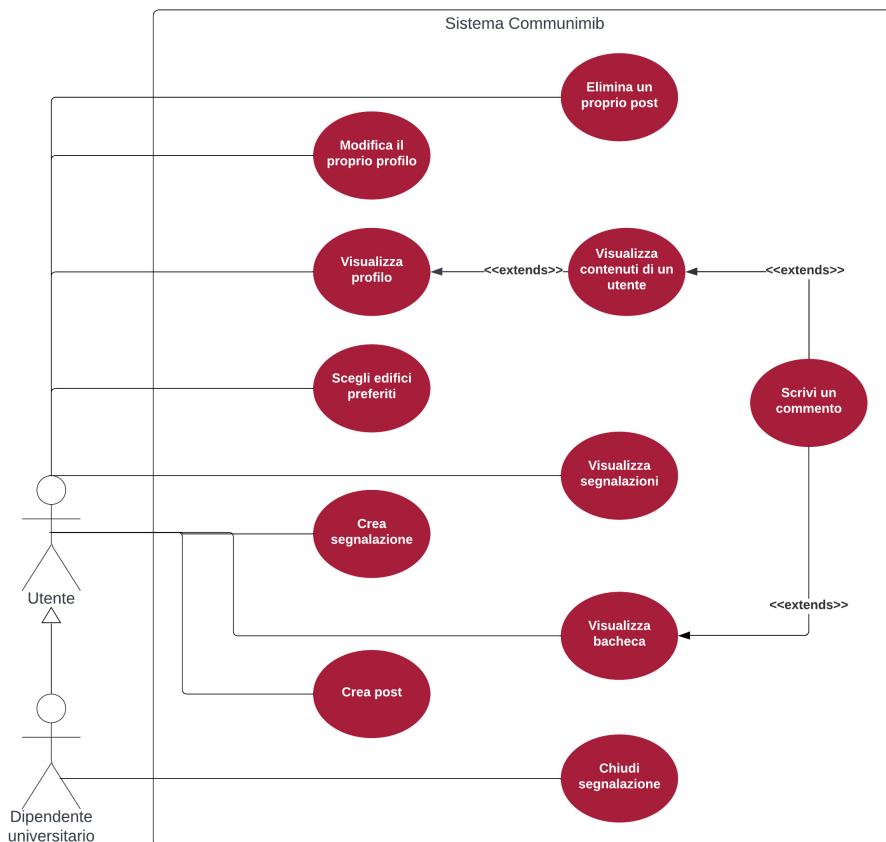


Figura 1.1: Diagramma dei casi d'uso

1.2.2 Attori

Durante le fasi preliminari dell'analisi, l'individuazione degli attori coinvolti nell'utilizzo dell'applicazione è un'attività sicuramente prioritaria: catalogare correttamente tali attori rappresenta un passaggio importante verso un'efficace stesura dei casi di utilizzo.

All'interno del diagramma dei casi d'uso (si veda Figura 1.1) sono presenti due attori che utilizzano il sistema Communimib, *Utente* e *Dipendente universitario*, rispettivamente descritti in Tabella 1.1 e Tabella 1.2 riportate di seguito.

Tabella 1.1: Attore - Utente

Nome dell'attore	Utente
Ruolo	Attore primario
Descrizione	Rappresenta una generica figura facente parte dell'università. Un utente, per poter essere considerato tale, deve aver completato la procedura di registrazione nel sistema utilizzando la propria email universitaria (la quale termina con il dominio "campus.unimib.it" oppure "unimib.it").

Tabella 1.2: Attore - Dipendente universitario

Nome dell'attore	Dipendente universitario
Ruolo	Attore primario
Descrizione	Rappresenta un generico componente del personale universitario, di conseguenza ha la possibilità di eseguire un insieme più ampio di operazioni rispetto ad un normale utente. Questa caratteristica viene rappresentata all'interno del diagramma dei casi d'uso mediante la relazione di generalizzazione posta tra gli attori "Utente" e "Dipendente universitario". Il criterio che distingue i dipendenti universitari dal resto degli utenti è il possesso di una mail universitaria avente dominio "unimib.it".

1.2.3 Descrizione dei casi d'uso in formato dettagliato

A partire dal diagramma dei casi d'utilizzo e dopo aver definito gli attori che ne fanno parte, la fase di analisi dei casi d'uso si concretizza con la descrizione dei casi d'uso in formato dettagliato; attraverso la stesura di tali elaborati, ciascun caso d'uso è stato opportunamente catalogato e commentato (si veda da Tabella 1.3 a Tabella 1.13) al fine di identificarne le caratteristiche fondamentali.

Tabella 1.3: Caso d'uso - scegli edifici preferiti

Nome del caso d'uso	Scegli edifici preferiti
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a selezionare un insieme di edifici preferiti tra quelli appartenenti all'ateneo.
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura gli edifici scelti dall'utente vengono registrati nel sistema ed associati al profilo personale.
Scenario principale di successo	<ol style="list-style-type: none"> 1. L'utente visualizza l'elenco degli edifici dell'ateneo. 2. L'utente seleziona l'insieme degli edifici a cui è interessato. 3. Il sistema memorizza gli edifici selezionati.
Estensioni	Scenario alternativo di fallimento: <ol style="list-style-type: none"> 3.a. Se il sistema fallisce nel salvataggio degli edifici selezionati, l'utente visualizza il relativo messaggio d'errore.
Requisiti speciali	Nessuno
Frequenza di ripetizione	Bassa

Tabella 1.4: Caso d'uso - visualizza segnalazioni

Nome del caso d'uso	Visualizza segnalazioni
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a consultare un insieme di segnalazioni tra quelle presenti nel sistema.
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura l'utente visualizza una lista di segnalazioni
Scenario principale di successo	<p>1. L'utente accede alla sezione dell'applicativo dedicata alle segnalazioni.</p> <p>2. Il sistema mostra all'utente le segnalazioni richieste.</p>
Estensioni	<p>Scenario alternativo di successo:</p> <p>1. L'utente può applicare uno dei seguenti filtri per visualizzare un sottoinsieme di segnalazioni presenti nel sistema:</p> <ul style="list-style-type: none"> • Edifici preferiti (si veda Tabella 1.3). • Edifici specifici (Es. edificio U1 ed U14). • Tutti gli edifici. <p>2. Il sistema mostra all'utente le segnalazioni filtrate.</p> <p>Scenario alternativo di fallimento:</p> <p>2.a. Se il sistema fallisce nella visualizzazione delle segnalazioni, viene mostrato all'utente il relativo segnale di errore.</p>
Requisiti speciali	Nessuno
Frequenza di ripetizione	Alta

Tabella 1.5: Caso d'uso - crea segnalazione

Nome del caso d'uso	Crea segnalazione
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a pubblicare una segnalazione nel sistema per condividere agli altri utenti un problema.
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura la segnalazione è stata registrata nel sistema.
Scenario principale di successo	<ol style="list-style-type: none"> 1. L'utente accede alla sezione per pubblicare una nuova segnalazione. 2. L'utente inserisce i dati richiesti per compiere la segnalazione. 3. Il sistema pubblica la segnalazione.
Estensioni	Scenari alternativi di fallimento: <ol style="list-style-type: none"> 2.a. Se l'utente inserisce dei dati non validi, la segnalazione non può essere pubblicata e viene mostrato il relativo messaggio d'errore. 3.a. Se il sistema fallisce nella pubblicazione della segnalazione, viene mostrato all'utente il relativo segnale di errore.
Requisiti speciali	Nessuno
Frequenza di ripetizione	Media

Tabella 1.6: Caso d'uso - chiudi segnalazione

Nome del caso d'uso	Chiudi segnalazione
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Dipendente universitario
Parti interessate ed interessi	Il dipendente universitario è interessato a chiudere una segnalazione.
Pre-condizioni	Il dipendente universitario deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura, il dipendente universitario ha correttamente chiuso la segnalazione desiderata.
Scenario principale di successo	<ol style="list-style-type: none"> 1. Il dipendente universitario sceglie la segnalazione che desidera chiudere. 2. Il sistema avvia la chiusura della segnalazione.
Estensioni	Scenario alternativo di fallimento: <ol style="list-style-type: none"> 2.a Se il sistema fallisce durante la chiusura della segnalazione, viene mostrato al dipendente universitario il relativo messaggio di errore.
Requisiti speciali	Nessuno
Frequenza di ripetizione	Bassa

Tabella 1.7: Caso d'uso - visualizza bacheca

Nome del caso d'uso	Visualizza bacheca
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a visualizzare i post pubblicati nella bacheca.
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura, l'utente ha visualizzato i post della bacheca.
Scenario principale di successo	<p>1. L'utente accede alla sezione dedicata alla visualizzazione della bacheca.</p> <p>2. Il sistema mostra all'utente i post pubblicati nella bacheca.</p>
Estensioni	<p>Scenario alternativo di successo:</p> <p>1. L'utente personalizza la visualizzazione dei post filtrandoli per categoria.</p> <p>2. Il sistema mostra i post appartenenti alla categoria selezionata.</p> <p>Scenario alternativo di fallimento:</p> <p>2.a Se il sistema fallisce durante la visualizzazione dei post, viene mostrato il relativo messaggio di errore.</p>
Requisiti speciali	Nessuno
Frequenza di ripetizione	Alta

Tabella 1.8: Caso d'uso - crea post

Nome del caso d'uso	Crea post
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a pubblicare un nuovo post nella bacheca.
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura, l'utente ha creato un nuovo post visualizzabile attraverso la bacheca.
Scenario principale di successo	<ol style="list-style-type: none"> 1. L'utente accede alla sezione che consente la pubblicazione di un nuovo post. 2. L'utente inserisce i dati relativi al post che desidera pubblicare. 3. Il sistema pubblica il post.
Estensioni	Scenari alternativi di fallimento: <ol style="list-style-type: none"> 2.a Se l'utente inserisce dei dati non validi, il post non può essere pubblicato e viene mostrato all'utente il relativo messaggio di errore. 2.b Se l'utente non inserisce informazioni sufficienti alla creazione del post, il post non può essere pubblicato e viene mostrato all'utente il relativo messaggio di errore. 3.a Se il sistema fallisce nella creazione del post, viene mostrato all'utente il relativo messaggio di errore.
Requisiti speciali	Nessuno
Frequenza di ripetizione	Media

Tabella 1.9: Caso d'uso - visualizza profilo

Nome del caso d'uso	Visualizza profilo
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a visualizzare un profilo (proprio oppure di terzi).
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura, l'utente ha visualizzato le informazioni relative al profilo desiderato.
Scenario principale di successo	<ol style="list-style-type: none"> 1. L'utente accede alla sezione per la visualizzazione del proprio profilo. 2. Il sistema mostra i dati associati al proprio profilo.
Estensioni	<p>Scenario alternativo di successo:</p> <ol style="list-style-type: none"> 1. L'utente richiede la visualizzazione del profilo di un altro utente. 2. Il sistema mostra i dati associati al profilo desiderato. <p>Scenario alternativo di fallimento:</p> <ol style="list-style-type: none"> 2.a Se il sistema fallisce nella visualizzazione dei dati, viene mostrato il relativo messaggio di errore.
Requisiti speciali	Nessuno
Frequenza di ripetizione	Media

Tabella 1.10: Caso d'uso - modifica il proprio profilo

Nome del caso d'uso	Modifica il proprio profilo
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a modificare le informazioni associate al profilo personale.
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura, l'utente ha modificato le informazioni relative al profilo personale.
Scenario principale di successo	<ol style="list-style-type: none"> 1. L'utente accede alla sezione di modifica del profilo. 2. L'utente sceglie di modificare i dati associati al proprio profilo. 3. Il sistema salva i dati inseriti e mostra un messaggio di conferma.
Estensioni	Scenario alternativo di fallimento: <ol style="list-style-type: none"> 3.a. Se il sistema fallisce nel salvataggio dei dati, viene mostrato il relativo messaggio di errore.
Requisiti speciali	Nessuno
Frequenza di ripetizione	Bassa

Tabella 1.11: Caso d'uso - Elimina un proprio post

Nome del caso d'uso	Elimina un proprio post
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a eliminare un post che ha pubblicato.
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura, l'utente ha eliminato il post.
Scenario principale di successo	<ol style="list-style-type: none"> 1. L'utente accede alla sezione per la visualizzazione del proprio profilo. 2. L'utente accede alla sezione per la visualizzazione dei propri post. 3. L'utente seleziona il post che desidera eliminare. 4. Il sistema elimina il post, dando all'utente la possibilità di annullare l'azione.
Estensioni	<p>Scenario alternativo di successo:</p> <ol style="list-style-type: none"> 4.a. L'utente richiede l'annullamento dell'eliminazione del post. 4.b. Il sistema ripristina il post. <p>Scenario alternativo di fallimento:</p> <ol style="list-style-type: none"> 4.a. Se il sistema fallisce durante l'eliminazione del post, viene mostrato il relativo messaggio di errore.
Requisiti speciali	Nessuno
Frequenza di ripetizione	Bassa

Tabella 1.12: Caso d'uso - visualizza contenuti di un utente

Nome del caso d'uso	Visualizza contenuti di un utente
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a visualizzare i post o le segnalazioni di un profilo (proprio o altrui).
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura, l'utente ha visualizzato i post o le segnalazioni del profilo desiderato.
Scenario principale di successo	<ol style="list-style-type: none"> 1. L'utente accede alla sezione dedicata alla visualizzazione di un profilo (proprio o altrui). 2. L'utente accede alla sezione per la visualizzazione dei post. 3. il sistema mostra all'utente i post pubblicati.
Estensioni	<p>Scenario alternativo di successo:</p> <ol style="list-style-type: none"> 1. L'utente accede alla sezione dedicata alla visualizzazione di un profilo (proprio o altrui). 2. L'utente accede alla sezione dedicata alla visualizzazione delle segnalazioni. 3. il sistema mostra all'utente le segnalazioni pubblicate. <p>Scenario alternativo di fallimento:</p> <ol style="list-style-type: none"> 3.a. Se il sistema fallisce durante la visualizzazione dei post o delle segnalazioni, viene mostrato il relativo messaggio di errore.
Requisiti speciali	Nessuno
Frequenza di ripetizione	Media

Tabella 1.13: Caso d'uso - Scrivi un commento

Nome del caso d'uso	Scrivi un commento
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a scrivere un commento in un post.
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura, l'utente ha aggiunto un commento a un post.
Scenario principale di successo	<ol style="list-style-type: none"> 1. L'utente accede alla sezione adibita alla visualizzazione della bacheca. 2. L'utente seleziona il post che vuole commentare. 3. L'utente scrive il commento e lo pubblica. 4. Il sistema memorizza il commento e lo mostra nell'elenco dei commenti presenti.
Estensioni	<p>Scenario alternativo di successo:</p> <ol style="list-style-type: none"> 1. L'utente accede alla sezione dedicata alla visualizzazione del profilo (proprio o altrui). 2. L'utente accede alla sezione dedicata alla visualizzazione dei contenuti del profilo. 3. L'utente clicca sul post che vuole commentare. 4. L'utente scrive il commento e lo pubblica. 5. Il sistema memorizza il commento e lo mostra nell'elenco dei commenti presenti. <p>Scenario alternativo di fallimento:</p> <ol style="list-style-type: none"> 4.a. Se il sistema fallisce durante la memorizzazione del commento, viene mostrato il relativo messaggio di errore.
Requisiti speciali	Nessuno
Frequenza di ripetizione	Bassa

1.3 Organizzazione dei requisiti

Mediante l'analisi dei casi d'uso è stato possibile scoprire e descrivere dettagliatamente i requisiti funzionali, delineando così una visione più chiara del sistema da implementare. Un ulteriore elaborato che risulta propedeutico per un buon svolgimento dell'attività di sviluppo è l'elenco dei requisiti; si noti che la sua stesura non può in alcun modo sostituire il processo di analisi dei casi d'uso svolto in precedenza, dal momento che l'elencazione dei requisiti non permette la loro scoperta, ma soltanto la loro organizzazione.

I requisiti individuati sono stati suddivisi in Tabella 1.14 e Tabella 1.15 che riportano rispettivamente i requisiti funzionali e non funzionali.

L'assegnazione della priorità è stata effettuata secondo il metodo MoSCoW, il quale prevede quattro diversi livelli di priorità:

- **M**, "Must": descrive un requisito che deve essere soddisfatto nella soluzione finale, affinché essa sia considerata un successo.
- **S**, "Should": rappresenta un aspetto importante che – nei limiti del possibile – dovrebbe essere compreso nella soluzione.
- **C**, "Could": descrive un requisito che è considerato auspicabile ma non necessario; sarà incluso se il tempo e le risorse lo permettono.
- **W**, "Would": descrive un requisito che può essere preso in considerazione negli sviluppi futuri del sistema. [8]

Tabella 1.14: Requisiti funzionali

ID	Descrizione	Priorità	Effettivo sviluppo
RF1	Communimib dovrà permettere agli utenti di creare segnalazioni in cui vengono descritti i problemi attualmente presenti negli edifici dell'ateneo	M	Implementato
RF2	Communimib dovrà raggruppare le segnalazioni sulla base dell'edificio a cui si riferiscono	S	Implementato
RF3	Communimib dovrà mettere a disposizione un insieme di categorie per catalogare segnalazioni di natura differente	M	Implementato
RF4	Communimib dovrà consentire al personale universitario di chiudere le segnalazioni riferite a problematiche che sono state risolte	M	Implementato

ID	Descrizione	Priorità	Effettivo sviluppo
RF5	Communimib dovrà eliminare tutte le segnalazioni attive alla fine di ogni giornata	W	Non implementato
RF6	Communimib dovrà permettere agli utenti di ricercare una segnalazione tra quelle presenti nel sistema	C	Implementato
RF7	Communimib dovrà permettere ad ogni utente di scegliere i propri edifici di interesse	S	Implementato
RF8	Communimib dovrà inviare una notifica a tutti gli utenti interessati ad un edificio quando viene aggiunta una nuova segnalazione ad esso associata	C	Non implementato
RF9	Communimib dovrà consentire agli utenti di creare post per la bacheca	M	Implementato
RF10	Communimib dovrà mettere a disposizione un insieme di categorie per catalogare post di natura differente	M	Implementato
RF11	Communimib dovrà mettere a disposizione del personale universitario una categoria che identifichi le comunicazioni ufficiali	M	Implementato
RF12	Communimib dovrà raggruppare ciascun post della bacheca sulla base della categoria ad esso associata	S	Implementato
RF13	Communimib dovrà consentire agli utenti di eliminare i propri post	S	Implementato
RF14	Communimib dovrà avvisare gli utenti che sono stati aggiunti nuovi post riferiti alla categoria che stanno visualizzando	C	Implementato
RF15	Communimib dovrà permettere agli utenti di ricercare un post tra quelli presenti nella bacheca	C	Implementato
RF16	Communimib dovrà permettere agli utenti di pubblicare commenti relativi ai post (propri o di altri utenti)	M	Implementato

ID	Descrizione	Priorità	Effettivo sviluppo
RF17	Communimib dovrà mettere a disposizione degli utenti la possibilità di modificare il proprio profilo personale	M	Implementato
RF18	Communimib dovrà consentire agli utenti di visualizzare il proprio profilo ed il profilo di altri utenti	M	Implementato
RF19	Communimib dovrà distinguere visivamente i profili dei dipendenti universitari dai profili degli utenti normali	S	Implementato

Tabella 1.15: Requisiti non funzionali

ID	Descrizione	Priorità	Effettivo sviluppo
RNF1	Communimib dovrà rendere disponibili in tempo reale i contenuti più recenti sull'app di ciascun utente	M	Implementato
RNF2	Communimib dovrà essere disponibile 24 ore su 24, 7 giorni su 7	S	Implementato
RNF3	Communimib dovrà essere limitatamente consultabile anche in assenza di una connessione ad internet	S	Implementato

1.4 Modello di dominio

Il modello di dominio è una rappresentazione visuale di classi concettuali o oggetti appartenenti ad uno specifico dominio; tale modello è stato realizzato per comprendere il contesto in cui deve operare il sistema, rappresentando le entità chiave con le relative relazioni.

