



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:
Giulia Raffaella Vitale
Matricola 885938

Anno Accademico 2023-2024

Indice

Introduzione	2
1 Analisi dei requisiti	3
1.1 Come nasce Communimib	3
1.2 Analisi dei casi d'uso	5
1.2.1 Diagramma dei casi d'uso	5
1.2.2 Attori	6
1.2.3 Descrizione dei casi d'uso in formato dettagliato	7
1.3 Organizzazione dei requisiti	18
1.4 Modello di dominio	21
2 Descrizione dell'implementazione	22
2.1 Analisi architetturale	22
2.1.1 Clean Architecture	23
2.1.2 Model-View-ViewModel (MVVM)	24
2.1.3 Diagramma dei package	26
2.2 Implementazione	27
2.2.1 Autenticazione	27
2.2.2 Bacheca	33
2.2.3 Segnalazioni	39
2.2.4 Profilo	46
2.2.5 Dettagli relativi all'uso di Firebase	49
3 Progetto di testing	51
3.1 Il ruolo dei test nello sviluppo Android	51
3.2 Test di unità	51
3.3 Test UI	52
4 Utilizzo di ChatGPT	54
4.0.1 Analisi dei requisiti	55
4.0.2 Descrizione dell'implementazione	55
4.0.3 Progetto di testing	57
4.0.4 Considerazioni finali	58
Conclusioni	60
Ringraziamenti	62
Sitografia	63

Introduzione

Il progresso tecnologico nel campo dell'intelligenza artificiale ha determinato la nascita di nuove tecnologie intelligenti capaci di svolgere molteplici attività in una vasta gamma di settori. Tale progresso pone, nella mente delle persone che non conoscono queste tecnologie, un quesito: come e in che misura l'intelligenza artificiale è in grado di supportare l'uomo nelle sue attività quotidiane e nei vari settori in cui questa viene applicata?

A tal proposito è stata scelta una delle tecnologie intelligenti più discusse del momento, ChatGPT, per condurre un esperimento mirato ad investigare l'effettiva efficacia e il supporto fornito durante il processo di sviluppo di un'applicazione mobile.

L'applicazione in questione è stata sviluppata nell'ambito dell'ateneo, con l'obiettivo di rendere più efficace la comunicazione tra coloro che frequentano l'ambiente universitario.

ChatGPT ha assunto un ruolo attivo durante lo sviluppo software, in quanto è stata consultata ogni qualvolta gli sviluppatori hanno avvertito la necessità di informazioni, pareri o supporto.

In tale contesto, il chatbot ha fornito assistenza e appoggio durante le diverse fasi dello sviluppo dell'applicazione, offrendo consigli e suggerendo le modalità corrette per l'implementazione delle funzionalità offerte dall'applicativo. In alcune circostanze, tuttavia, il chatbot ha commesso degli errori che hanno reso necessarie revisioni ed integrazioni.

Dall'esperimento si deduce dunque che ChatGPT è capace di fornire supporto significativo quando richiesto, dimostrando notevole utilità in vari contesti. Tuttavia, nonostante le sue capacità avanzate, ChatGPT non può ancora sostituirsi completamente all'uomo. La sua efficacia dipende dalla collaborazione con l'utente umano, che fornisce il contesto e interpreta i risultati. Questo suggerisce che, almeno per il momento, l'intelligenza artificiale funge da potente strumento complementare piuttosto che da sostituto completo dell'intervento umano.

Capitolo 1

Analisi dei requisiti

1.1 Come nasce Communimib

Communimib, come la maggior parte delle applicazioni mobili 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: molto spesso se ne trovano 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, segue 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 alla definizione dei casi d'uso dell'applicazione. Questo passo è 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.

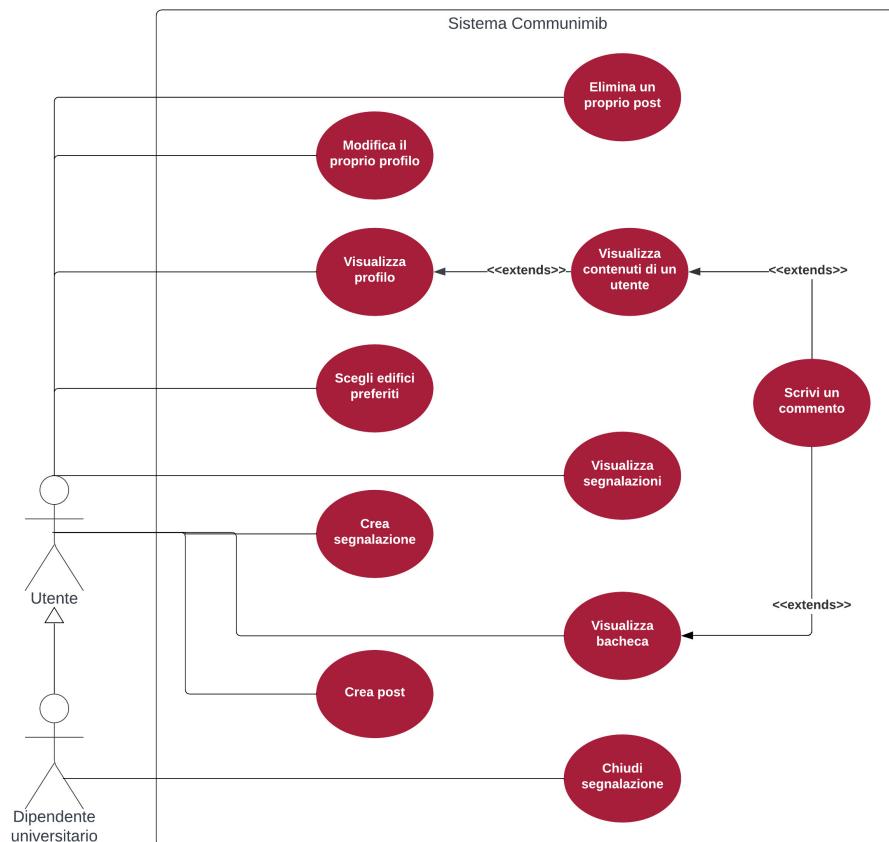


Figura 1.1: Diagramma dei casi d'uso

1.2.2 Attori

Durante le fasi preliminari dell'analisi, l'individuazione degli attori coinvolti nell'utilizzo dell'applicazione è un'attività 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	<ol style="list-style-type: none"> 1. L'utente accede alla sezione dell'applicativo dedicata alle segnalazioni. 2. Il sistema mostra all'utente le segnalazioni richieste.
Estensioni	<p>Scenario alternativo di successo:</p> <ol style="list-style-type: none"> 1. L'utente può applicare uno dei seguenti filtri per visualizzare un sottoinsieme di segnalazioni presenti nel sistema: <ul style="list-style-type: none"> • Edifici preferiti (si veda Tabella 1.3). • Edifici specifici (Es. edificio U1 ed U14). • Tutti gli edifici. 2. Il sistema mostra all'utente le segnalazioni filtrate. <p>Scenario alternativo di fallimento:</p> <ol style="list-style-type: none"> 2.a. Se il sistema fallisce nella visualizzazione delle segnalazioni, viene mostrato all'utente il relativo segnale di errore.
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 sono stati definiti e descritti dettagliatamente i requisiti funzionali, delineando così una visione più chiara del sistema da implementare. Un ulteriore elaborato che risulta propedeutico per un corretto 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 determina la loro definizione, ma 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 di alta priorità 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 a uno specifico dominio; tale modello è stato realizzato per comprendere il contesto in cui deve operare il sistema, rappresentando le entità chiave e 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.

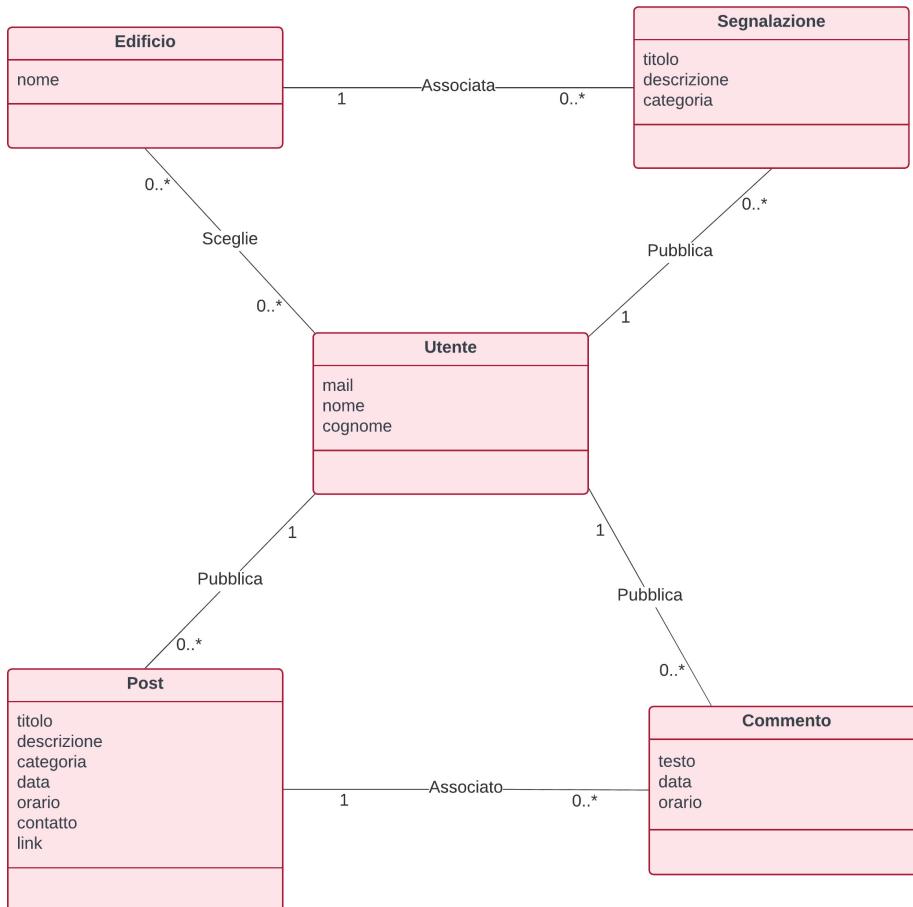


Figura 1.2: Diagramma del modello di dominio

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 processo di creazione di un sistema complesso, in quanto garantiscono:

- **Flessibilità.** Costituisce una caratteristica estremamente importante, poiché rappresenta il grado di semplicità di modifica e manutenzione del codice. Durante l’implementazione è comune identificare errori o problemi che generano malfunzionamenti all’interno del sistema; un’architettura sviluppata in modo coerente consente di gestire e risolvere tali problematiche senza generarne di nuove.
- **Scalabilità.** In generale un sistema non dovrebbe mai essere statico, chiuso e quindi limitato alle funzionalità implementate inizialmente. L’applicazione deve essere estendibile, ovvero deve consentire facilmente l’aggiunta di nuovi contenuti. Questa caratteristica garantisce anche durabilità del software, in quanto potrà essere modificato ed esteso nel tempo secondo le 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.