In particolare, nel dominio in esame sono state identificate tre componenti principali: *Utente*, *Post* e *Segnalazione*. Al fine di arricchire la rappresentazione sono stati successivamente introdotti due ulteriori elementi: *Commento*, il quale si riferisce ad un post e viene scritto da un utente, ed *Edificio*, il quale è associato ad una segnalazione.

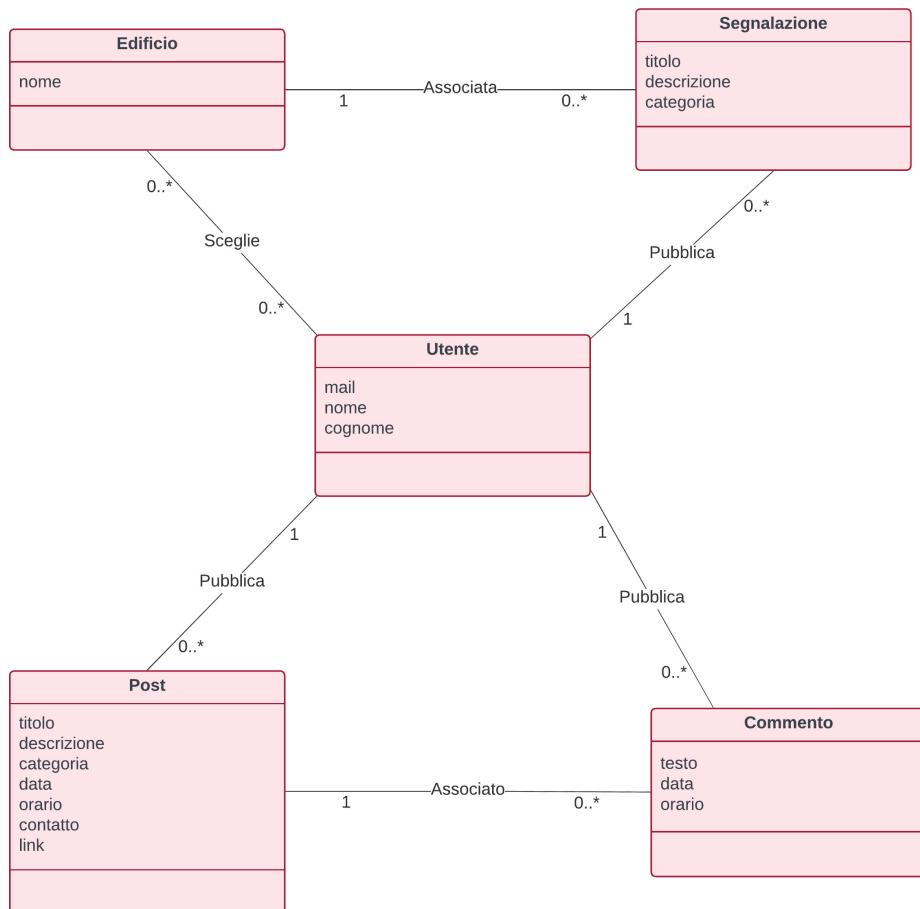


Figura 1.2: Diagramma del modello di dominio

Capitolo 2

Descrizione dell'implementazione

2.1 Analisi architetturale

L’architettura software di un’applicazione rappresenta il risultato delle decisioni significative che vengono prese per organizzare i componenti (classi, interfacce, ecc.) che costituiscono il sistema.

La progettazione e l’implementazione di un’architettura software corretta sono operazioni fondamentali nel procedimento di creazione di un sistema complesso, perché garantiscono:

- **Flessibilità.** Si tratta di una caratteristica estremamente importante, poiché rappresenta il grado di semplicità riscontrabile quando si eseguono operazioni di modifica e manutenzione del codice. Durante l’implementazione è comune identificare errori o problemi che generano malfunzionamenti all’interno del sistema: un’architettura stabile e coerente permette di gestire e risolvere tali problematiche senza generarne di nuove.
- **Scalabilità.** L’applicazione è estendibile, di conseguenza si presta facilmente all’aggiunta di nuovi contenuti. In generale un sistema non dovrebbe mai essere statico, chiuso e quindi limitato alle funzionalità implementate inizialmente. Questa caratteristica garantisce anche durabilità del software, poiché può essere modificato ed esteso nel tempo a seconda delle funzionalità richieste.
- **Riutilizzo.** Quando il contesto lo permette, è possibile riutilizzare il codice scritto, senza duplicarlo. Questo non solo riduce il tempo e gli sforzi necessari per lo sviluppo di nuove funzionalità, ma assicura anche una maggiore coerenza e qualità del codice, in quanto il riutilizzo di componenti già esistenti implica che questi siano già testati e ottimizzati.

Al fine di garantire tali principi, lo sviluppo di Communimib si basa sul concetto fondamentale di Clean Architecture e sull’applicazione del pattern architettonico Model-View-ViewModel, i quali sono descritti dettagliatamente nella sezione seguente.

2.1.1 Clean Architecture

Con il termine "Clean Architecture" si fa riferimento ad un insieme di linee guida per progettare in modo efficace l'architettura di un software, definendo come suddividerla in livelli e fissando confini chiari tra questi.

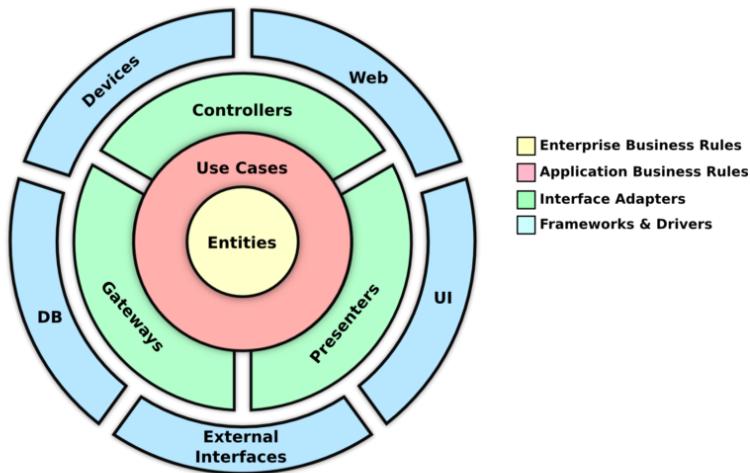


Figura 2.1: Clean Architecture

La regola fondamentale su cui si basa la Clean Architecture è la **Dependency Rule**, secondo la quale la direzione delle dipendenze deve sempre puntare verso i livelli centrali, che sono i meno dipendenti dai dettagli di implementazione. Questo approccio facilita il testing, la manutenzione e l'evoluzione del software. [7]

La Clean Architecture tende ad essere un punto di riferimento per lo sviluppo di sistemi complessi, categoria che non comprende le app Android; per tale motivo, nello sviluppo di applicazioni Android viene generalmente seguito un insieme di linee guida che si concretizzano nella **Modern App Architecture**, la quale rappresenta una versione più flessibile della Clean Architecture.

I principi cardine della Modern App Architecture sono:

- **Separation of concerns:** separazione delle responsabilità tra le componenti del software.
- **Drive UI from data models:** la UI dovrebbe essere aggiornata sulla base dei dati contenuti in appositi componenti separati dalla UI.
- **Single source of truth:** deve essere mantenuta una singola fonte di dati, che sia sicura e affidabile.
- **Unidirectional data flow:** lo stato fluisce in una sola direzione, dal livello basso al livello alto, mentre gli eventi che modificano i dati fluiscono in direzione opposta.

L'architettura può essere semplificata in tre livelli:

- **UI Layer**: gestisce gli input e gli output degli utenti e l'aggiornamento della visualizzazione; è dunque il layer che contiene tutti gli elementi UI, che vengono visualizzati sullo schermo, e gli state holder, che contengono i dati e gestiscono la logica con la quale questi vengono esposti all'interfaccia utente.
- **Domain Layer**: layer opzionale la cui funzione è quella di semplificare ed ottimizzare le interazioni tra UI e Data layer.
- **Data Layer**: contiene la logica di business dell'applicazione ed espone i dati ai layer superiori; questo livello definisce le regole che determinano le modalità di creazione, archiviazione e modifica dei dati, ed è composto da due elementi: i Repository e i Data Source. [4]

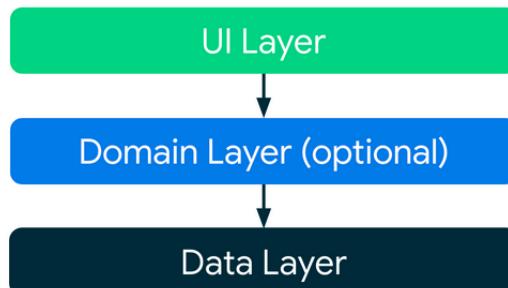


Figura 2.2: Architettura a tre layer

2.1.2 Model-View-ViewModel (MVVM)

Dati i principi precedentemente elencati, il pattern architettonico utilizzato nello sviluppo dell'applicativo è noto come **Model-View-ViewModel**.

Questo suggerisce la costituzione di tre componenti:

- **Model**: è il componente che contiene gli oggetti business che encapsulano i dati e definisce il comportamento del dominio applicativo;
- **View**: rappresenta la componente che fornisce l'interfaccia utente, pertanto si occupa di gestire la visualizzazione dei dati forniti dal ViewModel senza effettuare alcuna manipolazione su di essi.
- **ViewModel**: è il componente che gestisce l'interazione tra View e Model, mantenendo lo stato della View aggiornato sulla base dei dati presenti nel Model.

La View è costituita dalle classi Activity e Fragment, che contengono le componenti grafiche per la visualizzazione dei dati. Per ogni interfaccia è stato

implementato il relativo ViewModel, che fornisce i dati da mostrare all'utente. Ogni ViewModel reperisce i dati dal Model sottostante, costituito dalle classi Repository e dalle classi Datasource; i Repository fungono da intermediari tra i ViewModel e i Datasource, mentre questi ultimi rappresentano l'effettiva fonte dei dati. Per ogni tipologia di dato vengono generalmente implementati due Datasource, uno locale che sfrutta il dispositivo su cui l'applicazione viene eseguita ed uno remoto che invece utilizza i servizi offerti da Firebase per la memorizzazione dei dati.

La suddivisione delle componenti è visualizzabile nel diagramma seguente.

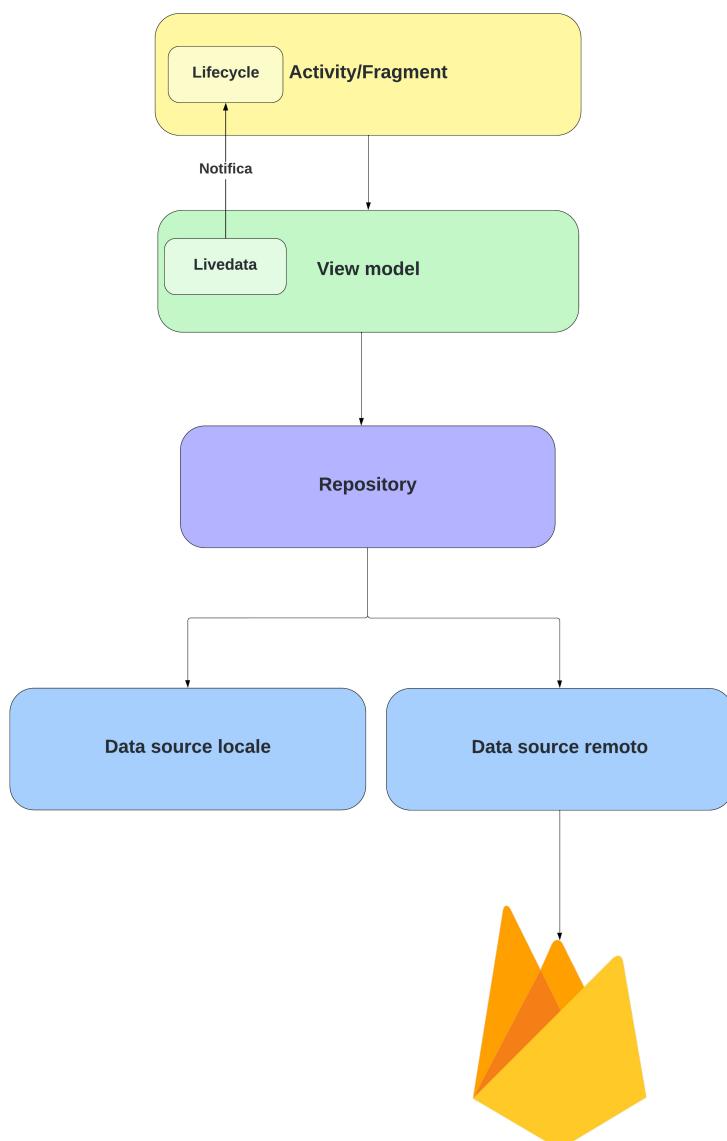


Figura 2.3: Diagramma dell'architettura

2.1.3 Diagramma dei package

Il diagramma dei package (Figura 2.4) è una rappresentazione grafica dell’architettura software ed identifica la trasposizione del concetto di Clean Architecture e l’applicazione del pattern MVVM all’interno del sistema Communimib. Come descritto precedentemente, l’architettura è stata suddivisa in due package principali: il primo permette di modellare l’interfaccia l’interfaccia grafica (UI), mentre il secondo rappresenta il data-layer. Il package util contiene invece classi utilizzate come supporto all’applicazione al fine di completare operazioni specifiche o ripetute. Infine, il package model presenta i dati gestiti ed utilizzati dall’applicativo.

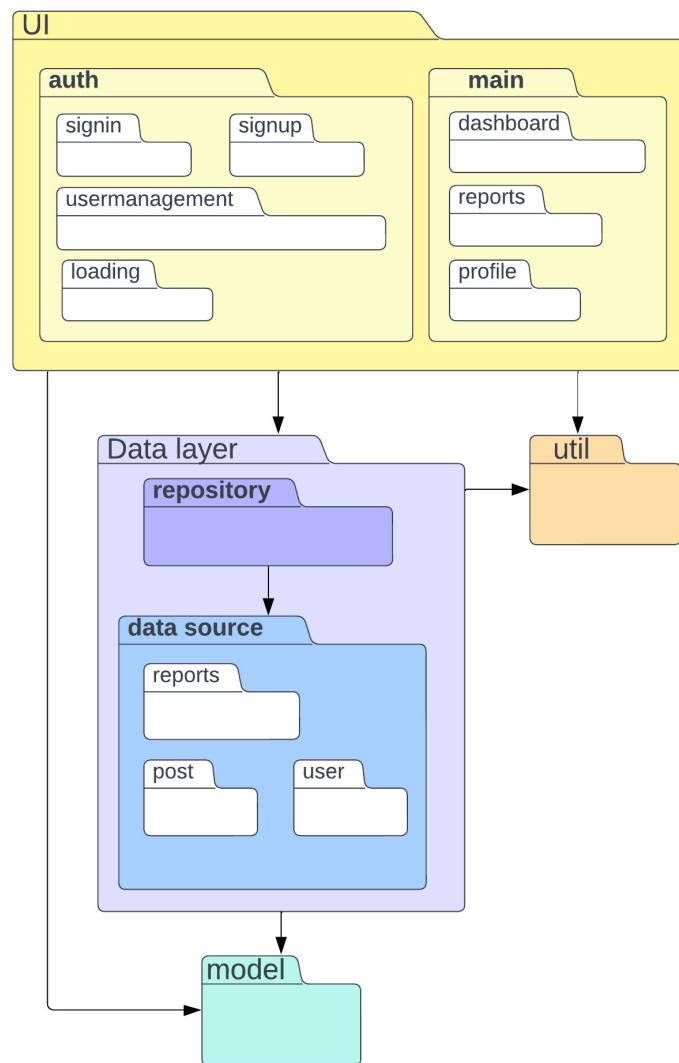


Figura 2.4: Diagramma dei package

2.2 Implementazione

Dopo aver fissato i requisiti e definito l'architettura, segue la fase di implementazione del software.

In questa sezione verranno descritti i dettagli tecnici relativi alla realizzazione dell'applicazione mobile; in particolare, verranno illustrati gli aspetti chiave della struttura del codice, le classi utilizzate, le interfacce definite e le principali funzionalità implementate.

2.2.1 Autenticazione

La sezione relativa all'autenticazione è fondamentale per garantire la sicurezza dell'utente all'interno dell'applicazione. Attraverso un sistema di autenticazione solido, tutti coloro che frequentano l'ateneo possono godere di un'applicazione a loro riservata, che non permette l'accesso a soggetti esterni all'università. Il meccanismo di autenticazione è stato strutturato e gestito su quattro aspetti fondamentali: registrazione, login, validazione della mail e recupero della password.