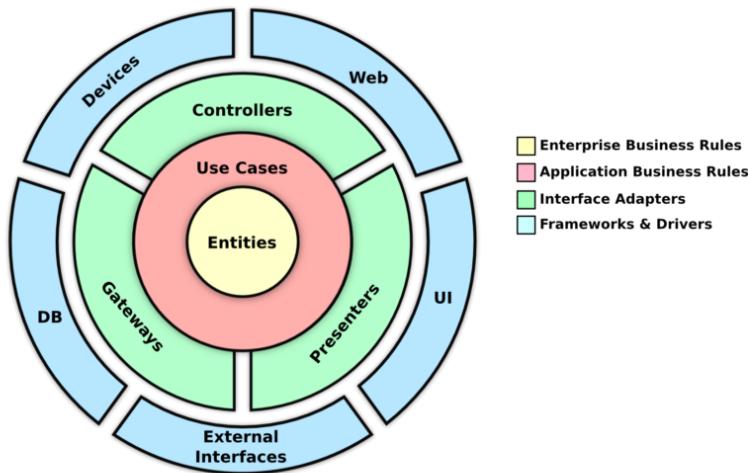


Figura 2.1: Clean Architecture

La regola fondamentale su cui si basa la Clean Architecture è la **Dependency Rule**, secondo la quale la direzione delle dipendenze deve puntare sempre 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 tuttavia ad essere un punto di riferimento per sistemi complessi e le app Android non ricadono in questa categoria; per tale motivo, nello sviluppo di applicazioni Android si segue un insieme di linee guida che si concretizzano nella **Modern App Architecture**, la quale rappresenta una versione più flessibile della Clean Architecture.

I principi su cui si basano le moderne architetture 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 e riutilizzare 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].

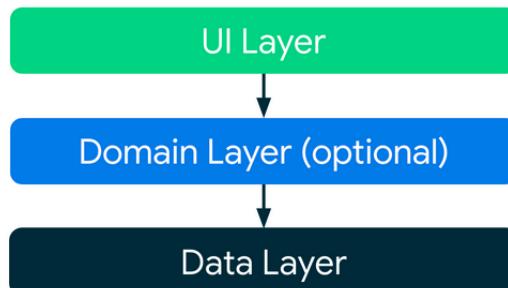


Figura 2.2: Architettura a tre layer

2.1.2 Model-View-ViewModel (MVVM)

Dati i principi precedentemente elencati, il pattern architettonale 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. Le classi Repository fungono da intermediari tra i ViewModel e le classi Datasource, mentre queste ultime rappresentano la fonte dei dati. Sono state implementate due tipologie di Datasource, uno locale ed uno remoto che utilizza i servizi offerti da Firebase per la memorizzazione dei dati.

La suddivisione delle componenti è visualizzabile nel seguente diagramma.

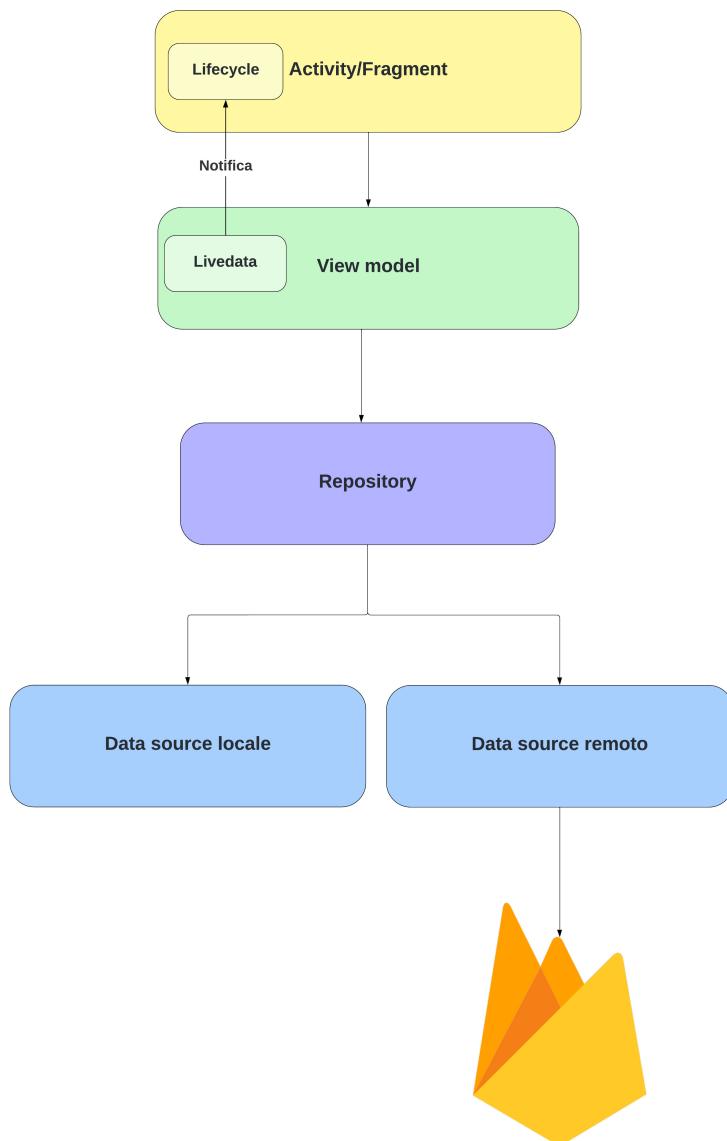


Figura 2.3: Diagramma dell'architettura

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.

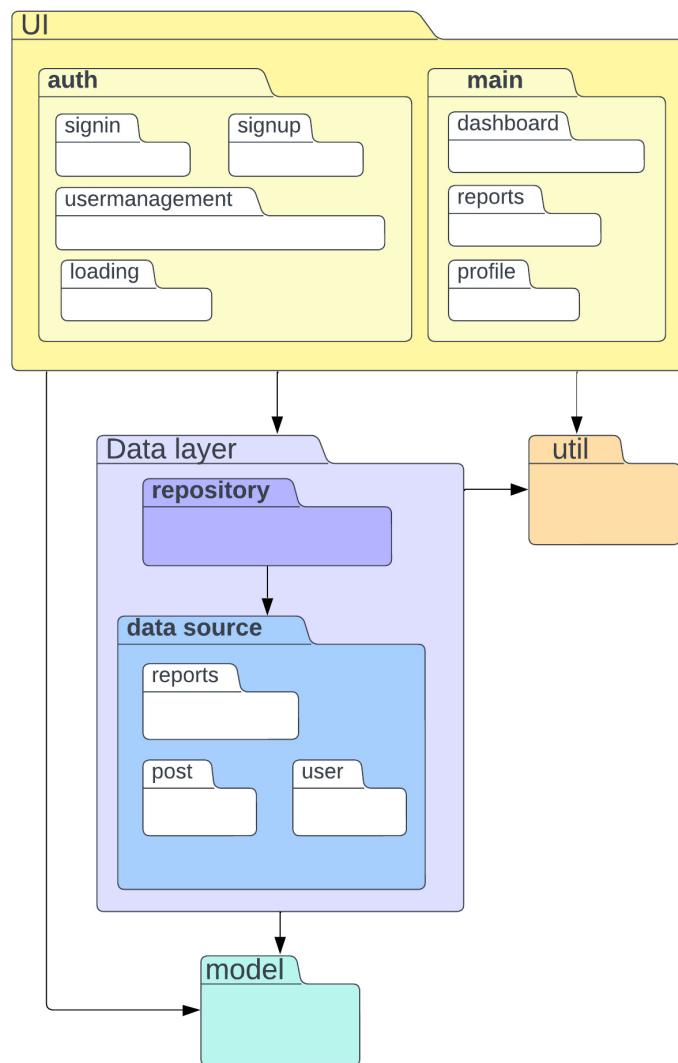


Figura 2.4: Diagramma dei package

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. Verranno illustrati gli aspetti chiave della struttura del codice, le classi utilizzate, le interfacce definite e le principali funzionalità implementate. L'applicazione è stata sviluppata utilizzando il linguaggio di programmazione Java; nella scrittura del codice, l'implementazione è stata suddivisa in quattro sezioni rappresentanti le quattro macro funzionalità: l'autenticazione, la bacheca, le segnalazioni e il profilo.