Registrazione (Sign up)

Al primo utilizzo dell'applicazione, l'utente visualizza la schermata relativa alla registrazione (si veda Figura 2.7), che permette di creare l'account personale. All'utente viene richiesto l'inserimento di alcuni parametri, i quali devono rispettare formati precisi e specifici al fine di completare correttamente la procedura:

- **Indirizzo e-mail.** Permette di identificare univocamente ogni utente, in quanto non possono esistere due utenti aventi lo stesso indirizzo e-mail.
- **Password.** Rappresenta una parola chiave scelta dall'utente che ne consente l'accesso al sistema. La password deve contenere almeno un numero, un carattere speciale, una lettera maiuscola e deve essere lunga almeno otto caratteri. L'utente deve inoltre confermare la password scelta nell'apposito campo.
- **Nome e cognome** dell'utente. In questo caso non è possibile inserire numeri e caratteri speciali.

Le operazioni di validazione e controllo vengono gestite interamente dalla classe **Validation**, rappresentata nell'immagine a lato. I dati vengono successivamente gestiti nei livelli inferiori dell'applicativo e registrati all'interno del backend di Firebase. E-mail e password dell'utente vengono memorizzati nel sistema Firebase Authentication, mentre i dati restanti vengono memorizzati in Firebase Realtime Database ed identificati mediante un id univoco.

Validation
<pre>- Validation() + checkEmail(String) : String + checkPassword(String) : String + checkConfirmPassword(String, String) : String + checkField(String) : String + validateNewReport(String, String, String, String) : String + checkBuildingsSpinner(String) : String + checkCategoriesSpinner(String) : String + checkEmptyField(String) : String + isEmailValid(String) : boolean + isValidLink(String) : boolean</pre>

Figura 2.5: Classe Validation

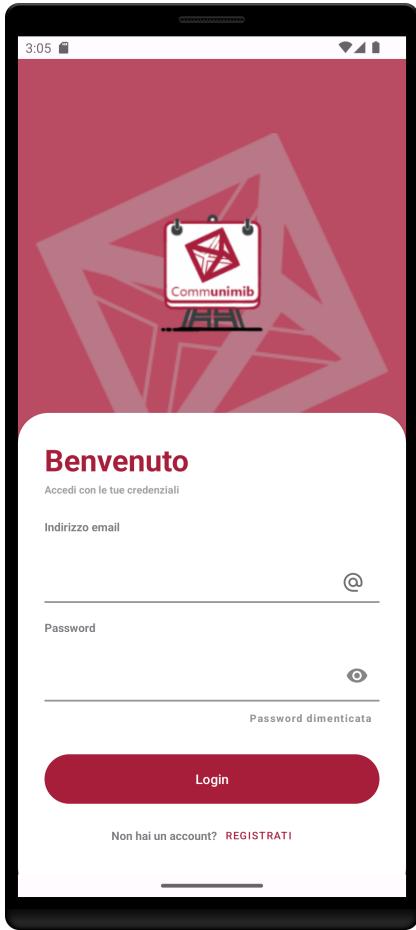


Figura 2.6: Login

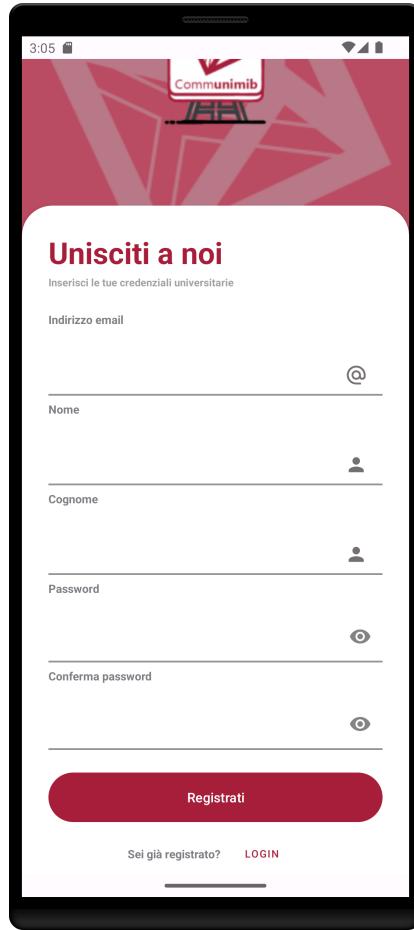


Figura 2.7: Registrazione

Login (Sign in)

Per ragioni di sicurezza, l'accesso all'applicazione è permesso solo agli utenti che dispongono di un profilo registrato nel sistema. L'applicativo offre dunque la funzionalità di login (Figura 2.6) per consentire agli utenti di autenticarsi nel sistema e procedere con l'utilizzo dell'applicazione.

La procedura di accesso si svolge come segue:

1. l'utente inserisce il suo indirizzo e-mail universitario e la password scelta in fase di registrazione.
2. il sistema provvede a verificare che l'indirizzo e-mail inserito sia valido.
3. se il sistema non segnala la presenza di errori, l'utente può effettuare un tap sul pulsante di login, innescando una query al database remoto.
4. se il sistema riscontra la presenza dell'utente nel database e riconosce le credenziali, l'utente può procedere con l'utilizzo dell'app.



Figura 2.8: Conferma mail

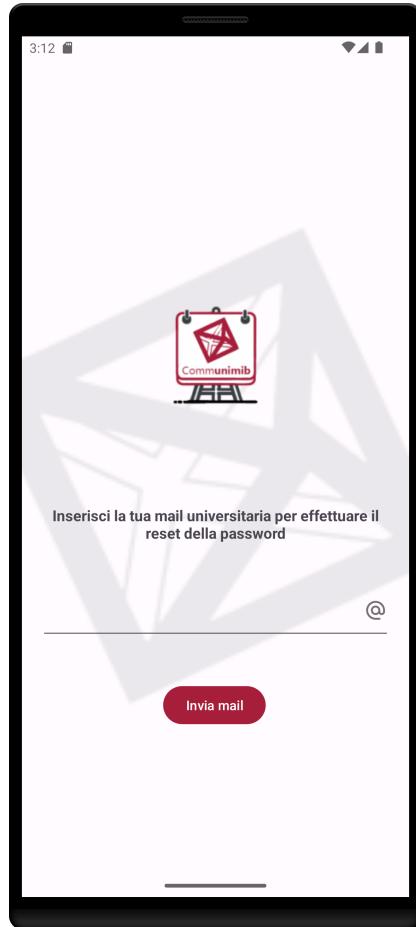


Figura 2.9: Reset password

Verifica e-mail

Ogni sistema di moderna concezione deve misurarsi con il rischio concreto che un utente malevolo possa fornire una e-mail non propria durante la procedura di registrazione: un simile fenomeno può arrecare seri danni al vero proprietario dell’indirizzo. Per questo motivo, Communimib prevede un sistema di verifica dell’indirizzo e-mail basato sulle API messe a disposizione da Firebase.

All’apertura dell’applicazione, il sistema si occupa di controllare lo stato di verifica dell’indirizzo e-mail associato all’account in uso: qualora risultasse ancora non verificato, sarebbe necessario procedere alla suddetta verifica per poter accedere alla homepage dell’app.

La procedura di verifica si compone di alcuni semplici passaggi:

1. Communimib indirizza l’utente verso una schermata che lo invita a controllare la sua casella di posta elettronica

2. Firebase invia un link di verifica all'indirizzo e-mail associato all'account dell'utente
3. L'utente apre il link ricevuto e visualizza sullo schermo un messaggio che conferma il successo della procedura
4. Communimib riscontra l'avvenuta verifica dell'indirizzo e-mail (attraverso le API di Firebase) e consente all'utente di procedere

L'utilizzo di Firebase come sistema backend per l'applicazione assicura diversi benefici, ma comporta tuttavia alcune limitazioni: una di queste è l'assenza di un meccanismo di ascolto e notifica per poter effettuare un periodico controllo sullo stato di verifica dell'indirizzo e-mail. A tal proposito, al fine di evitare macchinosi sistemi di aggiornamento manuale e con l'intenzione di garantire un'esperienza utente quanto più fluida e coerente possibile, è stato implementato un meccanismo di polling che interroga ciclicamente le API di Firebase con lo scopo di controllare se la verifica dell'indirizzo e-mail è avvenuta con successo.

La logica inherente al meccanismo di polling viene interamente gestita dalla classe `UserRepository` (visibile in Figura 2.11) mediante un Service eseguito in background sul dispositivo; a tale scopo, il sistema Android mette a disposizione la classe `ScheduledExecutorService` per poter eseguire task asincroni a cadenza regolare.

Gestione della sessione

Per quanto la procedura di autenticazione possa essere breve e ben studiata, spesso è vissuta dall'utente come un vero e proprio onore; per questo motivo essa non deve essere messa in atto ad ogni nuovo avvio dell'applicazione, ma soltanto quando strettamente necessario.

In tale contesto, Firebase è in grado di gestire automaticamente i controlli legati alla sessione attraverso un meccanismo implementato nel Data-Layer, esponendo successivamente i risultati all'UI-layer. Dato che in alcuni casi l'esecuzione di tale meccanismo può richiedere molto tempo (ad esempio, se si utilizza una connessione lenta), è stato definito ed implementato uno splash screen che viene visualizzato all'avvio dell'applicazione e mostra un'interfaccia di caricamento fino al completamento delle operazioni.

Reset Password

Poiché il processo di controllo della sessione consente all'utente di utilizzare il sistema per un lungo periodo di tempo senza dover inserire le proprie credenziali, può capitare di smarrire o dimenticare la propria password. Per ovviare al problema, Communimib è dotata di un sistema di ripristino della password basato, come per la verifica della e-mail, sulle API di Firebase.

Quando l'utente si reca nella sezione relativa al ripristino della password, l'applicazione richiede l'inserimento dell'indirizzo e-mail associato all'account su cui effettuare la procedura; in seguito, Firebase invia all'indirizzo specificato una mail contenente un link che consente all'utente di scegliere la nuova password.

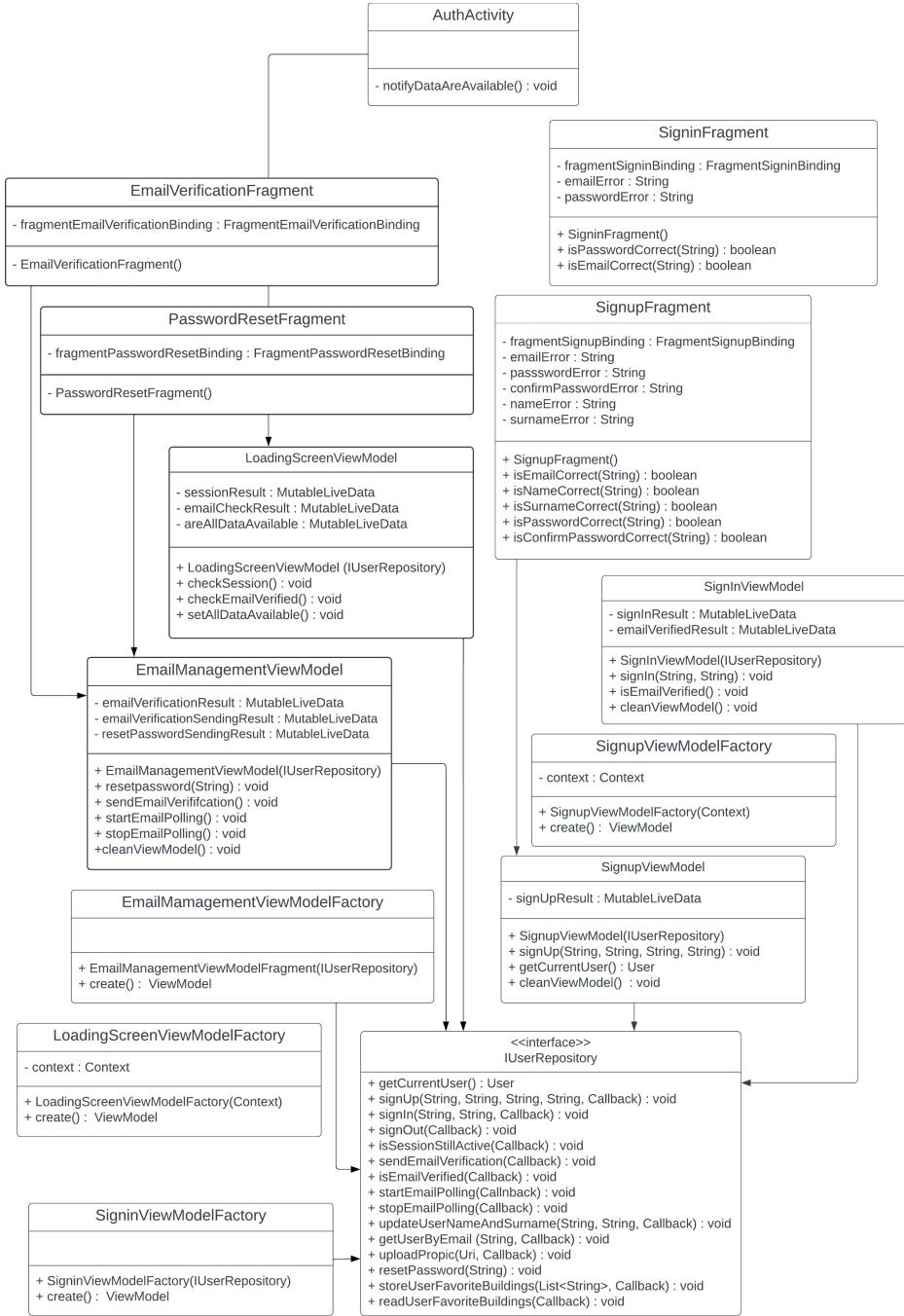


Figura 2.10: Diagramma classi autenticazione - UI layer

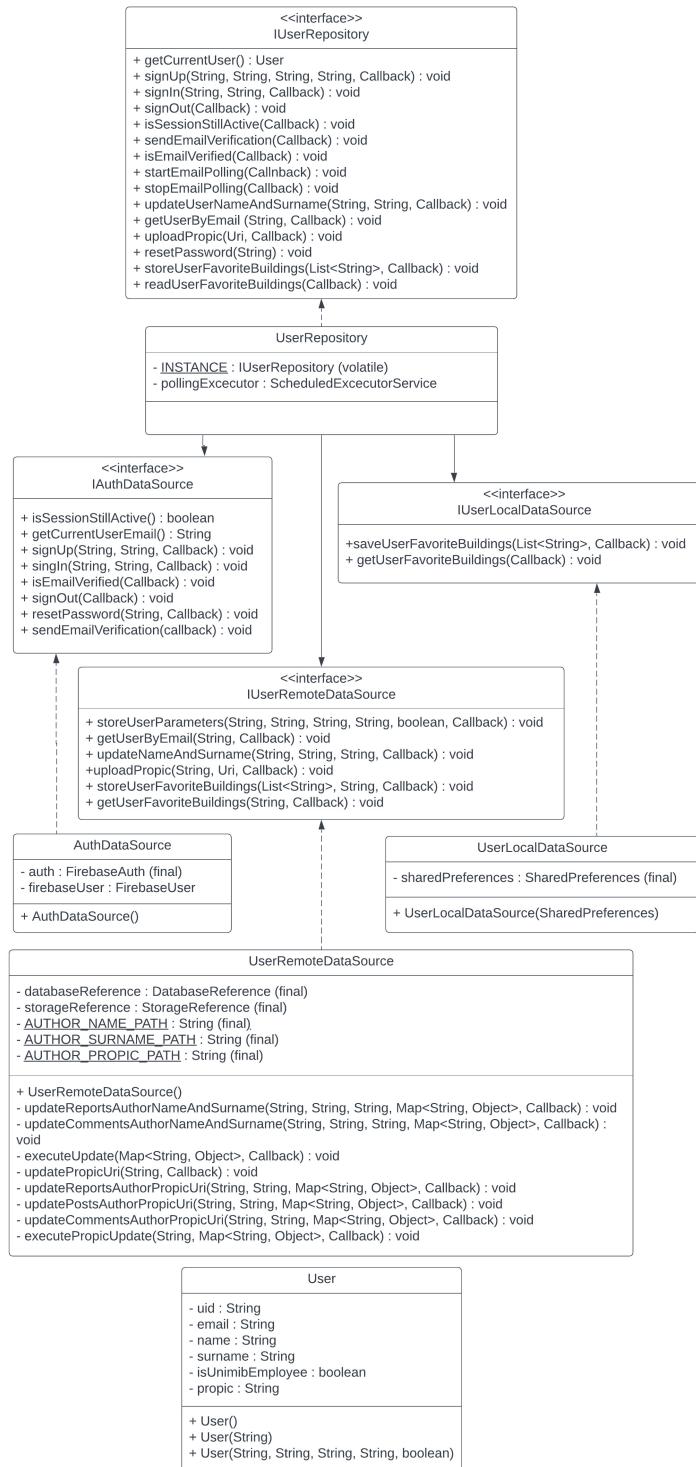


Figura 2.11: Diagramma classi autenticazione - Data layer

2.2.2 Bacheca

Come anticipato nel capitolo precedente, la bacheca costituisce la principale piattaforma di comunicazione messa a disposizione da Communimib: attraverso la pubblicazione di post, essa consente la condivisione di annunci, pensieri e notizie.

L'implementazione di questa funzionalità è stata oggetto di grande impegno da parte del team di sviluppo; dal momento che la bacheca rappresenta uno spazio condiviso tra molti utenti, sia il backend che l'applicazione mobile devono risultare stabili e svolgere efficientemente un lavoro congiunto, al fine di garantire ad ogni utente la migliore esperienza di utilizzo possibile.

Visualizzazione dei post nella bacheca

La visualizzazione della bacheca e dei suoi post ha rappresentato un elemento chiave nello sviluppo di suddetta funzionalità, in quanto è stata ricercata una modalità di visualizzazione che fosse accattivante per l'utente e al contempo chiara e di facile utilizzo (si veda Figura 2.12).

Al fine di essere certi che l'utente possa visualizzare in real-time la pubblicazione di nuovi post o eventuali modifiche ai post esistenti, è stato implementato nelle classi Datasource un sistema di listener tale per cui, ad ogni modifica dei post salvati nel Firebase, le classi Datasource vengono notificate e agiscono di conseguenza, comunicando ai livelli superiori l'avvenuto aggiornamento. In particolare, attraverso l'uso di observer, la schermata di visualizzazione dei post reagisce alle notifiche provenienti dai livelli sottostanti modificando i soli componenti coinvolti nell'aggiornamento, senza dover ricaricare l'intera schermata.

L'interfaccia garantisce all'utente la possibilità di personalizzare la scelta dei post visualizzati attraverso l'utilizzo di un componente RecyclerView a scorrimento orizzontale nel quale sono presenti le diverse categorie. Al tocco di una specifica categoria, la visualizzazione dei post viene modificata dal sistema in modo tale da mostrare a schermo solo i post appartenenti alla categoria selezionata, con l'obiettivo di consentire all'utente un rapido accesso alle informazioni desiderate.