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 sicuro, tutti coloro che frequentano l'ateneo possono godere di un'applicazione a loro riservata, che non permette l'accesso a coloro che sono 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. Sono richiesti alcuni parametri, che devono rispettare formati precisi e specifici per completare correttamente la procedura:

- **Indirizzo e-mail.** Permette di identificare univocamente l'utente, in quanto non esistono due e-mail universitarie associate alla stessa persona.
- **Password.** Rappresenta una parola chiave che viene scelta dall'utente, che garantisce l'accesso al sistema tramite l'indirizzo e-mail inserito. 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

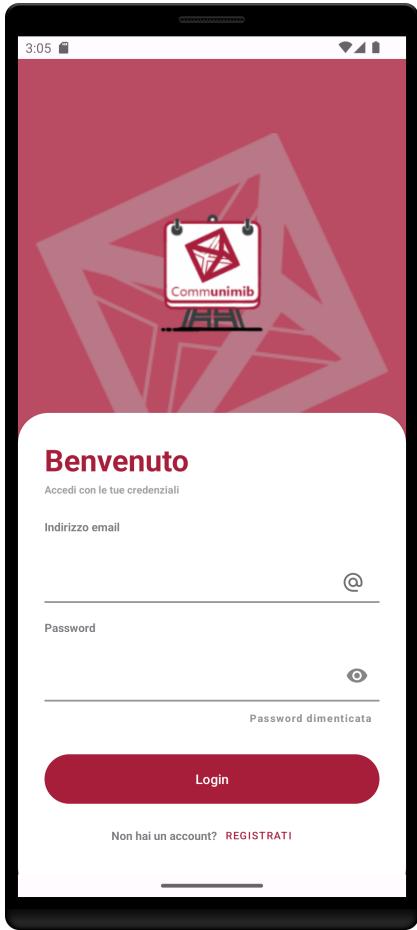


Figura 2.6: Login

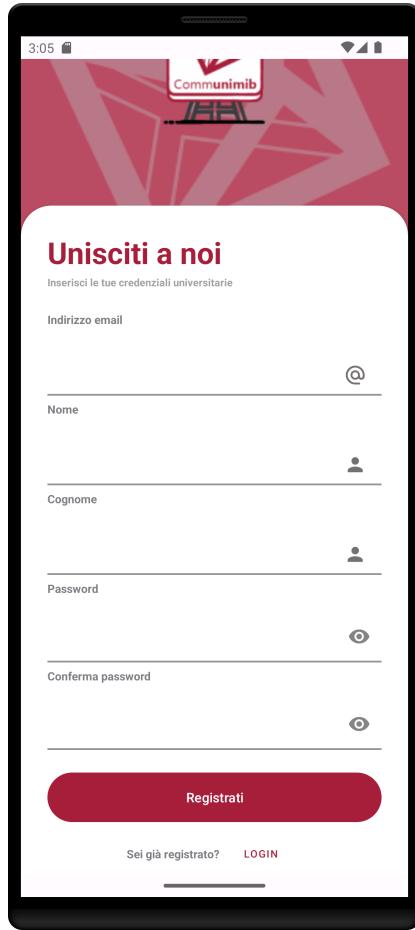


Figura 2.7: Registrazione

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 dunque offre la funzionalità di login (Figura 2.6), che consente ad un utente di autenticarsi nel sistema e procedere con l'utilizzo dell'applicazione.

Per effettuare l'accesso:

1. l'utente inserisce l'e-mail universitaria che è stata memorizzata nel sistema e la password scelta in fase di registrazione.
2. il sistema provvederà a verificare che l'e-mail inserita sia valida.
3. se il sistema non segnala la presenza di errori, con il click sul pulsante di login il sistema effettuerà una query al database.
4. se il sistema riscontra la presenza dell'utente nel database e riconosce le credenziali, l'utente potrà procedere con l'utilizzo dell'app.



Figura 2.8: Conferma mail

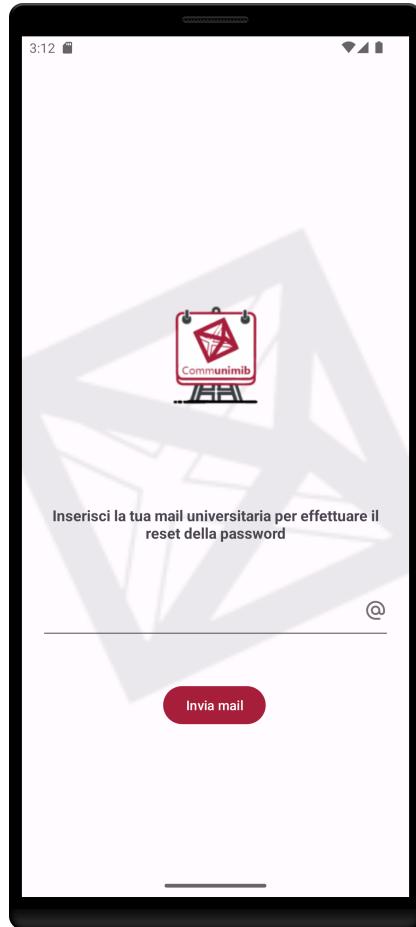


Figura 2.9: Reset password

Verifica e-mail

Ogni sistema di moderna concezione deve misurarsi con il rischio che un utente malevolo possa fornire una e-mail non propria durante la procedura di registrazione: un simile fenomeno può arrecare danni nei confronti del reale 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 (Figura 2.8) 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

La procedura di autenticazione descritta precedentemente 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. Talvolta l'esecuzione di tale meccanismo può richiedere molto tempo (ad esempio, se si utilizza una connessione lenta), di conseguenza 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 (si veda Figura 2.9), 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.