Inoltre, nel caso in cui l'utente voglia accedere direttamente ad uno o più post, può utilizzare una barra di ricerca dedicata: inserendo al suo interno una parola chiave si richiede al sistema di individuare e visualizzare i soli post contenenti tale parola chiave nel titolo o nella descrizione.

Dopo aver scelto la modalità di visualizzazione, che di default è impostata su "Tutti" e pertanto prevede che i post non vengano filtrati per categoria, l'utente potrà scrollare verso l'alto per visualizzare i post.

Per migliorare l'esperienza di utilizzo dell'applicazione, è stata introdotta una funzionalità che prevede la comparsa di un avviso quando vengono effettuate nuove aggiunte alla bacheca (si veda Figura 2.13). Communimib ordina i post in modo tale che i più recenti siano i primi ad essere visualizzati: se l'utente è intento a scrollare la bacheca mentre viene pubblicato un post, egli corre il rischio di non visualizzarlo.



Figura 2.12: Bacheca

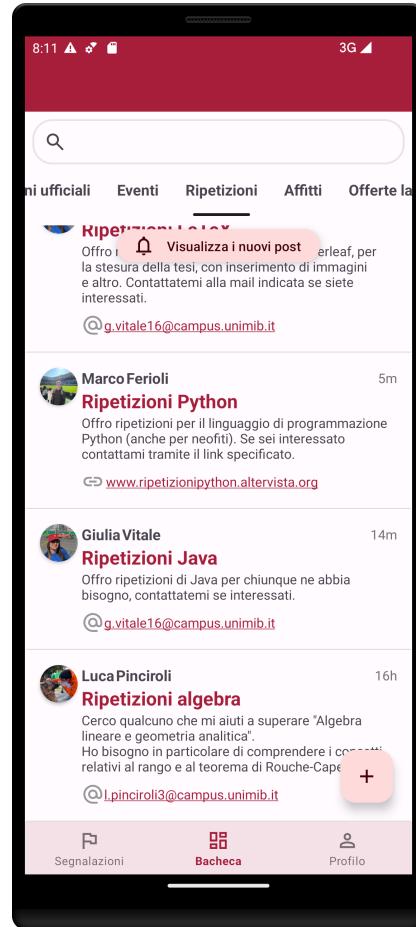


Figura 2.13: Avviso nuovo post

Creazione di un nuovo post

Ogni utente ha la possibilità di creare post (si veda Figura 2.14) e pubblicarli nel sistema affinché risultino visibili e consultabili anche dal resto della community. Per garantire una user-experience ottimale, la schermata che consente la creazione di un post deve essere intuitiva e rapida da compilare; per questo motivo non è richiesta la compilazione di tutti i campi presenti, bensì l'utente è tenuto ad inserire solamente i campi relativi al titolo, alla descrizione ed alla categoria del post.

Oltre alle caratteristiche precedentemente esposte, la creazione di un post deve garantire un grado di personalizzazione che vada oltre l'inserimento di un semplice blocco di testo: a tal proposito è stato sviluppato ed utilizzato un componente che consente il caricamento di un insieme di immagini da associare al post. Per mettere in atto questa operazione è stato fatto uso dell'apposita classe `PickVisualMediaRequest`, la quale effettua un'operazione di `Intent` verso la galleria immagini del dispositivo dove l'utente può

selezionare le foto che preferisce. Il risultato dell'operazione viene catturato utilizzando un `ActivityResultLauncher` ed inserito in un componente grafico, che svolge la funzione di Slider; quest'ultimo viene gestito dalla libreria esterna `ImageSlideShow` [1] importata utilizzando il sistema di gestione delle dipendenze Gradle.

Infine, l'utente ha la possibilità di completare il post aggiungendo un indirizzo e-mail di contatto ed un link; il primo rappresenta un recapito utile per contattare l'autore del post, mentre il secondo può essere sfruttato a seconda delle situazioni (un link può essere impiegato per condividere siti web, sondaggi, inviti a gruppi whatsapp, ecc.).

Quando l'utente conferma la pubblicazione del post, viene effettuata l'operazione di validazione dei parametri inseriti; se la procedura va a buon fine, il post viene registrato nel Firebase Realtime Database. Dal momento che quest'ultimo non supporta la memorizzazione delle immagini, viene quindi utilizzato Firebase Storage per effettuare tale operazione.



Figura 2.14: Creazione post

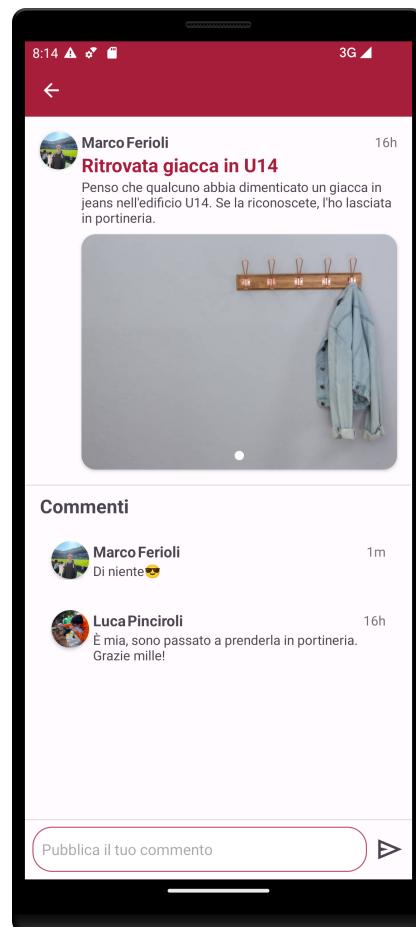


Figura 2.15: Commenti

Commenti

Sebbene i post rappresentino il principale mezzo di confronto tra gli utenti di Communimib, non è rilevante soltanto la loro pubblicazione, ma anche la possibilità di commentare quelli già esistenti. Per questo motivo, l'applicazione comprende un sistema che permette agli utenti di scrivere commenti associati ai post (si veda Figura 2.15): essi possono essere utilizzati non solo per esprimere opinioni sull'argomento trattato, ma anche come mezzo di contatto con l'autore (soprattutto se non ha fornito l'indirizzo e-mail di contatto).

Per visualizzare un singolo post e contestualmente scrivere un commento, è sufficiente selezionare il post desiderato con un tap: mediante l'ausilio del plugin SafeArgs, le informazioni relative al post selezionato vengono raccolte dal sistema e presentate su una schermata dedicata, la quale non solo mostra il post nella sua interezza, ma consente inoltre di visionarne tutti i commenti associati.

Esattamente come accade per la visualizzazione della bacheca, quando viene pubblicato un commento relativo ad un post, questo viene mostrato in tempo reale grazie alle funzionalità messe a disposizione da Firebase.

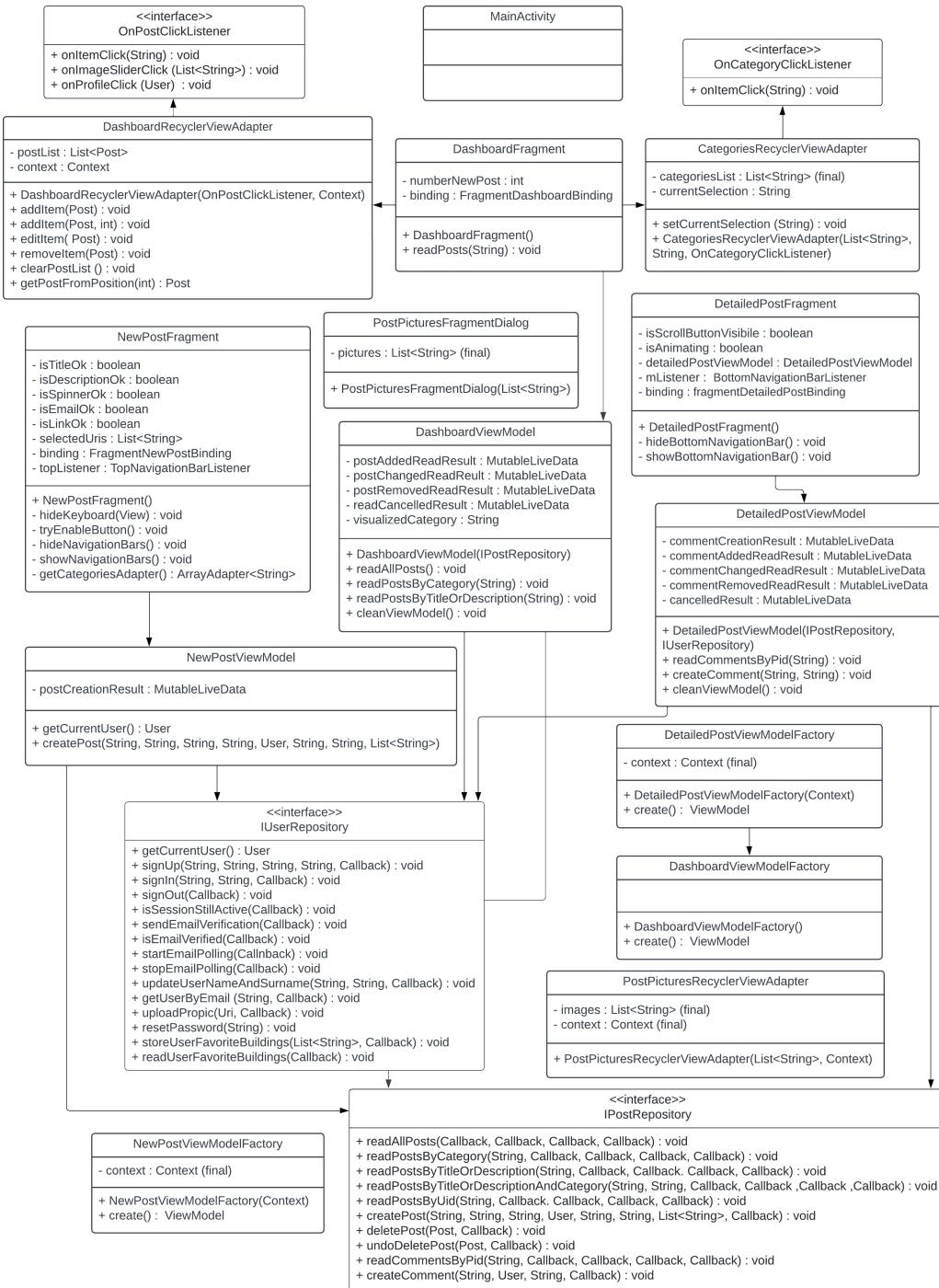


Figura 2.16: Diagramma classi bacheca - UI layer

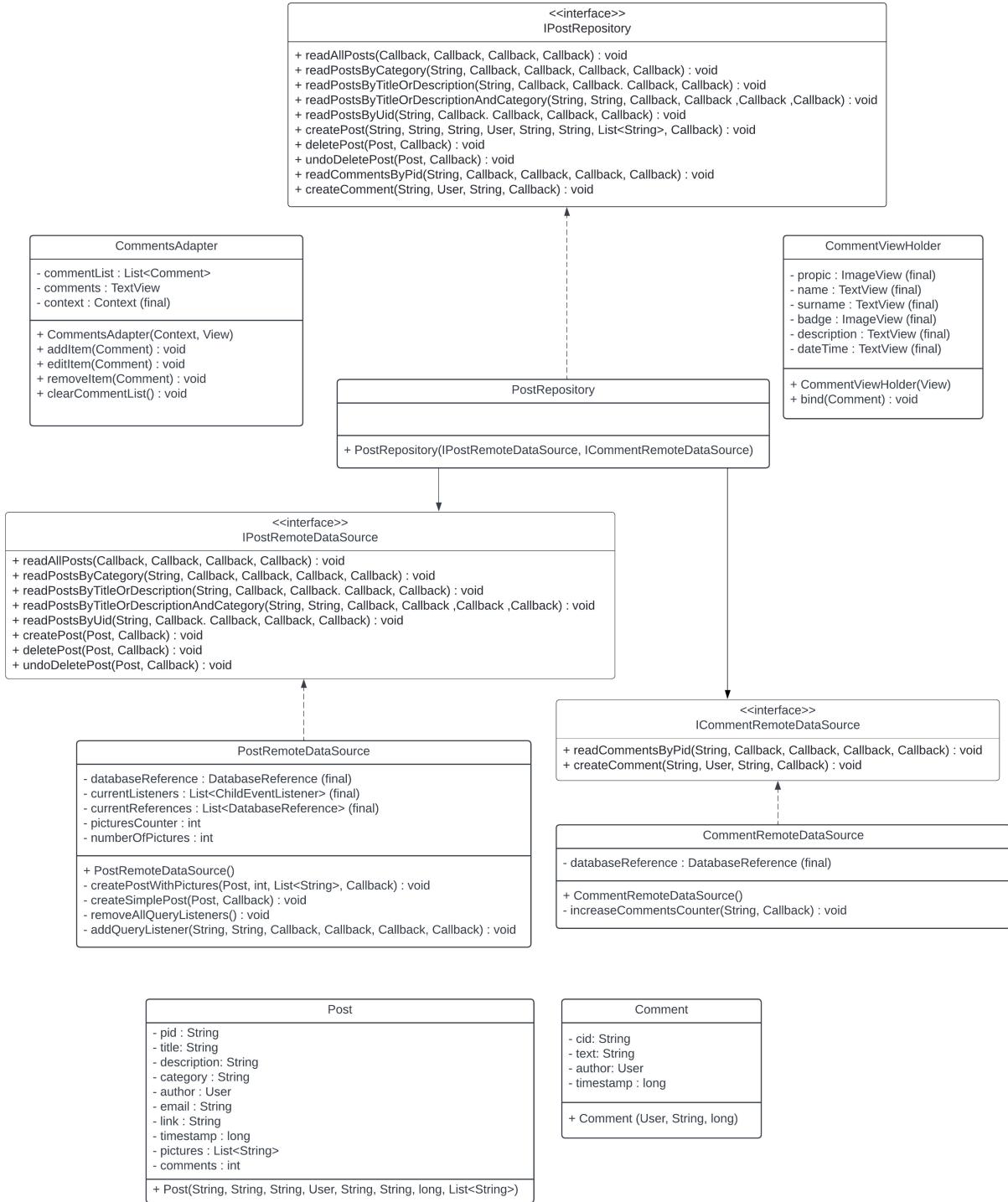


Figura 2.17: Diagramma classi bacheca - Data layer

2.2.3 Segnalazioni

Il sistema di segnalazione di Communimib nasce per rafforzare la collaborazione tra corpo studentesco e personale universitario: nel rispetto del ruolo di ciascuna controparte, l'utilizzo di tale strumento può portare ad un sensibile miglioramento nella vita d'ateneo.

Lo sviluppo del sistema è stato condotto nell'ottica della sua duplice modalità di utilizzo: quella degli addetti ai lavori, i quali hanno la necessità di identificare rapidamente i problemi aperti e procedere ad una celere risoluzione, e quella degli studenti, i quali necessitano di un mezzo per comunicare in maniera semplice e puntuale gli inconvenienti che possono verificarsi negli edifici.

Visualizzazione delle segnalazioni

La visualizzazione delle segnalazioni (si veda Figura 2.18) è stata gestita in modo diverso rispetto al sistema utilizzato nella bacheca: in questo caso il team ha ritenuto che potesse essere utile raggruppare le segnalazioni secondo l'edificio a cui queste si riferiscono, per rendere la visualizzazione più intuitiva, lineare e chiara.

A tale scopo è stato implementato un componente RecyclerView innestato, che visualizza verticalmente gli edifici. Per ciascun edificio, mediante l'uso di un componente RecyclerView a scrollamento orizzontale, vengono mostrate le segnalazioni corrispondenti secondo l'ordine di pubblicazione.

La RecyclerView innestata è stata realizzata implementando due differenti adapter:

- **ReportsHorizontalRecyclerViewAdapter:** è l'adapter usato per la RecyclerView con scroll orizzontale, nel quale ogni elemento della RecyclerView rappresenta una segnalazione.
- **ReportMainRecyclerViewAdapter:** è invece l'adapter usato per la RecyclerView con scroll verticale; in questo caso ogni elemento è costituito da un componente di testo, nel quale viene inserito il nome dell'edificio, e da un componente RecyclerView.

Poiché l'interrogazione a Firebase ha come risultato l'intero insieme di segnalazioni memorizzate nel sistema, l'associazione tra edificio e segnalazioni ad esso riferite viene effettuata in un secondo momento con la seguente modalità:

1. Per ogni edificio (memorizzato in locale sul dispositivo) viene istanziato un oggetto **ReportsHorizontalRecyclerViewAdapter** contenente le segnalazioni riferite a tale edificio.
2. Per ciascuna coppia data dall'edificio e dal rispettivo adapter, viene istanziato un oggetto **BuildingReport**.

Gli oggetti **BuildingReport** vengono infine aggiunti ad una lista utilizzata dall'adapter **ReportMainRecyclerViewAdapter** per costituire la schermata.

Inoltre, al fine di rendere l'interfaccia più funzionale, non vengono mostrati gli edifici ai quali non sono associate segnalazioni.

Per consentire all'utente di consultare in dettaglio la segnalazione, cliccando su quest'ultima è possibile visualizzarla a schermo intero. Come nella bache-

ca, i dati relativi alla segnalazione vengono trasmessi alla schermata dedicata attraverso l'uso del plugin SafeArgs.

Anche in questo caso, al fine di consentire una consultazione più rapida, è stata implementata una barra di ricerca che permette di trovare rapidamente una segnalazione inserendo una parola chiave presente nel suo titolo o nella sua descrizione. Questa funzionalità può rivelarsi particolarmente utile per gli addetti ai lavori incaricati a risolvere problemi specifici.

In conclusione, occorre specificare che il meccanismo utilizzato per l'aggiornamento real-time delle segnalazioni è il medesimo usato nella bacheca.

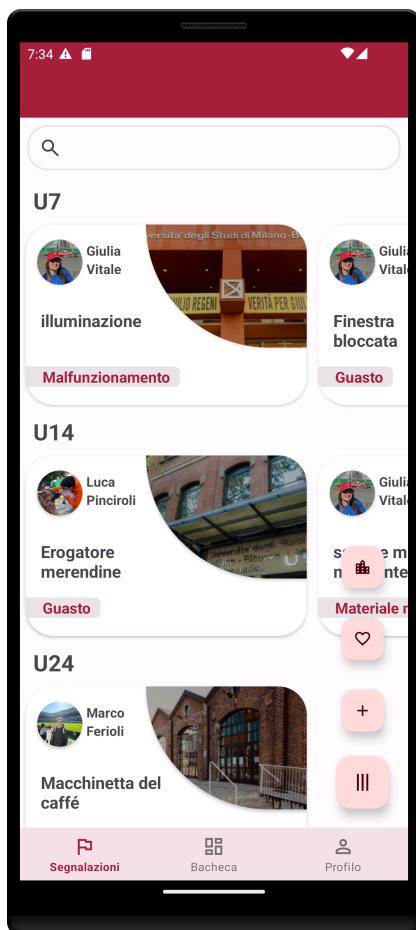


Figura 2.18: Segnalazioni

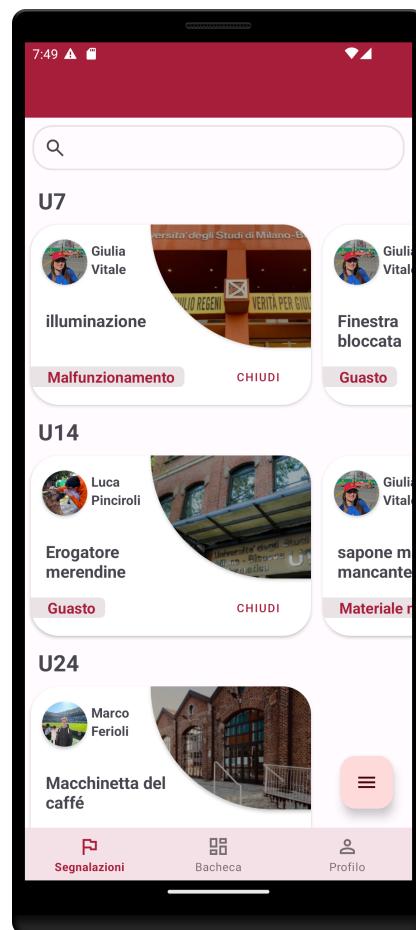


Figura 2.19: UI dipendente

Filtri ed edifici preferiti

Un utente deve avere la possibilità di selezionare solo un gruppo di segnalazioni tra quelle presenti nel sistema. Per soddisfare tale requisito è stato fatto uso di componenti chiamati **DialogFragment** (comunemente noti come "dialog"); essi vengono messi a disposizione dalla libreria Android e assumono un comportamento simile ai normali **Fragment** dal punto di vista programmatico e del ciclo di vita, presentando tuttavia alcune differenze sostanziali:

- **Due pulsanti per azioni distinte.** Un **DialogFragment** fornisce sempre due pulsanti associati a procedure differenti: l'operazione di annullamento, che consiste solitamente nell'invocazione del metodo **dismiss()**, il quale rimuove il dialog dalla schermata senza compiere ulteriori azioni, e l'operazione di conferma che permette di apportare delle modifiche alla schermata che ha originato il dialog.
- **Background trasparente e dimensioni limitate.** Un **DialogFragment** ha un background trasparente e occupa solo una porzione della schermata, consentendo all'utente di vedere la schermata sottostante anche quando il dialog è attivo.

Sulla base di questa premessa, di seguito sono descritti i due sistemi implementati per effettuare l'operazione di filtraggio dei dati:

1. **Filtri generali** (si veda Figura 2.21). L'utente ha la possibilità di scegliere un insieme di edifici, che vengono incapsulati in una **List** ed inviati al data layer. Qui viene effettuata una query verso Firebase Realtime Database per ottenere solo ed esclusivamente le segnalazioni associate agli edifici selezionati. Successivamente, queste segnalazioni vengono posizionate adeguatamente nella schermata sfruttando le proprietà delle **RecyclerView** descritte precedentemente.
2. **Edifici preferiti** (si veda Figura 2.20). In questa sezione l'utente può scegliere un insieme di edifici da eleggere come preferiti tra quelli registrati nell'applicazione. L'obiettivo di questo meccanismo è garantire a chi usa l'applicativo una migliore user-experience mediante la personalizzazione dei contenuti consultabili. Il sistema è stato costruito per fare in modo che, all'apertura dell'applicativo, vengano mostrate solo le segnalazioni relative agli edifici preferiti; nell'implementazione di questo meccanismo è stata utilizzata la classe **SharedPreferences** per salvare localmente la lista degli edifici preferiti. Inoltre, utilizzando Firebase Realtime Database, è stato implementato un sistema che permette di condividere gli edifici preferiti tra i dispositivi mobili che hanno effettuato l'accesso alla piattaforma con lo stesso account. La strategia applicata permette di leggere localmente i dati con lo scopo di evitare chiamate superflue verso il backend remoto, mantenendo comunque sincronizzati gli edifici preferiti su dispositivi differenti.



Figura 2.20: Selezione preferiti

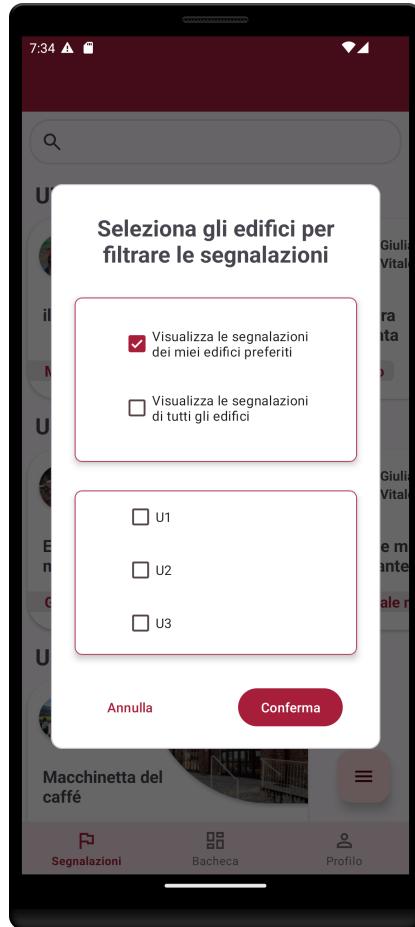


Figura 2.21: Selezione filtri

Creazione e chiusura di una segnalazione

Due operazioni fondamentali per il funzionamento del sistema di segnalazione sono la creazione e la chiusura. Indipendentemente dal ruolo ricoperto (personale o studente), ogni utente ha la possibilità di creare segnalazioni; ciò non vale per l'operazione di chiusura, di fatto attuabile soltanto dai membri del personale di ateneo.

La creazione di una nuova segnalazione (si veda Figura 2.23) può essere effettuata dall'utente mediante un apposito dialog, in cui vengono richieste le sole informazioni che racchiudono un conciso riassunto della problematica: un titolo, una breve descrizione del problema, la categoria di appartenenza (guasto, malfunzionamento, materiale mancante, ecc.) e l'edificio a cui fa riferimento. L'operazione di creazione innesta i listener Firebase posti in ascolto sui dispositivi degli altri utenti, manifestando l'immediata comparsa della segnalazione all'interno del sistema.

Dal momento che una delle due operazioni chiave è riservata al solo personale universitario, l'interfaccia grafica assume un comportamento differente in base alla categoria di utente che fa uso dell'applicazione: il pulsante adibito alla chiusura della segnalazione risulta visibile ai soli dipendenti di ateneo (si veda Figura 2.19).

Dal punto di vista implementativo la chiusura di una segnalazione equivale alla sua eliminazione, pertanto comporta la rimozione del rispettivo nodo da Firebase Realtime Database.



Figura 2.22: Segnalazione dettagliata



Figura 2.23: Creazione segnalazione

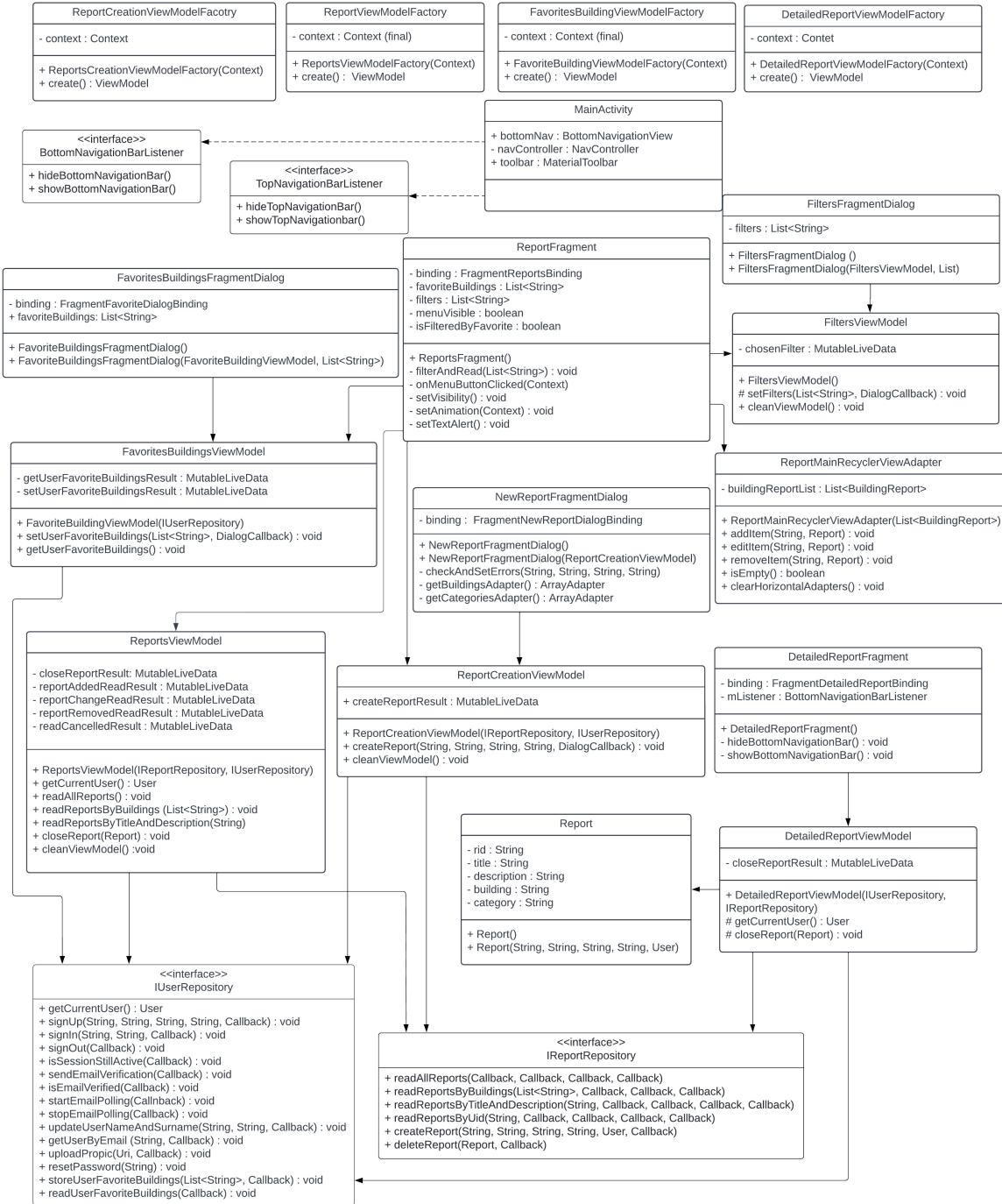


Figura 2.24: Diagramma classi segnalazioni - UI layer

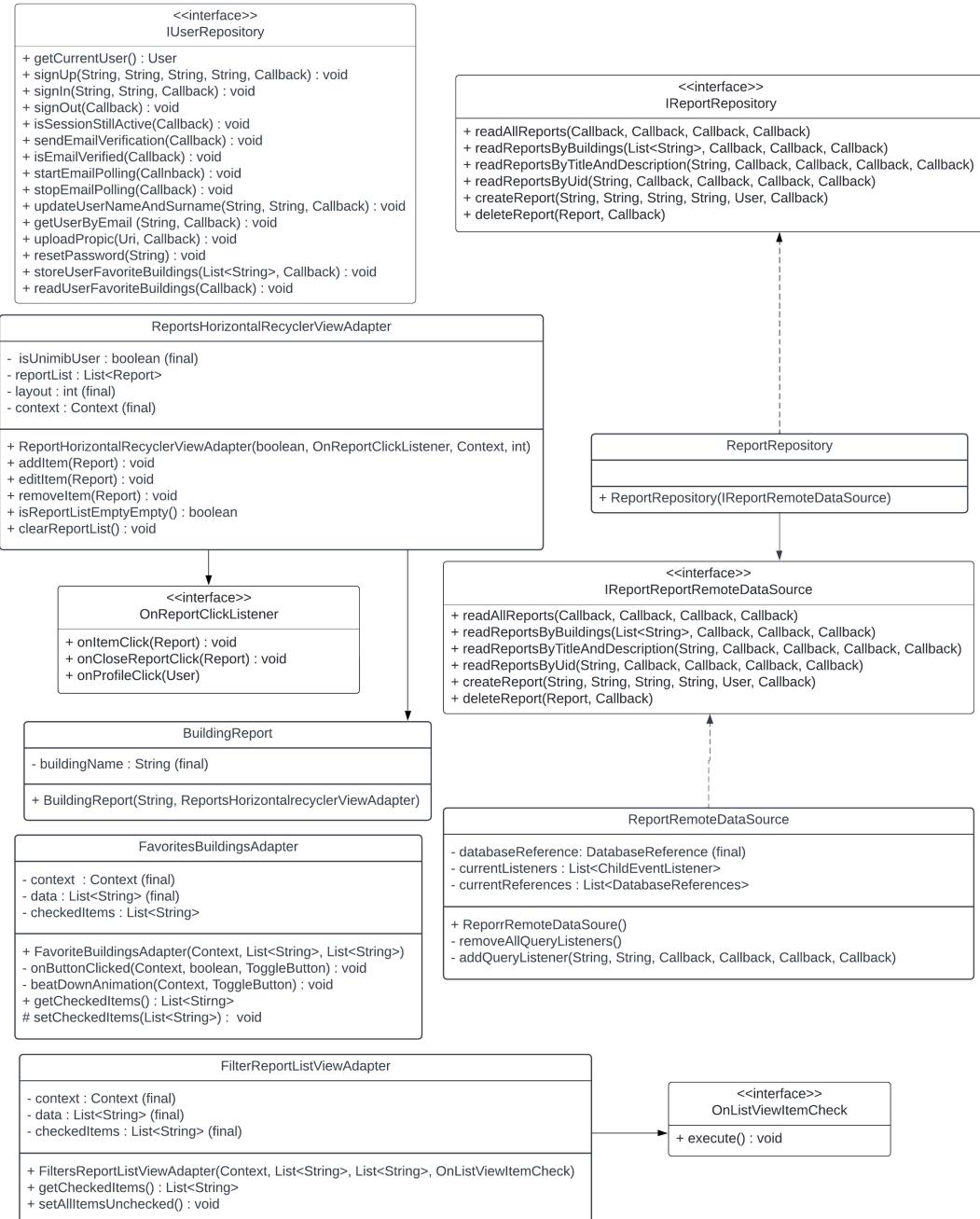


Figura 2.25: Diagramma classi segnalazioni - Data layer

2.2.4 Profilo

L'ultima macro funzionalità dell'applicazione è rappresentata dal profilo. Questa offre all'utente la possibilità di gestire il proprio account (si veda Figura 2.26), per apportare modifiche alle informazioni personali e visualizzare le segnalazioni o i post che ha pubblicato. La medesima schermata, a meno di alcuni particolari, viene utilizzata anche per la visualizzazione dei profili degli altri utenti: cliccando sui dati di un utente (nome, cognome o immagine profilo) presenti su una segnalazione o su un post, viene mostrato a schermo il suo profilo, potendo così consultare i suoi post e le segnalazioni da lui pubblicate.

Al fine di garantire maggiore riutilizzo, per la visualizzazione delle segnalazioni e dei post sono state usate le stesse classi Adapter e ViewHolder utilizzate nelle sezioni relative alla bacheca ed alle segnalazioni.

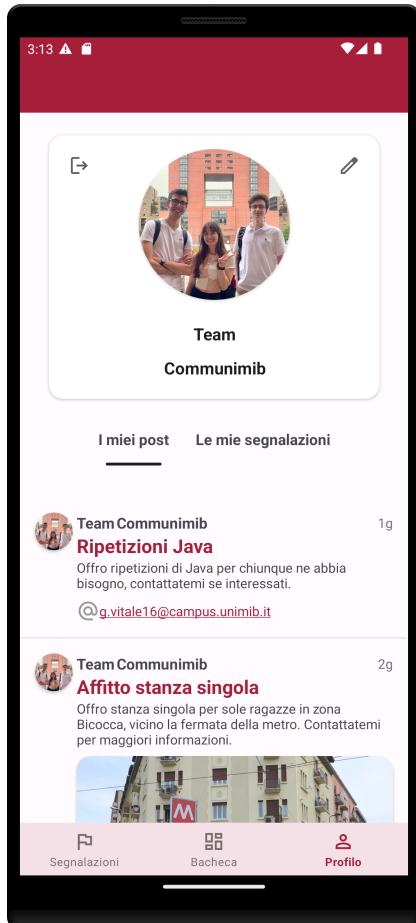


Figura 2.26: Profilo

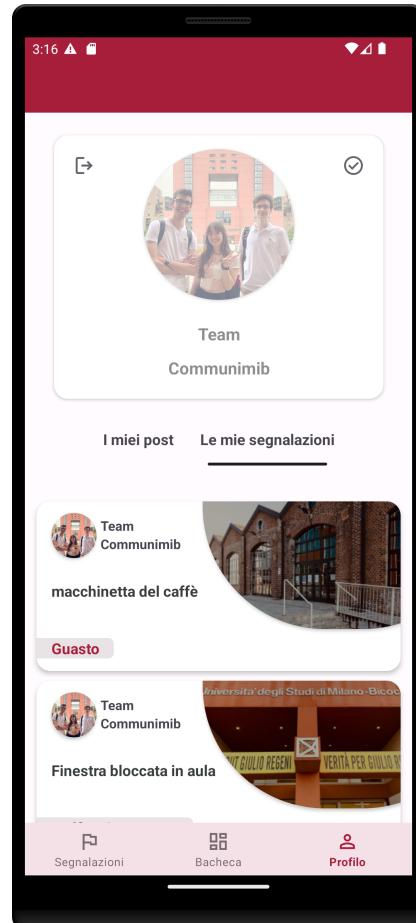


Figura 2.27: Modifica profilo

Immagine e parametri utente

Con lo scopo di fornire una maggiore libertà di personalizzazione, l'utente ha la possibilità di modificare arbitrariamente alcuni parametri che sono associati al profilo personale: nome, cognome ed immagine profilo (si veda Figura 2.27). Per ragioni estetiche, il team ha deciso di adottare una forma circolare per le immagini di profilo, rendendo di fatto necessaria l'implementazione di un sistema che consenta all'utente di selezionare solo una porzione rotonda dell'immagine che intende impostare sul suo profilo.