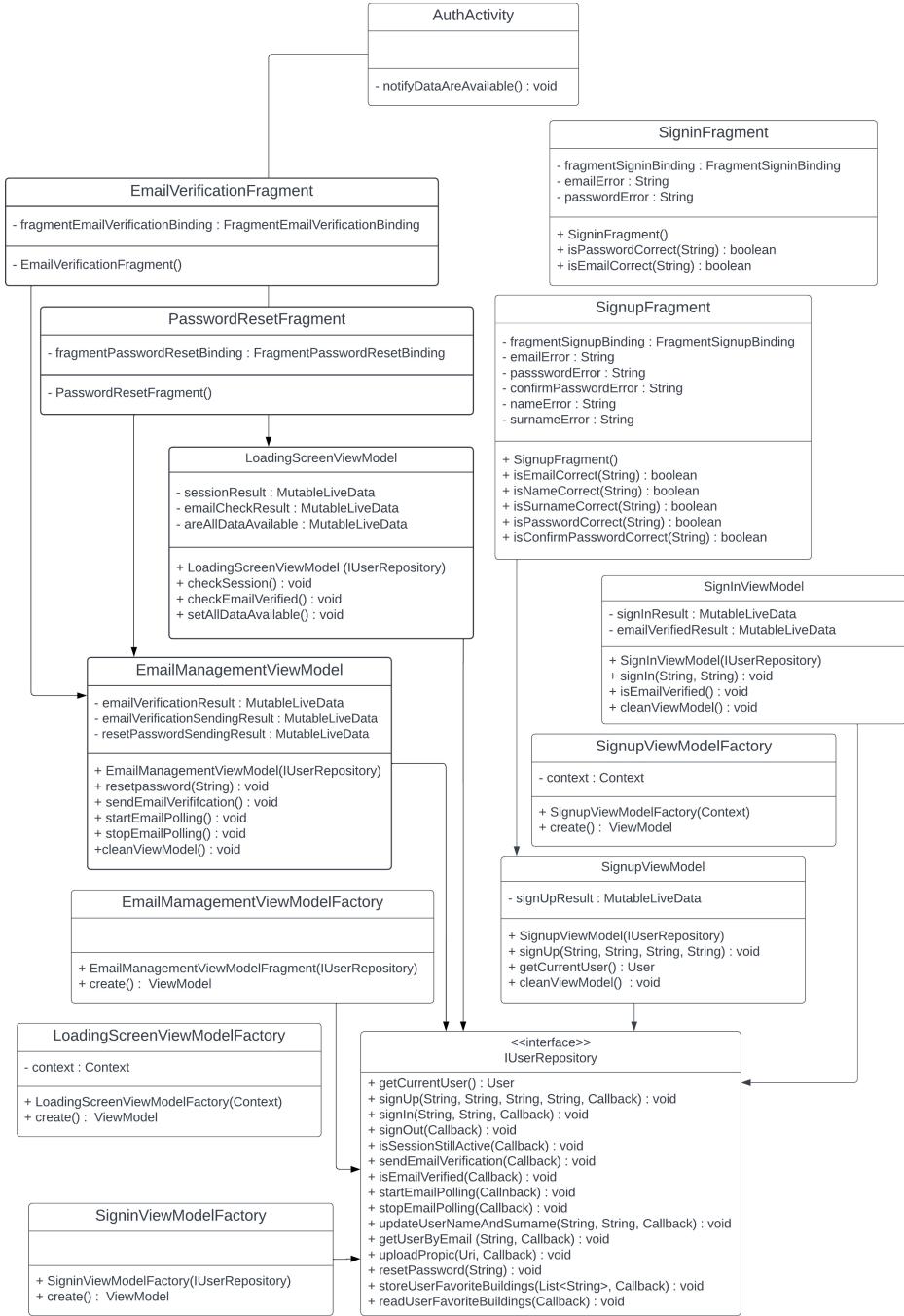


Figura 2.10: Diagramma classi autenticazione - UI layer

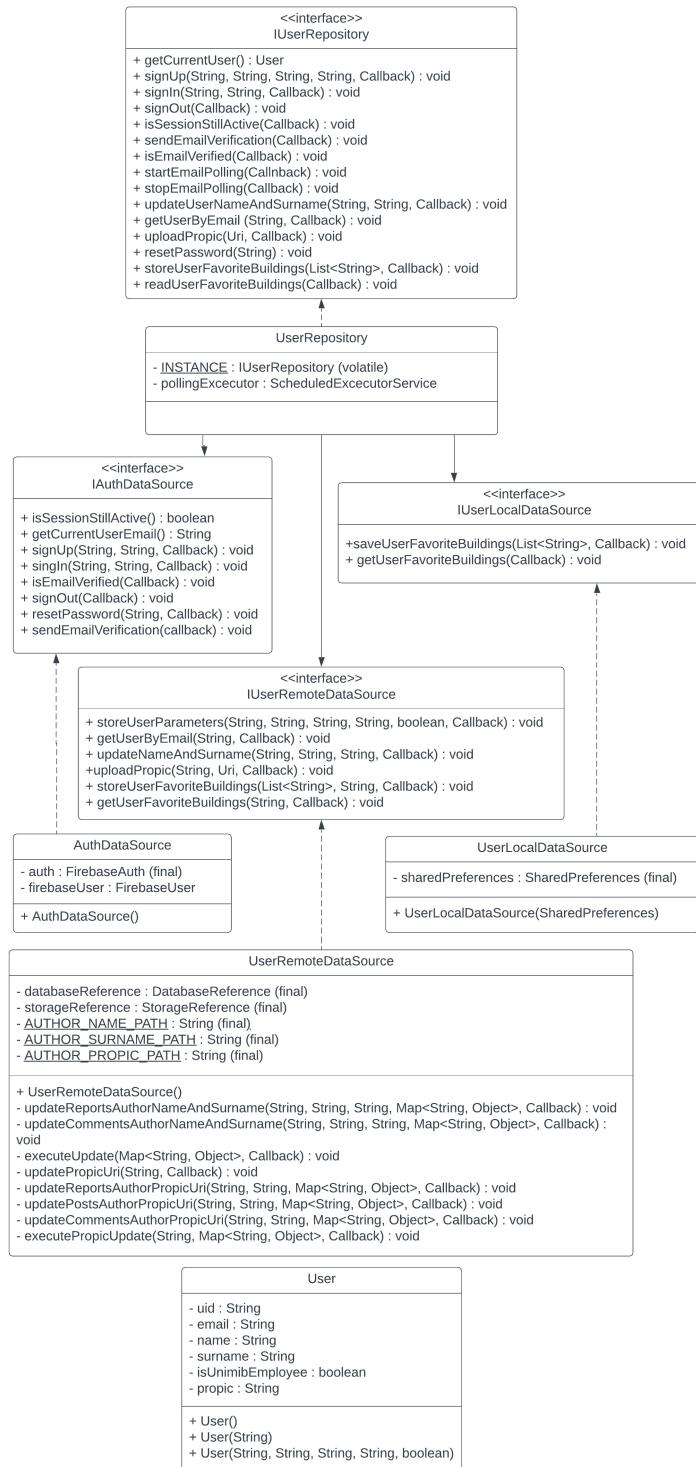


Figura 2.11: Diagramma classi autenticazione - Data layer

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 in quanto la bacheca rappresenta uno spazio condiviso tra molti utenti; ne consegue che 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 tale sezione, 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 in modo diretto ad uno o più post, viene fornita una barra di ricerca che, attraverso l'inserimento di una parola chiave presente nel titolo o nella descrizione del post, consente di effettuare una query più specifica a Firebase; quest'ultimo fornirà solo i post che presentano la parola chiave inserita.

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

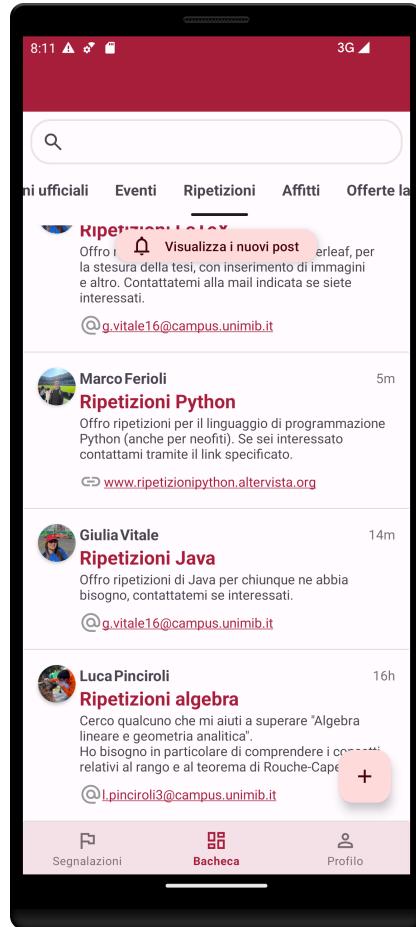


Figura 2.13: Avviso nuovo post

Creazione di un nuovo post

Un utente ha la possibilità di creare un post (si veda Figura 2.14) e di pubblicarlo nel sistema cosicché sia visibile e consultabile anche dagli altri utenti. Per ottimizzare la user experience, la schermata di creazione di un post deve essere semplice da utilizzare e veloce da compilare; per questo motivo, l’utente può decidere di inserire solo i campi relativi al titolo, descrizione e 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 una semplice porzione di testo. Per questo motivo, è stato sviluppato ed utilizzato un componente che permette di caricare un insieme di immagini, che vengono associate al post e mostrate nella schermata. Per effettuare questa operazione è stata utilizzata l’apposita classe `PickVisualMediaRequest`, che effettua un’operazione di `Intent` verso la galleria delle 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 aggiungere un indirizzo e-mail ed un link. Il primo permette di fornire un contatto a chi legge il post, mentre il secondo può rivelarsi utile, ad esempio, per la pubblicizzazione di un evento.

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 su Firebase Realtime Database. Poiché quest'ultimo non supporta la memorizzazione delle immagini, viene quindi utilizzato Firebase Storage per effettuare tale operazione.



Figura 2.14: Creazione post

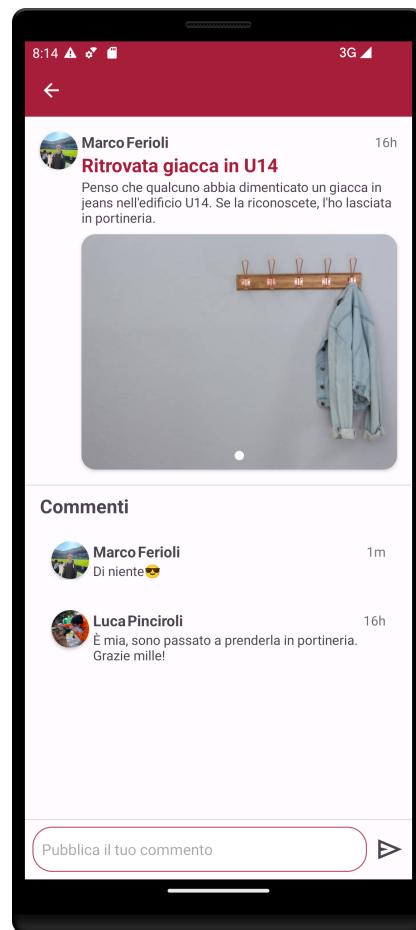


Figura 2.15: Commenti

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 (se, ad esempio, 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.