Questo requisito è stato gestito utilizzando il componente `uCrop` [9], importato dall'omonima libreria esterna: tale componente effettua un'operazione di Intent esplicito verso un'Activity definita all'interno della libreria, consentendo all'utente di selezionare un'immagine dalla sua galleria e modificarne dimensione ed inquadratura. Anche in questo caso viene sfruttato un `ActivityResultLauncher` per catturare il risultato dell'operazione di Intent (che rappresenta l'immagine modificata) e posizionarlo correttamente all'interno del rispettivo componente della schermata.

Cancellazione post

A differenza delle segnalazioni, le quali hanno la possibilità di essere chiuse (e quindi eliminate) dal personale alla risoluzione del problema corrispondente, i post possono rimanere permanentemente memorizzati all'interno del sistema; questo comportamento potrebbe non essere sempre gradito all'utente, il quale potrebbe manifestare l'esigenza di rimuovere uno o più post dal suo profilo.

Il layout visuale con cui Communimib mostra i post sulla propria interfaccia non è stato delineato per prevedere la presenza di un pulsante di eliminazione, poiché risulta importante che la UI sia snella ed intuitiva al fine di garantire semplicità di utilizzo; è stato quindi necessario adottare una diversa soluzione grafica per implementare la cancellazione dei post all'interno del profilo utente.

Un meccanismo elegante, moderno ed accattivante per consentire all'utente la cancellazione di un post è quello dello swipe; quando il post viene fatto scorrere verso il lato sinistro dello schermo, lo sfondo si colora di rosso e viene mostrata un'icona raffigurante un cestino. Allo scopo di realizzare questo comportamento dal punto di vista implementativo, è stato fatto uso del componente Android `ItemTouchHelper` per catturare gli eventi scatenati dall'utente sugli elementi della RecyclerView, tra cui l'operazione di swipe: quando l'utente scorre un post per procedere alla sua eliminazione, l'oggetto `ItemTouchHelper` innesca la procedura di rimozione dal sistema.

Dal momento che l'operazione di cancellazione può anche essere messa in atto per errore, quando un post viene cancellato l'applicazione mostra a schermo una Snackbar contenente un pulsante per annullare l'eliminazione; nel momento in cui la procedura di rimozione viene messa in atto, il post eliminato viene salvato localmente per poter essere eventualmente ripristinato mediante la pressione dell'apposito pulsante.

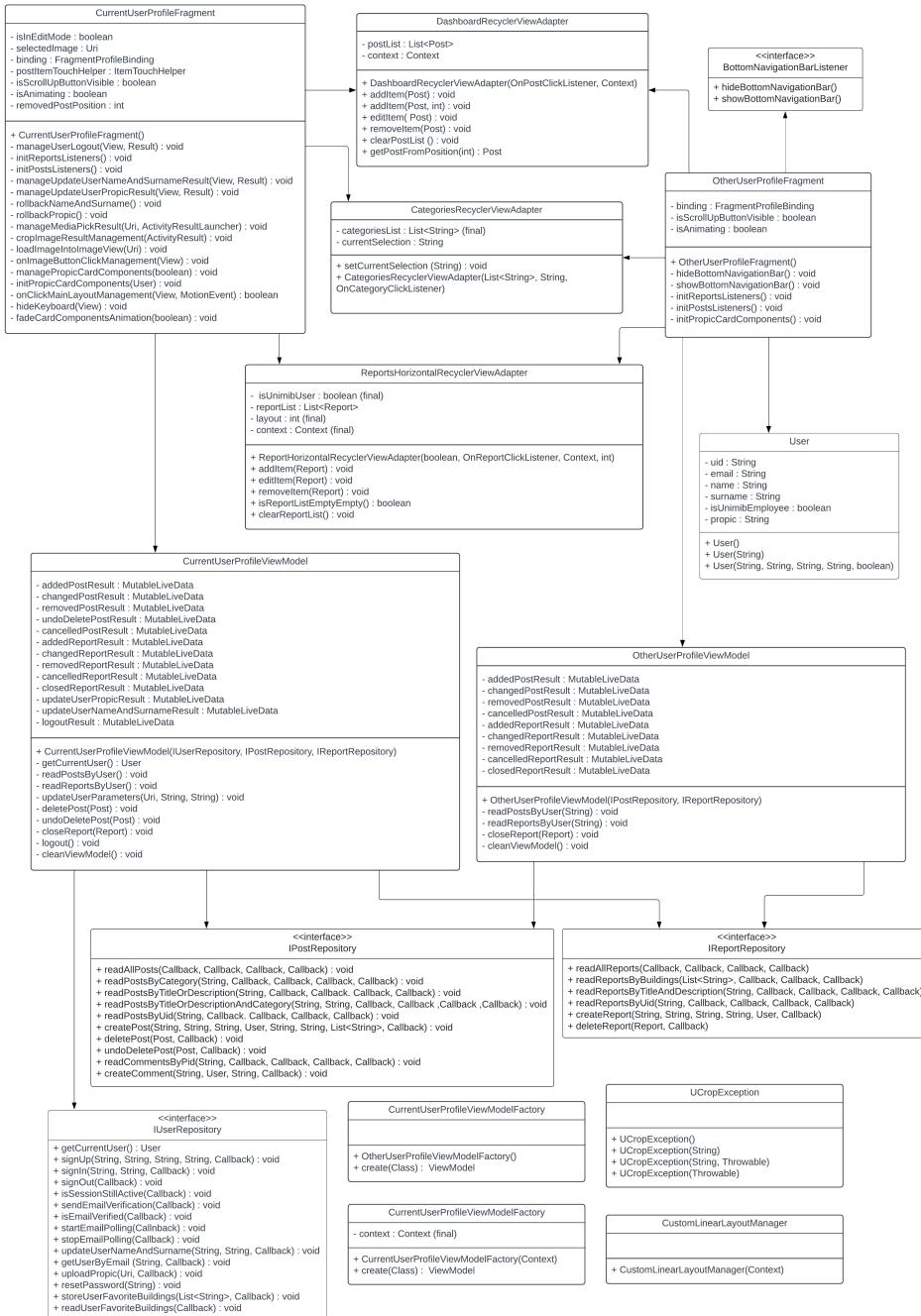


Figura 2.28: Diagramma classi profilo - UI layer

2.2.5 Dettagli relativi all'uso di Firebase

Per concludere la trattazione relativa alle caratteristiche implementative di Communimib, è opportuno descrivere come è stato fatto uso di Firebase per la realizzazione delle varie funzionalità dell'applicazione.

Firebase [3] è un servizio cloud offerto da Google che mette a disposizione un'ampia suite di strumenti a supporto degli sviluppatori mobile: comprende un sistema di gestione dell'autenticazione server-side, strumenti di analisi sul traffico dell'applicazione, uno spazio di archiviazione remoto, un servizio di AI generativa, ecc.

I servizi su cui Communimib si basa sono:

- **Firebase Authentication**, per la gestione degli account e dei processi di autenticazione.
- **Firebase Realtime Database**, per la memorizzazione dei dati e l'aggiornamento in tempo reale dei contenuti.
- **Firebase Storage**, per il salvataggio delle immagini caricate dagli utenti.

Firebase Authentication

Un'applicazione che mette a disposizione di ognuno la possibilità di creare la propria identità "virtuale" ha l'obbligo di garantire un adeguato livello di sicurezza: è per questa ragione che il sistema di autenticazione si basa su un servizio robusto ed affidabile come Firebase Authentication.

Durante la procedura di registrazione di Communimib vengono raccolti diversi dati relativi all'utente: tra questi sono presenti indirizzo e-mail e password, i quali vengono trasmessi a Firebase Authentication per dare origine ad un nuovo account avente il proprio identificativo univoco ed il proprio token di sessione; le informazioni restanti vengono invece memorizzate in Firebase Realtime Database. Con la creazione di un nuovo account mediante e-mail e password, l'applicazione delega a Firebase Authentication il compito di mantenere aggiornato il registro degli account e gestire i flussi di autenticazione sui diversi dispositivi.

Grazie ad un meccanismo basato su token, il servizio è inoltre capace di gestire le sessioni di utilizzo dell'applicazione affinché non sia necessario eseguire una nuova procedura di autenticazione ogni volta che l'app viene avviata.

L'utilizzo di Firebase Authentication ha permesso l'implementazione di un meccanismo di autenticazione solido ma flessibile, adatto ad essere utilizzato in un contesto complesso e dinamico come quello in cui Communimib si colloca.

Firebase Realtime Database

Le applicazioni basate sulla comunicazione tra utenti pongono la velocità di aggiornamento dei contenuti al primo posto tra i requisiti da rispettare: è per questa ragione che l'implementazione di Communimib ha richiesto l'impiego di un sistema backend che fosse rapido e capace di sostenere un buon carico di lavoro simultaneo. Firebase Realtime Database si inserisce perfettamente in tale

conto, essendo pensato per offrire un meccanismo di aggiornamento dei dati in tempo reale su tutti i dispositivi che ne fanno uso; attraverso l'utilizzo di appositi listener, ciascun dispositivo può mettersi in ascolto e rimanere costantemente aggiornato qualora i dati memorizzati nel database dovessero subire delle modifiche.

Un'ulteriore caratteristica di tale servizio è la gestione automatica dei momenti di discontinuità della connessione ad internet: i dispositivi mobili non hanno sempre a disposizione una rete internet stabile a cui rimanere connessi in maniera costante; di conseguenza, Firebase Realtime Database effettua automaticamente operazioni di caching locale che consentono la temporanea prosecuzione delle letture e scritture sul database anche in assenza di una connessione di rete.

Il database messo a disposizione da Firebase Realtime Database è di tipo non relazionale e prevede la memorizzazione dei dati in formato JSON, richiedendo così un modus operandi ben diverso da quello richiesto dai classici database relazionali: la base di dati viene progettata ed organizzata con l'obiettivo di essere facilmente espandibile in futuro, abbandonando il rigore e la rigidità dei modelli relazionali.

L'impiego di un database non relazionale implica, al fine di poterne effettivamente sfruttare i benefici, la necessità di compiere passi di denormalizzazione: ciò significa inserire ridondanza all'interno della base di dati con lo scopo di facilitare le operazioni di interrogazione. Con l'introduzione di ridondanza nei dati è necessario prestare particolare attenzione all'integrità ed alla coerenza di questi ultimi: se la stessa informazione è duplicata in punti diversi del database, si manifesta il rischio concreto di generare punti di inconsistenza che possono portare a malfunzionamenti nell'applicazione.

Sulla base di queste premesse è chiaro che le operazioni di aggiornamento devono essere svolte con particolare cura ed in maniera atomica, così da evitare interruzioni che potrebbero lasciare i dati in uno stato inconsistente; nel caso di Communimib, tutti questi concetti si sono rivelati fondamentali durante lo sviluppo di funzionalità chiave come la modifica del profilo utente (nome, cognome ed immagine profilo), la creazione di un nuovo post e l'apertura di una nuova segnalazione.

Firestore Storage

Dal momento che Firebase Realtime Database non consente la memorizzazione di file multimediali, il processo di creazione di Communimib ha richiesto l'utilizzo di un ulteriore servizio appartenente alla suite di Firebase: Firebase Storage.

Il suo impiego è stato fondamentale per fare in modo che l'applicazione potesse gestire tutte le immagini fornite dagli utenti, sia quelle associate ai profili che quelle indicate ai post. Le operazioni di upload e download delle immagini si basano sull'utilizzo di URI per la loro identificazione univoca; dal momento che Firebase Realtime Database è in grado di contenere soltanto informazioni testuali, è stato possibile associare le immagini ai corrispondenti profili e post mediante l'utilizzo dei rispettivi URI.

Capitolo 3

Progetto di testing

3.1 Il ruolo dei test nello sviluppo Android

L’operazione di testing è un passo fondamentale nel processo di sviluppo di un qualsiasi tipo di applicazione mobile o sistema informatico, perché permette di verificare e validare il corretto funzionamento dei componenti implementati prima del loro rilascio.

Sebbene i test manuali effettuati dagli sviluppatori possano essere particolarmente efficaci per verificare il funzionamento di quanto implementato, è altresì importante scrivere dei test automatici più veloci e ripetibili, che permettano di identificare rapidamente bug e malfunzionamenti nel sistema. Per questo motivo, durante il processo di realizzazione di Communimib sono state sviluppate diverse batterie di test automatici con l’ausilio della libreria JUnit4 [5]. In particolare, è stato fatto uso di test appartenenti a due diverse tipologie:

1. **unità.** Permettono di verificare il funzionamento una porzione molto piccola dell’applicazione, come una classe o, in casi specifici, un metodo. In riferimento al capitolo precedente, l’architettura sviluppata secondo una struttura a livelli facilita la scrittura di questa tipologia di test, poiché permette di testare singolarmente ciascun componente.
2. **UI.** Permettono di validare il funzionamento dell’interfaccia utente e generalmente consistono nell’esecuzione di un’activity in cui vengono simulati dei comandi, i quali rappresentano delle azioni generate dall’utente, con lo scopo di verificare il corretto funzionamento dell’interfaccia.

Lo sviluppo dell’applicazione è stato suddiviso in quattro iterazioni (una per macro funzionalità) e al termine di ciascuna è stato effettuato il testing di quanto implementato; ciò ha permesso di validare il funzionamento delle classi prodotte, permettendo il loro riutilizzo nelle successive fasi implementative.

3.2 Test di unità

I test di unità isolano una singola porzione di codice testandola indipendentemente dal resto del sistema, al fine di verificare che essa produca l’output

previsto a partire da un dato input. Per l'implementazione di tali test è stato utilizzato il framework Mockito [6], il quale consente di emulare il comportamento di una classe per testare i metodi che interagiscono con essa.

I test di unità sono stati scritti per le classi Repository e ViewModel; al contrario, le classi Datasource non sono state direttamente testate poiché interagiscono direttamente con Firebase e la documentazione di Android sconsiglia il testing di librerie esterne, il cui funzionamento è già stato validato.

Nelle classi Repository è stato seguito uno standard per la stesura dei test; poiché tali classi si occupano di effettuare delle chiamate alle classi Datasource, per testare che il comportamento dei metodi rispecchiasse quello atteso sono stati seguiti i seguenti passaggi:

- **setUp() della classe di test:** questo metodo, etichettato con l'annotazione `@Before` così che sia il primo metodo ad essere eseguito, prevede che al suo interno vengano istanziati tutti gli oggetti utili per il testing, come il Repository da testare e i `mock()` delle classi Datasource richiamate da esso.
- Per ogni metodo:
 1. **Invocazione metodo doAnswer().when():** questo metodo permette definire il comportamento della classe Mock, precisando i valori che devono essere assegnati alle callback utilizzate.
 2. **Chiamata al metodo da testare:** dopo aver definito i valori restituiti dalla classe mock, viene effettuata la chiamata al metodo interessato e si testa che i valori reali siano uguali ai valori specificati.
 3. **Invocazione metodo verify():** infine, si testa che il metodo esegua la chiamata alla classe Mock.

Il testing delle classi ViewModel segue gli stessi step precedentemente elencati, ma a differenza del testing delle classi Repository, per testare che i valori presenti nei LiveData coincidessero con i valori attesi, è stata impiegata in ogni metodo un'apposita classe `LiveDataTestUtil`. Essa mette a disposizione un metodo `getOrAwaitValue` che consente l'osservazione dei valori contenuti nei LiveData durante i test.

Dal momento che i risultati inseriti nei LiveData sono generalmente originati da operazioni asincrone, il metodo `getOrAwaitValue` osserva il LiveData ed attende un suo aggiornamento fino alla scadenza di un periodo di timeout (attualmente impostato a 3 secondi); appena il LiveData viene aggiornato, il suo valore viene restituito dal metodo. Se durante il periodo di osservazione non dovesse essere avvenuto alcun aggiornamento, il metodo `getOrAwaitValue` restituirebbe un'eccezione.

Infine, i valori presenti all'interno dei vari LiveData (e restituiti dal metodo `getOrAwaitValue`) vengono confrontati con i valori attesi.

3.3 Test UI

I test UI sono stati scritti utilizzando la libreria Espresso [2] (in combinazione con JUnit4), messa a disposizione da Android, per validare il funzionamento

dell’interfaccia grafica definita dalle Activity e Fragment. Questa tipologia di test, a differenza di quelli di unità, necessita di un ambiente di esecuzione complesso in cui sfruttare le API del sistema operativo Android; essi vengono infatti eseguiti utilizzando l’emulatore.

Al fine di mettere in atto una corretta esecuzione dei test UI è necessario specificare l’Activity che il sistema deve utilizzare come ”contenitore” per l’esecuzione dei casi di test. Questa operazione può essere effettuata secondo due modalità differenti:

- **Utilizzo della classe ActivityScenarioRule.** Questo meccanismo può essere applicato solo ed esclusivamente nei casi in cui non si debba eseguire alcuna operazione prima dell’avvio dell’Activity scelta. In questo caso è sufficiente etichettare l’oggetto di tipo `ActivityScenarioRule` utilizzando l’apposito tag `@Rule`; sarà poi il sistema a gestire automaticamente l’esecuzione dell’Activity scelta e dei casi di test.
- **Utilizzo della classe ActivityScenario con il metodo setUp().** In questo caso viene dichiarato un oggetto di tipo `ActivityScenario`, il quale definisce uno scenario di esecuzione senza avviare l’Activity specificata. Questa operazione viene effettuata in combinazione con il metodo `setUp()`, etichettato con il tag `@Before`. A tale scopo è stato definito il seguente algoritmo:
 1. Viene istanziato un `CountDownLatch` posto in attesa.
 2. Il sistema effettua un insieme di operazioni preliminari necessarie per eseguire correttamente l’Activity specificata, e successivamente rilascia il componente latch.
 3. Viene dunque eseguita l’Activity associata all’`ActivityScenario`.

Il metodo `setUp()` si occupa anche della navigazione alla schermata oggetto di test. A tale scopo sono state utilizzate due metodologie: nelle schermate di autenticazione sono state usate le classi `NavHostFragment` e `NavController`, mentre per i fragment della `MainActivity` è stata implementata l’interfaccia `ViewAction` allo scopo di performare un click sull’elemento del componente `BottomNavigationView` al quale è associata la schermata da testare.