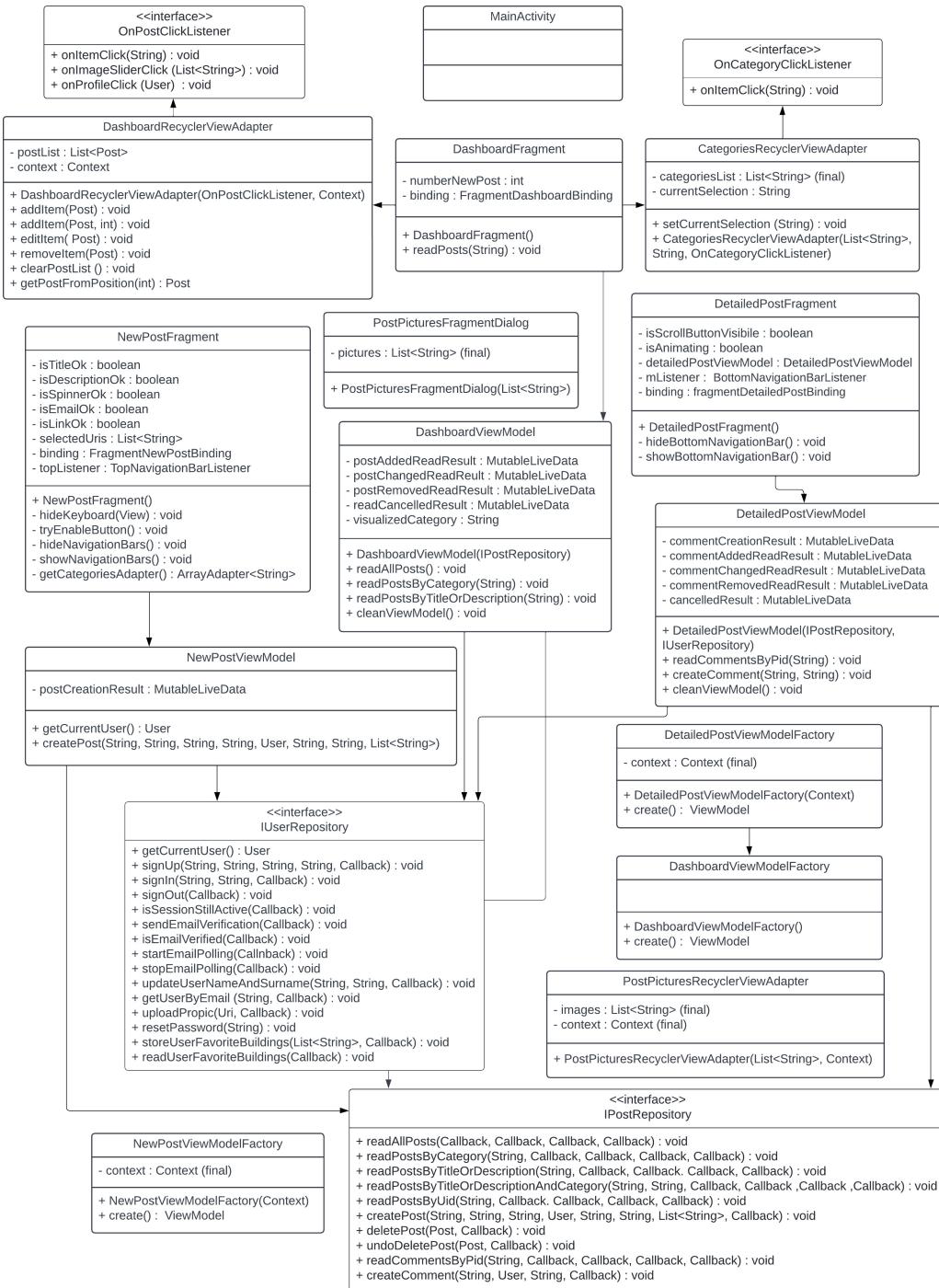


Figura 2.16: Diagramma classi bacheca - UI layer

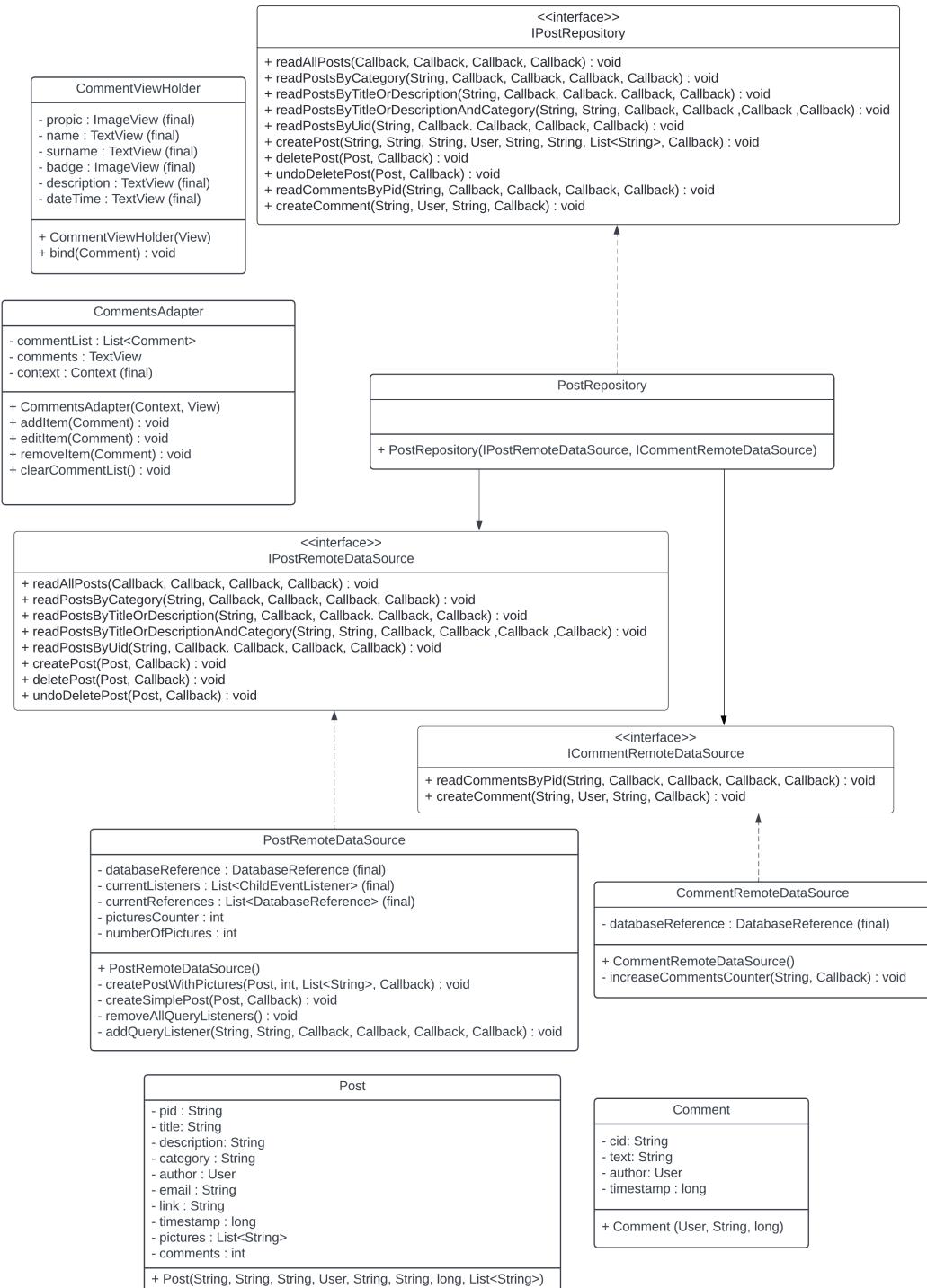


Figura 2.17: Diagramma classi bacheche - Data layer

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 a 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.

Infatti, 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 (si veda Figura 2.22).

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

Al fine di consentire una consultazione più rapida, è stata implementata una barra di ricerca che permette di trovare più velocemente una segnalazione inserendo una parola chiave presente nel titolo o nella descrizione. Ad esempio, questa funzionalità potrebbe rivelarsi utile per un addetto alla manutenzione che deve risolvere un problema.

Infine, il meccanismo utilizzato per l'aggiornamento real-time delle segnalazioni è il medesimo usato nella bacheca.

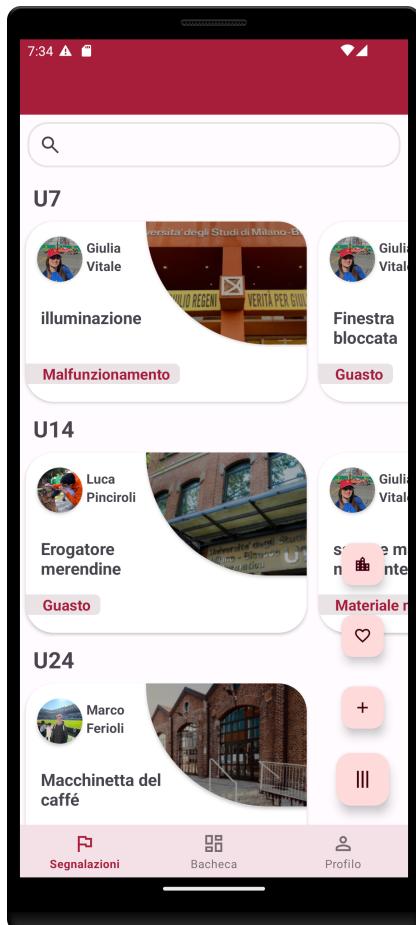


Figura 2.18: Segnalazioni

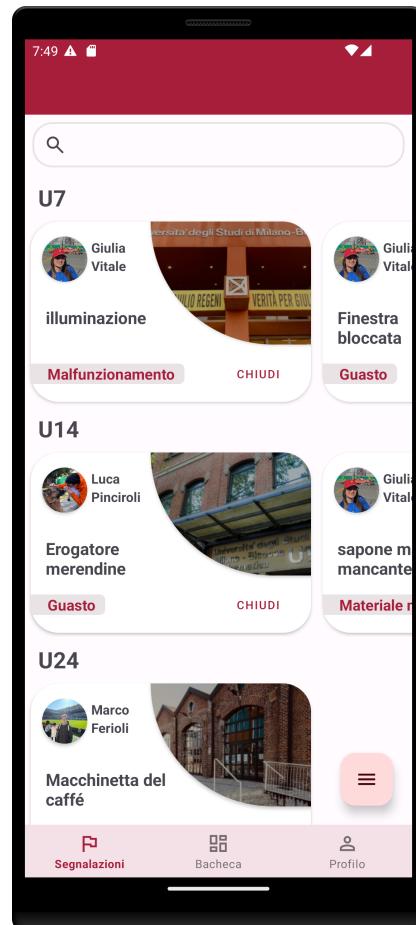


Figura 2.19: UI dipendente

Filtri ed edifici preferiti

Un utente deve avere la possibilità di selezionare solo un gruppo di segnalazioni tra quelle presenti nel sistema. Per rispondere a questo requisito, sono stati utilizzati dei `DialogFragment`. Questi componenti, messi a disposizione dalla libreria Android, si comportano come dei normali `Fragment` dal punto di vista programmatico e del ciclo di vita; tuttavia presentano 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 lista 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 l'adattabilità e 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; ad esempio un utente potrebbe essere interessato solo alle segnalazioni relative ad un edificio specifico. 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 quindi di leggere localmente i dati per evitare chiamate superflue sulla piattaforma remota, mantenendo comunque sincronizzati gli edifici preferiti su dispositivi differenti.



Figura 2.20: Selezione preferiti

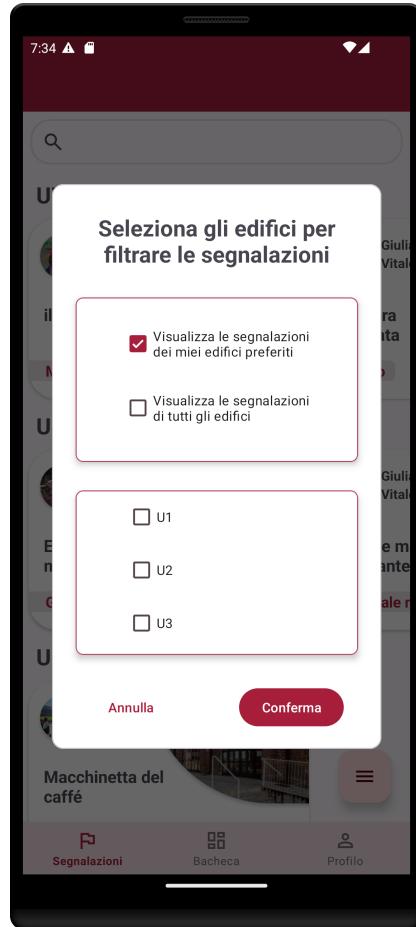


Figura 2.21: Selezione filtri

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, 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 riassumono la 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.