La libreria Espresso mette a disposizione alcuni metodi che permettono di testare il funzionamento dei singoli elementi che costituiscono la UI; seguono i più utilizzati per testare Communimib:

- `onView()`: per identificare un componente della UI attraverso un Matcher, come l’id (metodo `withId()`) o del testo (metodo `withText()`).
- `perform()`: per effettuare delle azioni sul componente attraverso un `ViewAction`, come il click (metodo `click()`).
- `check()`: per verificare che un componente assuma determinati comportamenti, attraverso l’uso di un `ViewAssertion` come il metodo `matches()`.

Nel progetto di testing sono stati inoltre utilizzati anche altri metodi offerti dalla libreria per effettuare dei test più specifici.

Capitolo 4

Utilizzo di ChatGPT

4.1 Scopo dell'analisi sperimentale

Nei capitoli precedenti è stato ampiamente illustrato l'intero processo che ha portato alla realizzazione di Communimib, dalla delineazione degli obiettivi che l'applicazione avrebbe dovuto raggiungere fino alla descrizione dettagliata della sua implementazione. Sebbene l'analisi, l'implementazione e il testing dell'applicazione siano stati gli aspetti che hanno richiesto il maggior impegno dal punto di vista pratico, il lavoro condotto per la stesura di questo elaborato non si limita alla sola creazione di uno strumento vantaggioso per l'ateneo, ma ha come obiettivo principale la valutazione dell'effettivo contributo dei Large Language Models – nel caso specifico ChatGPT – durante tutte le fasi dello sviluppo software.

Per fornire un minimo di contesto, i Large Language Models sono modelli di intelligenza artificiale progettati per comprendere, elaborare e generare linguaggio naturale. Questi modelli analizzano e processano il testo per inferire le relazioni statistiche tra le parole, facendo poi uso di complessi modelli probabilistici per generare testi articolati e fornire risposte coerenti e puntuali. Questa abilità viene acquisita attraverso un'intensa fase di addestramento che coinvolge l'analisi di grandi dataset testuali, durante la quale il modello impara le strutture linguistiche e semantiche necessarie per affrontare compiti che coinvolgono la manipolazione del linguaggio naturale.

L'avvento di ChatGPT ha veicolato l'attenzione mediatica verso questa famiglia di strumenti, i quali possono essere potenzialmente utilizzati per compiere una vasta gamma di operazioni: alcuni esempi possono essere la scrittura e la rielaborazione di testi, la traduzione di contenuti in lingua straniera, la ricerca di informazioni, la generazione di codice sorgente, ecc.

Soprattutto in ambito informatico, l'impiego di strumenti come i Large Language Models può rappresentare una vera e propria rivoluzione nelle modalità di lavoro: l'obiettivo dell'analisi condotta durante lo sviluppo di Communimib è proprio comprendere come l'utilizzo di ChatGPT, massimo esponente (per fama e potenzialità) dei Large Language Models, può impattare durante tutto l'arco di produzione del software, partendo dall'ideazione fino ad arrivare al rilascio.

4.2 Modalità di esecuzione e risultati ottenuti

Fissati gli obiettivi che l'analisi sperimentale vuole raggiungere, è possibile chiarificare le modalità pratiche che hanno consentito il perseguitamento di tali obiettivi. Contestualmente all'uso di ChatGPT, nel corso delle diverse fasi di progettazione e sviluppo di Communimib, è stato redatto un registro atto a monitorarne l'utilizzo: la compilazione del suddetto registro è stata effettuata non soltanto al fine di misurare la frequenza con cui lo strumento è stato utilizzato, ma anche per valutarne (caso per caso) l'effettiva utilità.

Le sottosezioni riportate successivamente hanno lo scopo di suddividere l'intera analisi sulla base del contesto in cui i diversi utilizzi hanno avuto luogo: a diverse fasi del processo di sviluppo possono infatti corrispondere diverse modalità di utilizzo di ChatGPT.

4.2.1 Analisi dei requisiti

La fase di analisi dei requisiti è quella che contraddistingue l'avvio del processo di realizzazione di qualsiasi sistema software; rappresenta un passo estremamente delicato per la buona riuscita del lavoro, poiché qualora venisse svolta in maniera sommaria e svogliata, la conseguenza concreta può essere il fallimento dell'intero progetto.

Sebbene ChatGPT sia uno strumento estremamente potente e versatile, durante l'analisi dei requisiti esso non ha trovato modo di rivelarsi utile; i motivi che si celano dietro questo fenomeno sono fortemente legati alle caratteristiche proprie di questa tappa del processo di sviluppo.

Le difficoltà riscontrabili in questa fase non sono date soltanto dalla sua grande importanza per le fasi successive, ma anche dalle competenze che essa richiede per essere messa in atto correttamente: al fine di condurre una buona analisi dei requisiti è necessario disporre di buone doti di ragionamento, capacità di pensiero critico ed una buona comprensione delle dinamiche che si verificano nel dominio di interesse.

Lo studio del dominio, l'individuazione degli attori coinvolti e la formulazione dei requisiti da soddisfare sono operazioni dal carattere fortemente "umano", in cui il contributo di un LLM può difficilmente rivelarsi determinante: ogni situazione va analizzata sulla base di condizioni specifiche, che strumenti come ChatGPT spesso non hanno la sensibilità di cogliere.

Il registro degli utilizzi conferma pienamente quanto constatato fino a questo momento: durante l'intero arco dell'analisi dei requisiti non è stato fatto alcun uso di ChatGPT. Un lettore esperto e critico potrebbe trovare bizzarra la totale mancanza di utilità dell'assistente virtuale in questa fase; essendo ChatGPT un eccellente elaboratore di testi, avrebbe forse potuto essere d'aiuto nel processo di stesura e revisione dei casi d'uso. Questo non è accaduto perché i casi d'uso espressi in formato dettagliato hanno una struttura specifica ed un proprio stile di scrittura, che difficilmente lascia spazio a grandi possibilità di rimodellazione.

In ultima istanza è possibile inoltre affermare che la stesura dei casi d'uso, il principale elaborato prodotto nella fase di analisi dei requisiti, è un'operazio-

ne che predilige nettamente la cura del contenuto a discapito della forma; dal momento che i fattori oggetto dell'analisi non possono essere noti a ChatGPT perché dipendenti dal contesto specifico, è chiaro che lo strumento non risulti adatto ad essere utilizzato in questo ambito.

4.2.2 Implementazione

Il processo di implementazione è sicuramente quello che ha richiesto il maggior impiego di tempo e risorse, fornendo allo stesso tempo molte occasioni per sperimentare l'utilità di ChatGPT. Questa fase non è data dalla sola scrittura del codice sorgente dell'applicazione, ma è bensì costituita da un susseguirsi continuo di progettazioni, idee, esperimenti, tentativi e correzioni: è proprio in questo contesto che ChatGPT ha potuto dimostrare a pieno le sue potenzialità.

Chiaramente, la generazione del codice sorgente rappresenta la principale funzionalità sfruttata durante lo sviluppo.

Molto spesso lo sviluppatore si trova a delineare dal punto di vista teorico il funzionamento di un meccanismo da implementare, senza però conoscere quali classi specifiche ha la necessità di utilizzare per metterlo in atto: in questo contesto ChatGPT si rivela spesso molto utile, fornendo codice immediatamente funzionante oppure informando chi lo utilizza dell'esistenza di classi o librerie adatte ai propri scopi.

A supporto di questa tesi è possibile prendere in esame l'implementazione del meccanismo di polling per il controllo dello stato di verifica dell'indirizzo e-mail: la modalità con cui strutturare l'implementazione è definita in partenza, pertanto nel prompt di ChatGPT (visibile nella Chat 4.1) è stata posta una domanda volta a scoprire come mettere in atto la strategia delineata.

Chat 4.1: Thread polling per la verifica dell'e-mail

Utente:

Sto scrivendo un'app android in Java.

Come faccio, all'interno di un metodo di una classe, a creare un thread che esegua periodicamente del codice (a intervalli di 5 secondi) e che sia anche interrompibile manualmente all'occorrenza?

ChatGPT:

Per creare un thread che esegua periodicamente del codice a intervalli di 5 secondi e che sia interrompibile manualmente, puoi utilizzare un'implementazione con `ScheduledExecutorService`.

Questa classe fornisce un'interfaccia per eseguire task in modo pianificato, consentendo anche l'interruzione del task.

Ecco un esempio di come potresti farlo all'interno del metodo di una classe in un'app Android in Java:

```
import java.util.concurrent.Executors;  
import java.util.concurrent.ScheduledExecutorService;
```

```

import java.util.concurrent.TimeUnit;

public class MyClass {
    private ScheduledExecutorService executorService;

    public void startPeriodicTask() {
        executorService = Executors.
            newSingleThreadScheduledExecutor();
        executorService.scheduleAtFixedRate(() -> {
            // Codice da eseguire periodicamente
            // Ad esempio, puoi inserire qui la
            // logica che desideri eseguire ogni 5
            // secondi
            // Assicurati di non bloccare il thread
            // principale con operazioni lunghe
            // Se devi aggiornare l'interfaccia
            // utente, utilizza Handler.post o
            // runOnUiThread
        }, 0, 5, TimeUnit.SECONDS);
    }

    public void stopPeriodicTask() {
        if (executorService != null &&
            executorService.isShutdown()) {
            executorService.shutdownNow();
        }
    }
}

```

In questo esempio, il metodo `startPeriodicTask()` avvia un `ScheduledExecutorService` che eseguirà il codice fornito nel lambda a intervalli di 5 secondi (`scheduleAtFixedRate`).

Questo metodo restituisce un `ScheduledFuture`, ma poiché non è necessario riferirsi ad esso per interrompere l'esecuzione, non viene memorizzato.

Il metodo `stopPeriodicTask()` interrompe il task in esecuzione chiamando `shutdownNow()` sull'`ExecutorService`.

Questo metodo tenta di interrompere immediatamente l'esecuzione del task in corso e restituisce una lista dei task pendenti non eseguiti. Se necessario, puoi gestire tali task in sospeso in base alle tue esigenze.

Ricorda che il codice all'interno del lambda di `scheduleAtFixedRate()` verrà eseguito su un thread diverso dal thread principale dell'applicazione, quindi assicurati di non bloccare questo thread con operazioni lunghe. Se è necessario aggiornare l'interfaccia utente dall'interno del lambda, utilizza `Handler.post()` o `runOnUiThread()` per farlo in modo sicuro.

La programmazione di un sistema software può portare con sé difficoltàlegate non soltanto alla scrittura del codice, ma anche alla sua organizzazione. La differenza tra un programmatore alle prime armi ed un programmatore con esperienza può emergere anche nelle scelte compiute per organizzare il proprio codice: gli sviluppatori con maggiore esperienza tendono a non sottovalutare fattori quali leggibilità e facilità di manutenzione, prediligendo così una chiara nomenclatura delle variabili, un corretto uso dell'indentazione ed un frequentericorso al refactoring.

Nonostante sia un compito banale nella maggior parte dei casi, denominare correttamente una variabile può talvolta rivelarsi tanto importante quanto complesso; il nome scelto per una variabile deve scongiurare qualsiasi tipo di ambiguità, richiamandone immediatamente il contenuto. Nel corso dello sviluppo di Communimib, ChatGPT è stato sporadicamente utilizzato anche per ottenere suggerimenti in merito alla denominazione delle variabili, così come si può constatare osservando la Chat 4.2.

Chat 4.2: Scelta del nome per uno dei tanti MutableLiveData

Utente:

Devo scegliere il nome di alcune variabili per il mio progetto Android. Inizialmente ho definito un MutableLiveData chiamato emailVerificationResult che serve a sapere se l'utente ha già confermato o no la sua mail.

Poi ho dovuto definire un ulteriore MutableLiveData, questa volta mi serve a sapere se l'invio della mail di conferma è andato o meno a buon fine: che nome mi consigli di mettere?

ChatGPT:

Potresti chiamarlo `emailConfirmationStatus`, `emailConfirmationResult`, o semplicemente `emailConfirmation`. Scegliere un nome chiaro e descrittivo aiuta a rendere il codice più leggibile e comprensibile per te e per gli altri sviluppatori che potrebbero lavorare sul progetto in futuro.

Utente:

Ma `emailConfirmationResult` fa intendere il primo caso, non il secondo: poni enfasi sul fatto che è una variabile che rappresenta l'esito dell'INVIO della mail.

ChatGPT:

Hai ragione, hai fatto una distinzione importante. In tal caso, potresti chiamarlo `emailConfirmationSendingResult`, `emailSendingStatus`, o `emailSentSuccess`.

Questi nomi mettono in evidenza che si tratta dell'esito dell'invio della mail di conferma.

Utente:

E se invece ho una variabile che ha la stessa funzione, ma riguarda la mail di password reset? come la chiamo?

ChatGPT:

Per la variabile che tiene traccia dell'esito dell'invio della mail di reset della password, potresti utilizzare nomi simili a quelli suggeriti per l'invio della mail di conferma. Ad esempio: `passwordResetSendingResult`, `passwordResetSendingStatus`, `passwordResetSentSuccess`.

Questi nomi mantengono la coerenza e la chiarezza nella nomenclatura, evidenziando che si tratta dell'esito dell'invio della mail di reset della password

Oltre ad essere una buona fonte di consigli, ChatGPT si è rivelato anche in grado di portare a termine compiti prettamente pratici come, ad esempio, l'applicazione di semplici step di refactoring volti ad aumentare il livello di riusabilità del codice. Allo stesso tempo, è possibile utilizzare tale strumento anche per verificare che l'applicazione di una data operazione di refactoring sia sensata in merito alle scelte architettoniche effettuate durante le prime fasi di progettazione del sistema.

All'interno del registro contenente tutti gli utilizzi di ChatGPT sono presenti diversi esempi di questa tipologia: la Chat 4.3 riporta quello più chiaro ed emblematico, in cui viene richiesto di effettuare l'estrazione di una classe `ViewHolder` per separarla dal relativo `Adapter` (si noti che la chat è stata riportata solo parzialmente per questioni di spazio, dal momento che la maggior parte dell'output generato è costituito da codice Java).

Chat 4.3: Estrazione di una classe ViewHolder

Utente:

...

Osservando questo codice, come faccio ad estrarre il `ViewHolder` in una classe a parte per poterlo riutilizzare in una diversa `RecyclerView`?

ChatGPT:

Per estrarre il `ViewHolder` in una classe separata, è necessario creare una nuova classe che estende `RecyclerView.ViewHolder`. Di seguito sono riportati i passaggi dettagliati per farlo:

1. Creare una nuova classe per il ViewHolder:
 - Crea una nuova classe chiamata `PostViewHolder` in un file separato.
 2. Trasferire il codice del ViewHolder:
 - Sposta il codice del `ViewHolder` dalla classe `DashboardRecyclerAdapter` alla nuova classe `PostViewHolder`.
 - Aggiungi un costruttore che accetta un `View` e inizializza i vari `View` elementi.
 3. Adattare il ViewHolder per essere riutilizzabile:
 - Assicurati che la nuova classe `PostViewHolder` accetti un `OnItemClickListener` nel costruttore, se necessario, per gestire i clic sugli elementi.
 4. Modificare l'adapter esistente per utilizzare il nuovo ViewHolder:
 - Modifica la classe `DashboardRecyclerAdapter` per utilizzare il nuovo `PostViewHolder`.
- ...

Utente:

...

Dammi nuovamente la soluzione considerando questa versione del codice; voglio inoltre estrarre anche `OnPostClickListener` in un file separato

ChatGPT:

...

`OnPostClickListener.java:`

```
package it.unimib.communimib.ui.main.dashboard;
import it.unimib.communimib.model.Post;

public interface OnPostClickListener {
    void onItemClick(Post post);
    void onImageSliderClick(Post post);
}
```

`PostViewHolder.java:`

...

Un’ulteriore modalità di utilizzo di ChatGPT in fase di implementazione è quella relativa all’individuazione ed alla conseguente correzione di bug e malfunzionamenti nel codice. In ambito programmatico possono manifestarsi diverse tipologie di errori, alcune più semplici da debellare rispetto ad altre: alcuni esempi possono essere gli errori di compilazione, runtime, logici, ecc.

In questo contesto, i Large Language Models possono essere impiegati per analizzare il codice fornito in input e segnalare la presenza di errori al suo interno; nonostante la loro efficacia sia alquanto altalenante, talvolta questi strumenti possono rivelarsi in grado di individuare le cause dei malfunzionamenti e spiegare al programmatore le motivazioni che si celano dietro al bizzarro comportamento assunto dall’applicazione.

Durante la programmazione dell’app Communimib è stato dedicato molto tempo alla risoluzione degli errori emersi in fase di sviluppo, trasformando così il progetto in un vero e proprio banco di prova per valutare le prestazioni di ChatGPT in ambito di ricerca e correzione dei bug. Come emerge anche da registro, si sono verificate diverse interazioni con l’assistente virtuale durante le sessioni di bug-fixing; a titolo esemplificativo viene riportata la Chat 4.4 (riportata parzialmente per questioni di spazio) che ha avuto un ruolo determinante nel posizionamento di alcuni elementi grafici (`RecyclerView` e `SearchView`) all’interno di una delle tante schermate.

Chat 4.4: Posizionamento corretto della `RecyclerView`

Utente:

...

In questo layout la `RecyclerView` si sovrappone alla `SearchView`, come posso risolvere?

ChatGPT:

Per risolvere il problema della sovrapposizione della `RecyclerView` sulla `SearchView` nel layout XML fornito, puoi modificare i vincoli della `RecyclerView` all’interno del `ConstraintLayout`.

Attualmente, hai posizionato la `RecyclerView` sotto la `SearchView` utilizzando `app="@+id/searchView2"`. Tuttavia, questo può causare sovrapposizioni poiché entrambi gli elementi occupano lo stesso spazio sull’asse Y.

Per risolvere questo problema, puoi posizionare la `RecyclerView` sopra la `SearchView` o utilizzare vincoli che consentano loro di essere disposti in modo appropriato.

Ecco un esempio di come puoi modificare i vincoli per posizionare correttamente la `RecyclerView` sopra la `SearchView`:

...

Utente:

Io ho bisogno che la **SearchView** rimanga sopra alla **RecyclerView**

ChatGPT:

Capisco, vuoi che la **SearchView** rimanga sopra la **RecyclerView**. Puoi ottenere questo risultato modificando i vincoli della **SearchView** e della **RecyclerView** in modo che la **RecyclerView** sia posizionata sotto la **SearchView**.