Poiché una delle due operazioni chiave è riservata al solo personale universi-

tario, 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

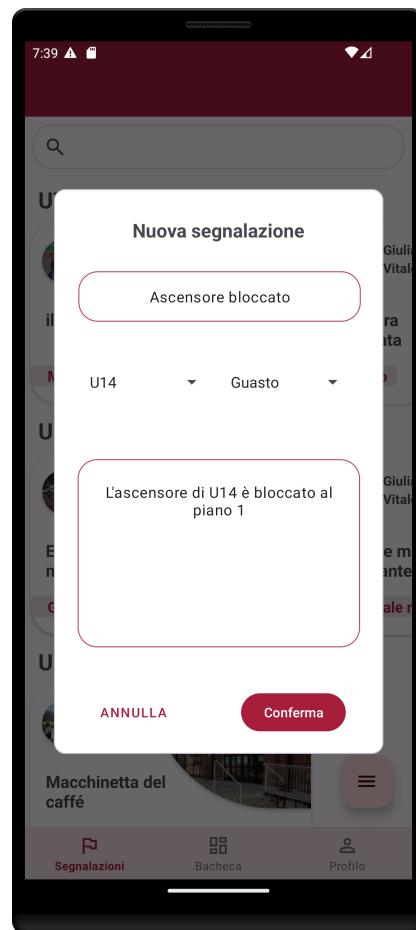


Figura 2.23: Creazione segnalazione

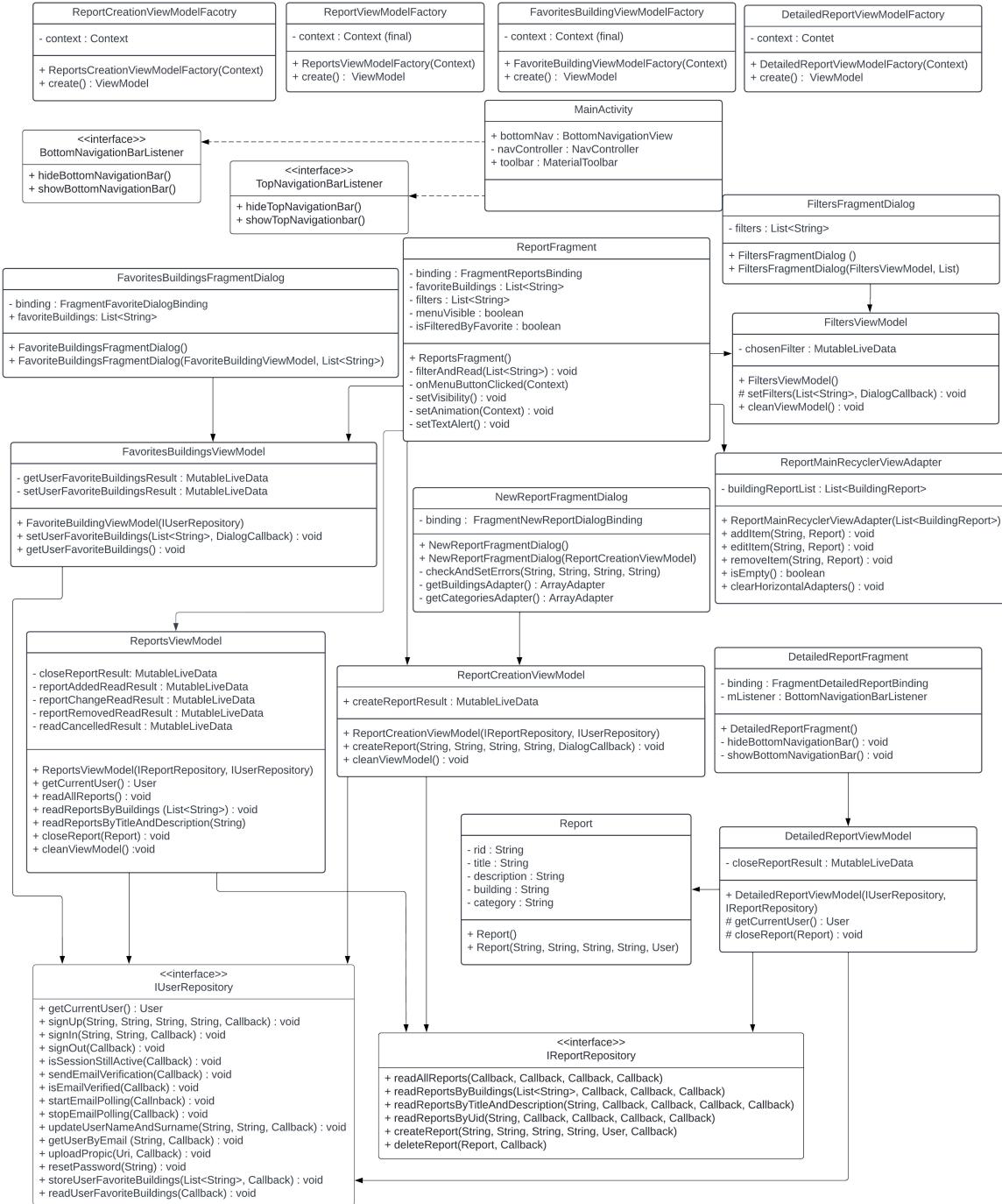


Figura 2.24: Diagramma classi segnalazioni - UI layer

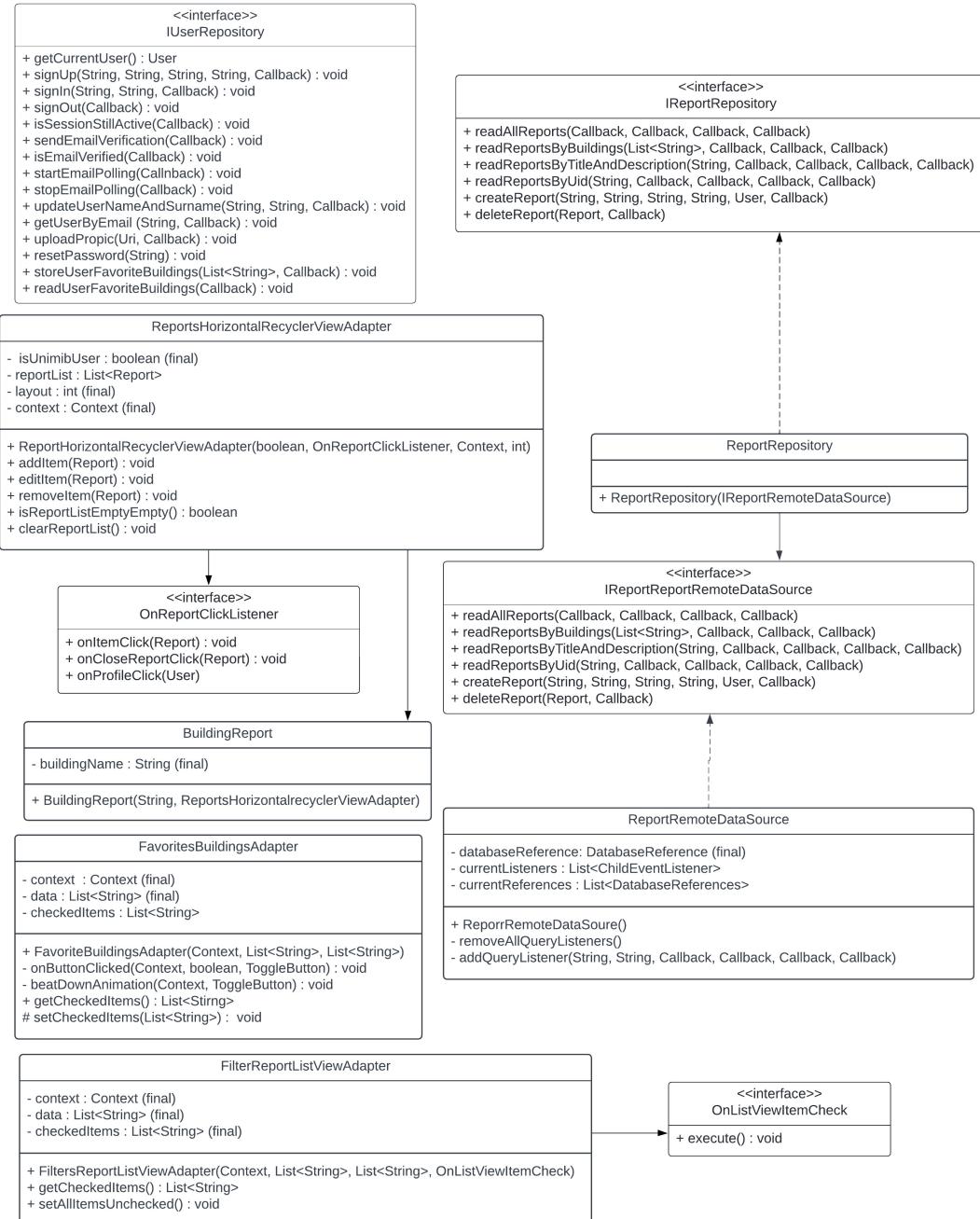


Figura 2.25: Diagramma classi segnalazioni - Data layer

2.2.4 Profilo

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

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 e alle segnalazioni.

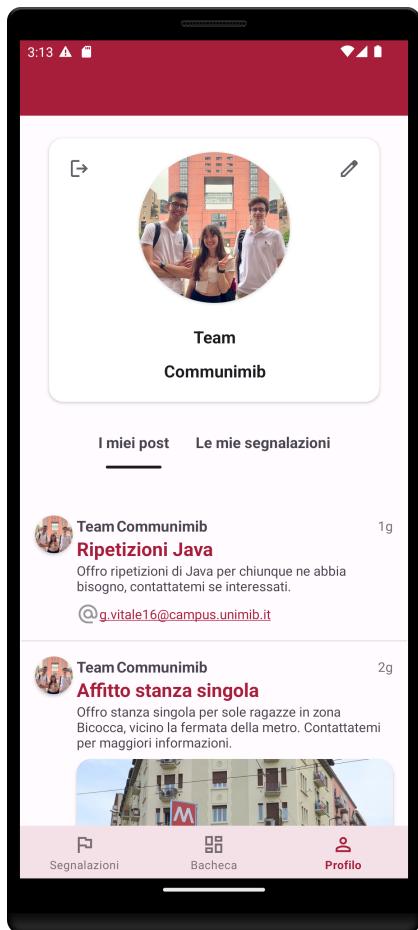


Figura 2.26: Profilo

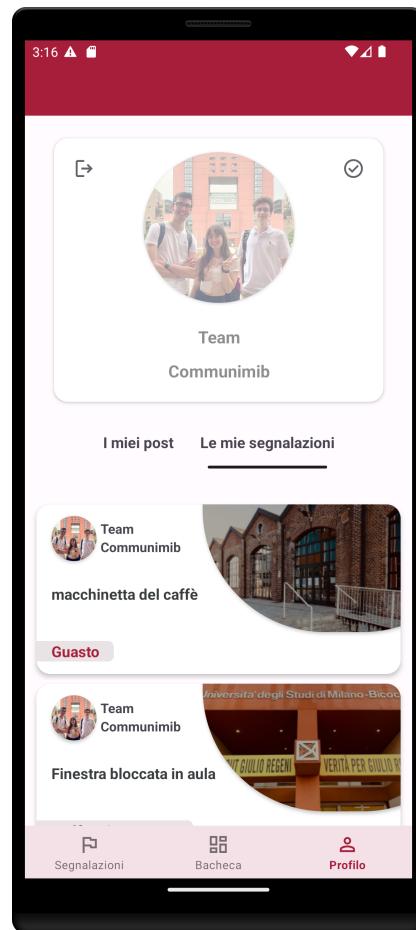


Figura 2.27: Modifica profilo

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 necessaria l'implementazione di un sistema che consenta di selezionare solo una porzione rotonda dell'immagine.