Ecco come puoi modificare il layout XML per ottenere questo effetto:

```
<androidx.recyclerview.widget.RecyclerView  
    android:id="@+id/fragmentReport_recyclerView"  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    app:layout_constraintTop_toBottomOf="@+id/  
        searchView2"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintEnd_toEndOf="parent" />  
  
<SearchView  
    android:id="@+id/searchView2"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_marginStart="8dp"  
    android:layout_marginTop="8dp"  
    android:layout_marginEnd="8dp"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

Con questi vincoli, la **SearchView** è posizionata all'inizio dell'asse Y (in cima) del **ConstraintLayout**, mentre la **RecyclerView** è posizionata sotto la **SearchView**, occupando tutto lo spazio rimanente fino al bordo inferiore del **ConstraintLayout**.

Ciò garantirà che la **SearchView** rimanga sopra la **RecyclerView**, evitando sovrapposizioni indesiderate.

Ricapitolando, in fase di implementazione è stato fatto uso di ChatGPT a supporto delle seguenti mansioni:

- Scrittura di codice sorgente
- Organizzazione del codice e refactoring
- Individuazione e risoluzione di malfunzionamenti

4.2.3 Testing

In ambito di sviluppo, specialmente durante l'applicazione di un processo iterativo, la fase di testing è un passo cruciale per il corretto avanzamento del progetto fino al raggiungimento del rilascio. Gli sforzi messi in atto per effettuare una buona scrittura dei test vengono sempre ripagati in un secondo momento, poiché la loro esecuzione consente di validare il funzionamento del sistema software in maniera estremamente rapida ed automatizzabile; a tal proposito è stato fatto uso della piattaforma GitHub Actions per mettere in atto meccanismi di Continuous Integration basati sull'esecuzione automatica dei casi di test.

ChatGPT è stato utilizzato in questo scenario sia per configurare correttamente il sistema di Continuous Integration, che per l'effettiva scrittura delle singole classi di test: le Chat 4.5 e 4.6 costituiscono due dei numerosi esempi di interazioni avvenute in fase di testing.

Chat 4.5: Configurazione GitHub Actions

Utente:

...

Modifica questo codice per fare sì che le successive Github Actions eseguano automaticamente i test di unità

ChatGPT:

Per eseguire i test di unità con Gradle, puoi aggiungere un altro passo dopo il passo di compilazione per eseguire i test.

Ecco come puoi modificare il tuo flusso di lavoro:

```
name: Android CI

on:
  push:
    branches:
      - main
      - development
  pull_request:
    branches:
      - main
      - development

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
```

```

    - uses: actions/checkout@v4

    - name: Set up JDK 17
      uses: actions/setup-java@v4
      with:
        java-version: '17'
        distribution: 'temurin'
        cache: gradle

    - name: Grant execute permission for gradlew
      run: chmod +x gradlew

    - name: Build with Gradle
      run: ./gradlew build

    - name: Run unit tests
      run: ./gradlew test

```

Con questa modifica, dopo la compilazione del progetto con Gradle, verranno eseguiti automaticamente i test di unità.

Chat 4.6: Implementazione del metodo `getUserByEmail`

Utente:

```

import java.util.Map;

public class FakeUserRemoteDataSource implements
    IUserRemoteDataSource {
    private Map<String, User> users;

    @Override
    public void storeUserParameters(String uid,
        String email, String name, String surname,
        Callback callback) {
        if (users.containsKey(uid)) {
            users.replace(uid, new User(email, name,
                surname));
        } else {
            users.put(uid, new User(email, name,
                surname));
        }
        callback.onComplete(new Result.Success());
    }

    @Override

```

```

public void getUserByEmail(String email, Callback
    callback) {
    // Implementazione necessaria per ottenere l'
        utente per email
}

@Override
public void updateNameAndSurname(String uid,
    String name, String surname) {
    if (users.containsKey(uid)) {
        User user = users.get(uid);
        user.setName(name);
        user.setSurname(surname);
    } else {
        // Gestione nel caso l'utente non sia
        // presente
        // Potresti lanciare un'eccezione o
        // gestire diversamente a seconda delle
        // necessita
        throw new IllegalArgumentException("User
            with uid " + uid + " not found");
    }
}

```

implementa il metodo `getUserByEmail`

ChatGPT:

Per implementare il metodo `getUserByEmail` nella classe `FakeUserRemoteDataSource`, dovresti iterare attraverso la mappa degli utenti e cercare un utente con l'email specificata. Ecco un esempio di come potresti farlo:

```

@Override
public void getUserByEmail(String email, Callback
    callback) {
    User foundUser = null;
    for (User user : users.values()) {
        if (user.getEmail().equals(email)) {
            foundUser = user;
            break;
        }
    }

    if (foundUser != null) {
        callback.onComplete(new Result.Success(
            foundUser));
    } else {

```

```
        callback.onComplete(new Result.Error("User  
not found with email: " + email));  
    }  
}
```

In questo metodo, scorriamo attraverso tutti gli utenti nella mappa `users` e confrontiamo l'email di ciascun utente con l'email fornita come parametro. Se troviamo un utente con l'email corrispondente, chiamiamo il metodo `onComplete` del callback con un oggetto `Result.Success` contenente l'utente trovato. Se non troviamo alcun utente con l'email specificata, chiamiamo `onComplete` con un oggetto `Result.Error` contenente un messaggio appropriato.

Le modalità di utilizzo di ChatGPT in fase di testing si collocano dunque sulla stessa linea di quelle messe in atto durante la fase di implementazione: anche in questo caso lo strumento è stato utilizzato sia per generare codice sorgente, che per correggere gli errori presenti nel codice già esistente.

4.2.4 Sintesi dei dati raccolti

In seguito all'illustrazione dei vari impieghi di ChatGPT durante il progetto, l'analisi si conclude con l'esposizione dei dati riscontrati durante l'esperimento.

Uno dei dati più rilevanti ai fini di un'osservazione critica dei risultati prodotti è quello relativo alla distribuzione degli utilizzi tra le diverse fasi di sviluppo dell'applicazione; il grafico osservabile in Figura 4.7 dimostra ampiamente i fenomeni precedentemente descritti, evidenziando come la maggior parte delle interazioni siano avvenute in fase di implementazione. Dal momento che durante l'analisi dei requisiti non è stato fatto uso dell'assistente virtuale, tale fase non compare nel grafico.

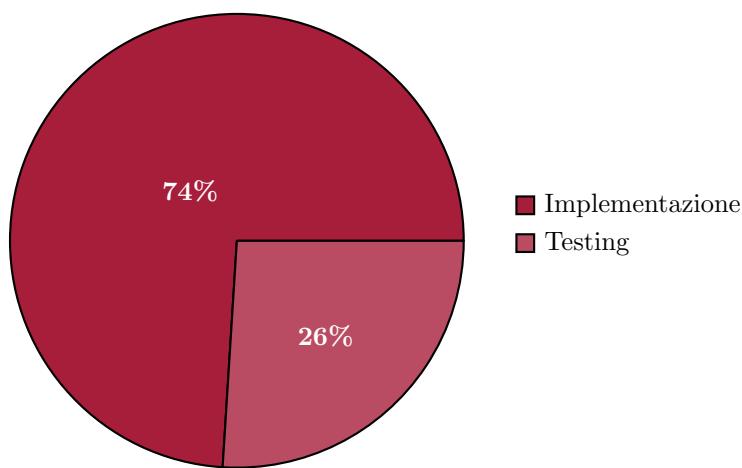


Figura 4.7: Distribuzione delle interazioni con ChatGPT nelle diverse fasi di sviluppo

Oltre alla disparità degli impieghi di ChatGPT nelle varie fasi di sviluppo, è infine importante comprendere l'effettiva utilità delle interazioni avvenute durante la conduzione della sperimentazione.

I grafici visibili in Figura 4.8 e Figura 4.9 consentono di riassumere facilmente il livello di utilità manifestato da ChatGPT per ciascuna mansione in cui è stato utilizzato, rispettivamente per le fasi di implementazione e testing; all'interno dei grafici, le mansioni sono indicate come segue:

- **SC** - Scrittura codice
- **OR** - Organizzazione e refactoring
- **BF** - bug-fixing

Con il supporto dei grafici è infine possibile affermare che, durante l'intero processo di realizzazione di Communimib, ChatGPT si è rivelato uno strumento utile, versatile e capace di ritagliarsi un ruolo importante nel corso delle quotidiane attività compiute in ambito di sviluppo.

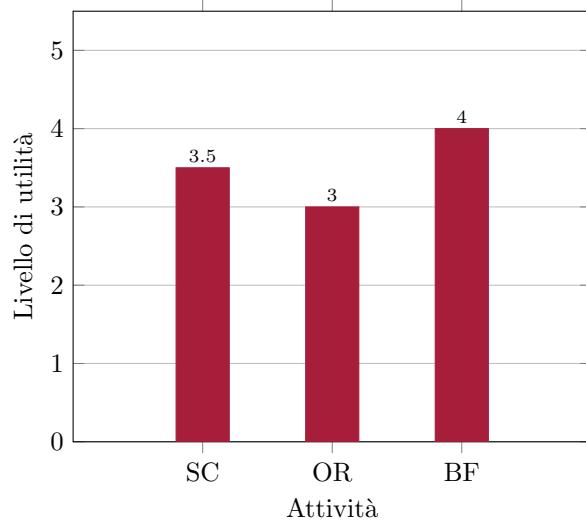


Figura 4.8: Utilità di ChatGPT durante la fase di implementazione

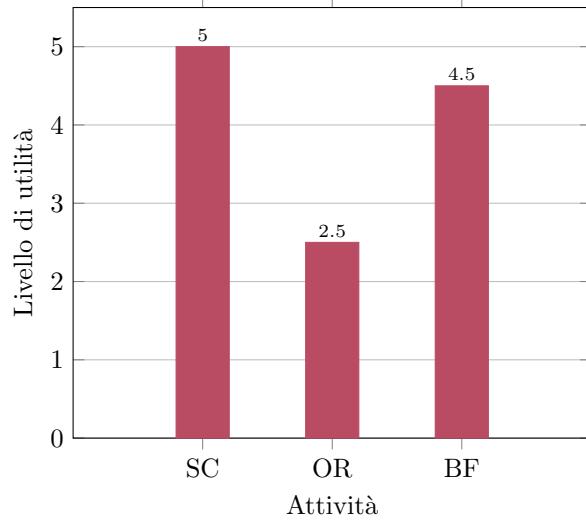


Figura 4.9: Utilità di ChatGPT durante la fase di testing

Conclusioni

Lo studio oggetto del presente elaborato ha avuto come principale obiettivo la valutazione dell'utilità di ChatGPT nel contesto dello sviluppo di un'applicazione mobile. È stato possibile perseguire tale obiettivo attraverso la conduzione di un'analisi sperimentale che ha permesso, mediante lo sviluppo di Communimib, di testare concretamente il ruolo che gli LLM (ed in particolare ChatGPT) possono assumere nel processo di realizzazione di un sistema software articolato.

Così come testimoniato dai dati emersi, nonostante non sia stato equalmente utilizzato in tutti e tre i principali ambiti che caratterizzano il processo di sviluppo, nel corso del progetto ChatGPT si è rivelato una risorsa preziosa, spesso in grado di agevolare lo svolgimento delle principali attività.

Il giudizio relativo alla sua utilità non può che essere positivo, sebbene lo strumento presenti talvolta efficacia altalenante a seconda delle richieste che vengono formulate e del contesto in cui lo si adopera.

La conduzione di un progetto di sviluppo è una mansione tipicamente svolta in gruppo; la realizzazione di Communimib non fa eccezione, dal momento che ha richiesto impegno e dedizione da parte di un piccolo gruppo composto da tre persone. In questo scenario, ChatGPT è stato occasionalmente utilizzato anche come metro di giudizio durante i confronti interni al team di sviluppo: non sono mancate occasioni in cui i membri del team si sono trovati in disaccordo sulle scelte programmatiche da mettere in atto a livello di codice, e in tale contesto sono state poste a ChatGPT domande volte a comprendere la sua opinione sull'argomento al fine di trovare un punto di incontro.

Questa tipologia di utilizzo rappresenta una delle molte possibilità offerte dall'impiego di ChatGPT a supporto dello sviluppo software, non solo come mero strumento in grado di compiere le operazioni più semplici e ripetitive, ma anche come importante alleato capace talvolta di fornire spunti interessanti in situazioni controverse.

In ultima istanza, l'utilità di ChatGPT in ambito di sviluppo risulta comprovata. La nascita di questo strumento ha però destato diverse preoccupazioni all'interno della comunità dei programmatore, poiché molti hanno iniziato a cavalcare l'onda delle opinioni apocalittiche che delineano la figura dello sviluppatore come semplice "operaio del codice", presto sostituito totalmente da modelli di intelligenza artificiale più rapidi e preparati di lui.

Esprimendomi ora non soltanto in qualità di autore di questo elaborato ma anche in qualità di informatico, conservo la speranza che il mio contributo non si limiti solamente a testimoniare la validità di ChatGPT come strumento di

supporto in materia di sviluppo, ma possa anche servire a dimostrare come la figura dello sviluppatore non vada banalizzata e ridotta al solo ruolo di scrittore di codice, bensì vada osservata nella sua interezza in quanto dotata di creatività e spirito critico. A mio modesto parere, tali caratteristiche sono quelle che permetteranno allo sviluppatore, in chiave futura, di lavorare al fianco di strumenti come gli LLM, senza che essi possano sostituirlo.

Ringraziamenti

In questa breve sezione vorrei spendere qualche riga per ringraziare tutti coloro che hanno reso possibile la stesura del presente elaborato; senza il sostegno di ciascuna di queste persone, probabilmente nessuno avrebbe mai potuto leggere queste pagine.

In primo luogo desidero ringraziare la Professoressa Daniela Micucci e la Dottoressa Maria Teresa Rossi, per avermi concesso l'opportunità di realizzare questo lavoro e per avermi guidato in maniera sempre presente e puntuale fino al suo completamento.

Vorrei inoltre ringraziare i miei colleghi Marco Ferioli e Giulia Vitale, che mi hanno accompagnato non soltanto nella realizzazione di Communimib e nella conduzione dell'analisi sperimentale oggetto di questa tesi, ma anche in tutto l'arco del mio percorso universitario.

Un ringraziamento speciale va alla mia mamma e al mio papà, sempre estremamente presenti e pronti a darmi il loro supporto in ogni situazione.

Li ringrazio per avermi guidato con amore, permettendomi di fare le mie scelte senza condizionamenti e sostenendomi indipendentemente dai risultati ottenuti. Se oggi posso dire di aver raggiunto un grande traguardo, gran parte del merito è da attribuire a loro: nonostante fosse un compito impegnativo, sono stati capaci di spronarmi senza mai mettermi sotto pressione.

Se è vero che il "lavoro" del genitore è il lavoro più difficile al mondo, qui siamo davanti a due veri professionisti: le poche righe che mi è concesso dedicargli non potranno mai in alcun modo esprimere l'immenso amore e gratitudine che provo ogni giorno nei loro confronti.

Vorrei inoltre ringraziare la mia nonna e il mio nonno, i quali hanno rappresentato due figure fondamentali per la mia crescita e la mia maturazione: affettuosi, pazienti e saggi, hanno saputo trasmettermi valori che mi porterò dietro per tutta la vita.

La loro presenza è stata immensamente importante e per questo li ringrazio calorosamente entrambi, con la speranza che possano condividere insieme a me tante altre di queste conquiste.

Un sentito ringraziamento lo voglio poi dedicare ai miei zii, che non hanno mai fatto mancare il loro grande sostegno; spero che in futuro possano continuare ad accompagnarmi lungo le tante altre sfide che mi attendono, proprio come hanno sempre fatto con quelle accademiche.

Infine, vorrei ringraziare Sara.

Il mio mentore, la mia spalla, il mio presente e (forse) il mio futuro: senza il suo sostegno e senza la sua capacità di interpretarmi e condurmi sulla strada corretta, sicuramente non starei scrivendo queste pagine.

Averla accanto a me come esempio da seguire mi ha aiutato tantissimo, non soltanto in ambito accademico; la sua tenacia, la sua dedizione e il suo modo di essere contemporaneamente sia forte che fragile sono solo alcune delle caratteristiche che la rendono insostituibile.

Indipendentemente da cosa possa riservare il futuro, la sua presenza è stata fondamentale per raggiungere questo importante traguardo; per questo la ringrazio infinitamente e le auguro di riuscire a realizzare tutti i sogni che ha nel cassetto.

Voglio chiudere il presente elaborato ricordando Rio, il gigante buono che insieme ai nonni ha colorato le mie giornate vedendomi diventare adulto. Il conseguimento della laurea non poteva che arrivare ad un anno esatto dalla sua partenza: non ho dubbi che, almeno in parte, dietro questa coincidenza ci sia il suo zampino, o per meglio dire, il suo zampone.

Sitografia

- [1] Deniz Coskun. *ImageSlideShow — Android Image Slider*. 2023. URL: <https://github.com/denzcoskun/ImageSlideshow> (visitato il giorno 20/05/2024).
- [2] Google for Developers. *Espresso - Android UI testing framework*. 2024. URL: <https://developer.android.com/training/testing/espresso> (visitato il giorno 12/04/2024).
- [3] Google for Developers. *Firebase*. n.d. URL: <https://firebase.google.com/> (visitato il giorno 23/05/2024).
- [4] Google for Developers. *Guide to app architecture*. 2023. URL: <https://developer.android.com/topic/architecture> (visitato il giorno 23/06/2024).
- [5] JUnit. *JUnit 4 - simple framework to write repeatable tests*. 2021. URL: <https://junit.org/junit4/> (visitato il giorno 08/07/2024).
- [6] Mockito. *Mockito - most popular mocking framework for Java*. 2024. URL: <https://site.mockito.org/> (visitato il giorno 09/04/2024).
- [7] João Guilherme Berti Sczip. *Clean Architecture — A Little Introduction*. 2020. URL: <https://medium.com/swlh/clean-architecture-a-little-introduction-be3eac94c5d1> (visitato il giorno 23/06/2024).
- [8] Wikipedia. *Metodo MoSCoW*. 2023. URL: https://it.wikipedia.org/wiki/Metodo_MoSCoW (visitato il giorno 25/06/2024).
- [9] Yalantis. *uCrop - Image Cropping Library for Android*. 2024. URL: <https://github.com/Yalantis/uCrop> (visitato il giorno 15/05/2024).