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 nella 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 a seguito della risoluzione del problema corrispondente, i post possono permanere all'interno del sistema per un tempo indefinito; 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 moderno, intuitivo 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.

Poiché l'operazione di cancellazione potrebbe 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; quando 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.

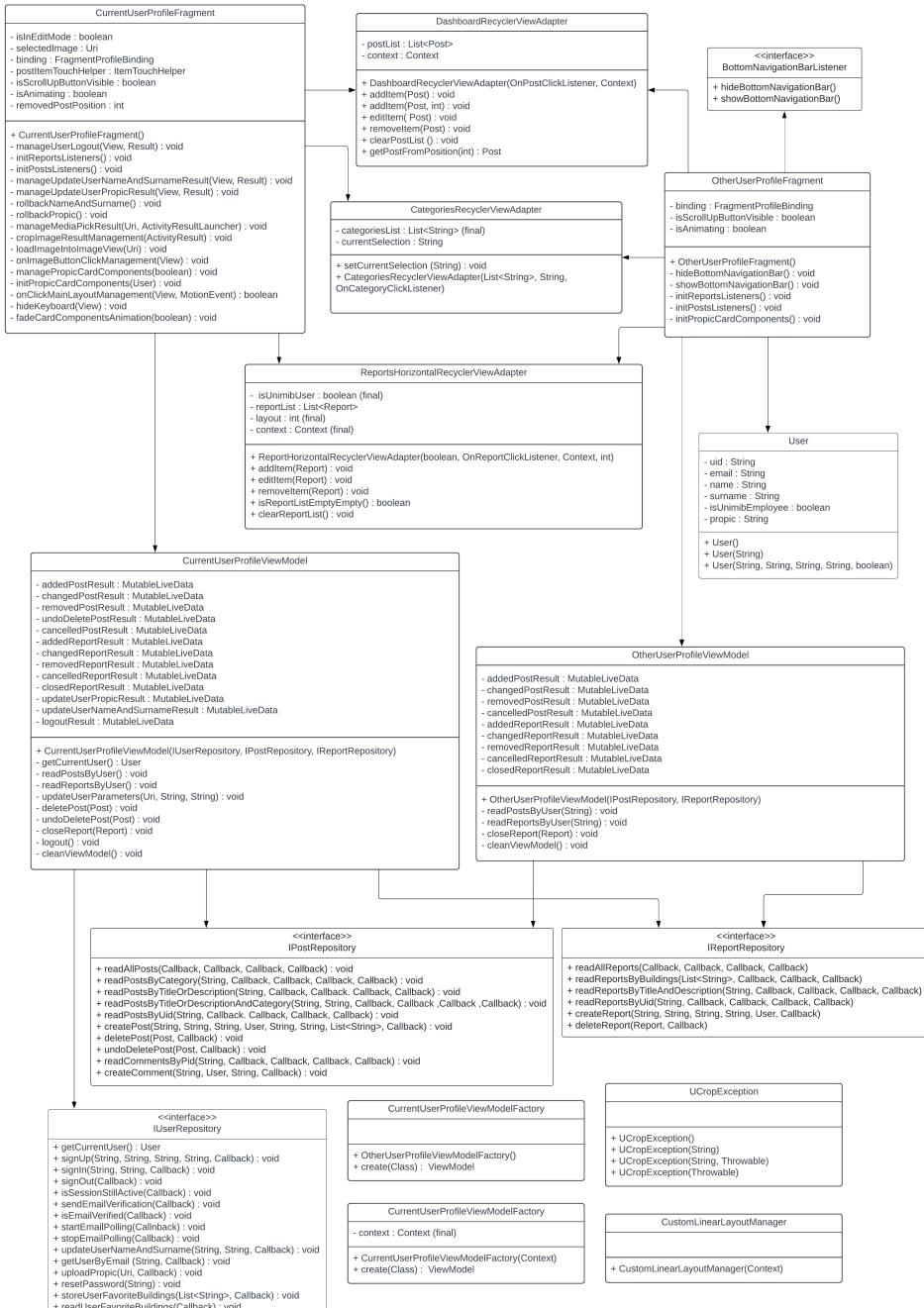


Figura 2.28: Diagramma classi profilo - UI layer

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 offre agli utenti la possibilità di creare la propria identità 'virtuale' deve garantire un adeguato livello di sicurezza. Per questo motivo, il sistema di autenticazione si basa su un servizio robusto e affidabile come Firebase Authentication.

Durante la procedura di registrazione 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 elevato 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 restare 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 di generare punti di inconsistenza che possono portare a malfunzionamenti nell'applicazione.

Sulla base di queste premesse, le operazioni di aggiornamento sono state gestite con particolare cura ed in maniera atomica, al fine di evitare interruzioni che potrebbero lasciare i dati in uno stato inconsistente; nel caso di Communimib, 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.

Firebase Storage

Poiché 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 associate ai post. Le operazioni di upload e download delle immagini si basano sull'utilizzo di URI per la loro identificazione univoca; poiché Firebase Realtime Database è in grado di contenere soltanto informazioni testuali, le immagini sono state associate 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, in quanto permette di verificare e validare il corretto funzionamento dei componenti implementati prima del loro rilascio.

Sebbene i test manuali effettuati dagli sviluppatori tramite l'utilizzo dell'emulatore 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 sono stati sviluppati ed utilizzati due tipologie di test tramite l'utilizzo della libreria JUnit4 [5]:

1. **test di 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, perché permette di testare singolarmente ciascun componente.
2. **test UI**: consentono di validare il funzionamento dell'interfaccia utente e generalmente consistono nell'esecuzione di un'activity in cui vengono simulati dei comandi, che rappresentano delle azioni generate dall'utente, con lo scopo di verificare il corretto funzionamento dell'interfaccia.

Poiché il progetto è stato suddiviso in quattro iterazioni (una per sezione), al termine di ciascuna è stato effettuato il testing di quanto implementato. Questa operazione ha permesso di validare il funzionamento delle classi prodotte, che sono quindi state riutilizzate nelle fasi implementative successive.

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 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` cosicché sia il primo metodo ad essere eseguito, prevede che al suo interno siano istanziate tutte le classi e oggetti utili per il testing, come la classe Repository da testare e i `mock()` delle classi Datasource usate dal Repository.
- Per ogni metodo:
 1. **Invocazione metodo doAnswer().when():** questo metodo consente di delineare 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 verifica 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.

Dato 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 avviene alcun aggiornamento, il metodo `getOrAwaitValue` restituirà un'eccezione.

I valori presenti sui LiveData, sono stati successivamente 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, al fine di validare il funzionamen-

to 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, all’interno di tale metodo, 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 il testing dei 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

Il crescente progresso dell'intelligenza artificiale ha portato alla nascita di nuove tecnologie intelligenti che, dovutamente addestrate, si dimostrano capaci di effettuare task autonomamente, senza l'assistenza dell'uomo. Tra queste tecnologie, quella che è stata maggiormente oggetto di discussione, viste le sue potenzialità, è ChatGPT.

ChatGPT è un modello di linguaggio sviluppato da OpenAI, basato sull'architettura GPT-4; l'acronimo "GPT" indica "Generative Pre-trained Transformer", una tecnologia di intelligenza artificiale che utilizza le reti neurali per generare testi simili a quelli prodotti dall'uomo; è un chatbot specializzato nella conversazione con un utente umano ed è capace di generare testo simile a quello usato dalle persone, tanto da superare con risultati ottimi il test di Turing (il test suggerito da Alan Turing per determinare se una macchina è in grado di mostrare un comportamento intelligente).

ChatGPT rappresenta un ausilio prezioso per le attività umane, offrendo versatilità di impiego in una vasta gamma di settori: effettuare una ricerca, imparare una lingua, scrivere un testo o un riassunto, fornire spiegazioni, supportare un programmatore nello sviluppo di un software.

Quest'ultimo fattore costituisce il fulcro attorno al quale è stato concepito e realizzato il presente esperimento: si vuole verificare che livello di supporto può fornire al programmatore un chatbot come ChatGPT durante lo sviluppo di un software, misurando l'accuratezza delle risposte fornite e la precisione nell'individuazione del problema con l'obiettivo di determinare se le soluzioni proposte siano immediatamente applicabili o se richiedano ulteriori rielaborazioni prima di poter essere utilizzate.

A tal fine, si è deciso di sviluppare Communimib, avvalendosi di ChatGPT come strumento di supporto per lo sviluppo; di seguito, divisi secondo le principali fasi di sviluppo, sono descritti gli utilizzi di chatGPT che hanno suscitato maggiore interesse.

4.0.1 Analisi dei requisiti

L’analisi dei requisiti riveste un ruolo fondamentale nello sviluppo del software poiché definisce chiaramente le funzionalità che il software deve svolgere e le modalità con cui dovrà farlo, prevenendo errori costosi nelle fasi successive del processo.

L’unico aspetto discusso con ChatGPT durante questa fase è la ricerca del nome dell’applicativo. Il nome, infatti, gioca un ruolo importante nella conquista dell’utente al fine di incoraggiarlo ad utilizzare l’applicazione; per tale ragione è importante che questo sia accattivante, originale e rappresentativo.

Il quesito proponeva la combinazione della parola ’Bicocca’ o ’unimib’ con un termine che rimandasse al concetto di comunità, società e gruppo, con l’intento di ricordare il legame che si crea tra coloro che frequentano l’ambiente universitario.

Nonostante il nome scelto non è tra le soluzioni suggerite dal chatbot, da questa chat è emersa la capacità di ChatGPT di formulare nomi creativi, inizialmente proponendo combinazioni tra i termini specificati nella richiesta, successivamente suggerendo soluzioni che contenessero sinonimi di questi, per infine proporre accostamenti a tratti fuori dal comune come ’Sphere’ per simbolizzare una comunità unita e ’Hub’ per rappresentare il luogo centrale dove la comunità si ritrova.

Durante la fase di analisi dei requisiti ho apprezzato il confronto di idee con il team, dunque non ho ritenuto necessario utilizzare ulteriormente ChatGPT. Inoltre, pur essendo un potente modello di linguaggio basato sull’intelligenza artificiale, ChatGPT ha alcune limitazioni legate alla sua capacità di comprendere pienamente il contesto specifico e le esigenze dettagliate di un progetto.

Nella fase di analisi dei requisiti, è essenziale ottenere una comprensione approfondita delle funzionalità che si vogliono implementare e dei vincoli tecnici che influenzano il design e lo sviluppo del sistema. Fare affidamento a ChatGPT può comportare il rischio di interpretazioni imprecise o incomplete delle specifiche, che potrebbero compromettere la qualità complessiva del progetto.

4.0.2 Descrizione dell’implementazione

L’implementazione è la fase a cui è stato dedicato più tempo, in quanto prevede che vengano realizzate le funzionalità definite nell’analisi dei requisiti. In tale fase il supporto di ChatGPT è stato maggiore; quest’ultimo, infatti, è in grado di generare codice il quale può essere immediatamente applicato al contesto di sviluppo per risolvere una problematica quale un crash o un bug. Tuttavia, il codice fornito non sempre è pronto all’utilizzo, ma può accadere che questo generi errori data la presenza di variabili o valori non definiti, oppure sia incompleto, rappresenti una soluzione parziale, rendendo necessaria la rielaborazione e l’integrazione con altre fonti. Nonostante ciò, l’assistenza di ChatGPT si è rivelata utile durante l’implementazione per la realizzazione di alcuni componenti dell’interfaccia utente.

Una delle questioni più rilevanti, in cui il supporto del chatbot si è dimostrato utile, è stata la gestione dei filtri; durante lo sviluppo dell'applicativo è stata ricercata, sia per la bacheca che per le segnalazioni, una modalità di visualizzazione che permettesse all'utente di acquisire in modo facile ed intuitivo le informazioni desiderate, evitando contestualmente quelle non necessarie.

A tale scopo sono state poste diverse domande a ChatGPT per esplorare le opzioni disponibili al fine di individuare quella esteticamente più gradevole. L'idea iniziale prevedeva l'implementazione del componente Navigation Drawer che presto però si è rivelato poco efficace rispetto al funzionamento richiesto; a conferma di ciò, ChatGPT non ha suggerito l'utilizzo di tale componente.

Seguendo il modello di molte applicazioni esistenti sul mercato è stata individuata una modalità di visualizzazione delle segnalazioni che le raggruppasse secondo l'edificio a cui queste si riferiscono. In questo contesto ChatGPT si è rivelato d'aiuto poiché ha compreso tempestivamente il problema e ha suggerito l'utilizzo di due componenti RecyclerView con due relativi Adapter, una su cui effettuare uno scroll orizzontale, la cui funzione è quella di mostrare le segnalazioni, e una RecyclerView principale che contenesse le RecyclerView orizzontali e permetta di effettuare uno scroll verticale. Tuttavia, nonostante la soluzione proposta sia stata utile dal punto di vista teorico e abbia fornito diversi spunti di implementazione, nell'applicazione pratica, si è rivelata incompleta facendo emergere la necessità di ulteriori elaborazioni.

In merito alla bacheca, si è optato per una più semplice visualizzazione verticale, senza rinunciare a un modo gradevole e innovativo per rappresentare il sistema di filtri. A tal proposito, è stato richiesto il parere del chatbot, il quale si è espresso favorevolmente, riguardo alla predisposizione di una RecyclerView orizzontale, posizionata sotto la barra di ricerca. Tale configurazione consente di visualizzare le categorie disponibili e, al tap su una di esse, di attivare un meccanismo che mostra solo i post relativi alla categoria selezionata, rendendo così la visualizzazione più piacevole ed efficace. Per garantire la reattività della schermata, è stato chiesto a ChatGPT se esistesse un modo per modificare il layout di un elemento della RecyclerView al click dell'utente, così che questo sia consci della categoria che sta visualizzando. Il chatbot ha quindi proposto alcune soluzioni come l'uso di animazioni e il cambio di colore o di stato dell'elemento.

Un suggerimento particolarmente interessante ha riguardato l'utilizzo dei file XML per lo styling delle componenti dell'interfaccia utente. Il chatbot ha proposto la creazione di un file XML specifico per personalizzare la forma di una TextView, suggerendo cosa inserire nel file affinché il risultato rispecchiasse le richieste formulate. Tale soluzione è stata applicata in diversi contesti e ha permesso l'adattamento delle componenti al design dell'applicativo.

Sono state inoltre poste diverse domande relative alla correzione (fixing) di alcune componenti dell'interfaccia utente. Sebbene le risposte ottenute si siano rivelate utili, hanno richiesto ulteriori rielaborazioni e revisioni per essere pienamente efficaci e integrate correttamente nel progetto. Questo processo ha permesso di migliorare la qualità delle soluzioni implementate, adattandole meglio alle specifiche esigenze del team.

Infine, un numero significativo di domande è stato incentrato sull'immen-

tazione del sistema delle notifiche. Il chatbot ha fornito una panoramica dettagliata del funzionamento generale, suggerendo l'uso di Firebase Cloud Messaging e spiegando in modo chiaro il meccanismo dei token. Nonostante l'utilità delle informazioni ottenute, è stato deciso di rimandare l'implementazione del sistema di notifiche agli sviluppi futuri, per garantire una maggiore attenzione alle altre priorità del progetto e assicurare una realizzazione più completa e integrata in una fase successiva.

4.0.3 Progetto di testing

Il progetto di testing rappresenta la fase dello sviluppo che ha maggiormente goduto del supporto di ChatGPT. Quest'ultimo si è dimostrato prezioso in quanto ha fornito una panoramica dettagliata sui tipi di test disponibili, inclusi i test di integrazione e specifiche informazioni sul testing di Firebase. In aggiunta, ChatGPT ha spiegato in modo esaustivo il funzionamento dei test di unità e UI, influenzando positivamente l'implementazione di questi approcci nel progetto. Questa assistenza ha contribuito a migliorare la qualità complessiva del processo di testing, garantendo una maggiore copertura e affidabilità delle funzionalità sviluppate.

Il supporto di ChatGPT si è rivelato fondamentale nella stesura dei test di unità, poiché il chatbot è stato capace di chiarire quali fossero le classi da testare e le modalità corrette per farlo.

Secondo la suddivisione in livelli dell'architettura software, ogni componente utilizza una classe del livello inferiore per fornire un risultato a quella del livello superiore; ciò comporta che per testare una classe, ad esempio un Repository, che utilizza un Datasource per fornire un risultato al ViewModel, si rende necessario simulare il funzionamento del Datasource. A tale scopo il chatbot ha suggerito l'utilizzo del framework Mockito per creare mock delle classi, cioè degli oggetti simulati che vengono utilizzati per sostituire parti reali del software durante l'esecuzione dei test. I mock, infatti, replicano il comportamento di porzioni di codice dell'applicazione che non sono direttamente coinvolte nell'esecuzione dello specifico test.

In tal modo è stato possibile definire i valori che la classe mock è tenuta a restituire quando un suo specifico metodo viene invocato, riproducendo così il comportamento della classe originale.

ChatGPT ha inoltre illustrato in modo dettagliato i principali metodi offerti da Mockito per facilitare il testing. Tra questi, i metodi `verify()`, `doAnswer()` e `doReturn()` sono stati particolarmente evidenziati per la loro utilità, rispettivamente, nel verificare il comportamento degli oggetti, nel definire risposte personalizzate e nel simulare valori di ritorno. Queste spiegazioni hanno contribuito a migliorare la comprensione e l'applicazione delle tecniche di testing all'interno del progetto.

Un'ulteriore richiesta posta a ChatGPT riguardante i test di unità ha avuto come oggetto la gestione degli observer durante il testing dei ViewModel; il chatbot ha fornito alcune soluzioni, come l'utilizzo di un mock della classe `Observer`. Tuttavia, successivamente, è stato deciso di adottare una differente metodologia.

In merito ai test UI, ChatGPT ha suggerito l'utilizzo della libreria Espresso, al fine di facilitare la stesura e migliorarne l'efficacia.

Le richieste formulate al chatbot sono state finalizzate all'implementazione di test più specifici, come quello relativo alla visualizzazione di un'immagine sulla schermata successivamente al suo caricamento dalla galleria dell'utente. Per effettuare questo test si è reso necessario simulare sia l'apertura della galleria dell'utente che il selezionamento di un'immagine; a tale scopo è stato chiesto al chatbot un modo per effettuare tale simulazione. In questo caso il chatbot non si è rivelato d'aiuto, poiché ha suggerito l'utilizzo di un Intent, implementando un sistema per sua la gestione, ma nonostante siano state seguite tutte le indicazioni fornite, il codice non risultava funzionante.

Inoltre, un'altra domanda rivolta a ChatGPT ha riguardato la gestione della navigazione nella `MainActivity`, indagando sull'esistenza di un modo per simulare il tap sul componente BottomNavigationView al fine di navigare alla schermata oggetto di test. Sono state necessarie due chat in questo caso: alla prima chat il chatbot ha suggerito l'utilizzo di una metodologia che ha funzionato su una schermata, ma che si è rivelata inutilizzabile per testare le altre. Sono state dunque chieste spiegazioni ed è stato il chatbot stesso ad affermare che il metodo precedentemente suggerito non era quello adatto allo scopo; ChatGPT ha dunque proposto un'altra modalità, l'implementazione dell'interfaccia `ViewAction`, la quale si è rivelata applicabile su tutte le schermate ed è stata quindi adottata nel progetto.

4.0.4 Considerazioni finali

Dagli utilizzi sopra citati, si riscontra che, sebbene ChatGPT abbia mostrato un elevato supporto nello sviluppo di alcune funzionalità, in altri casi l'aiuto prestato dal chatbot non si è rivelato sufficiente al completamento del task.

In particolare, è evidente che il chatbot sia altamente performante nell'offrire spiegazioni accurate, fornendo a chi lo utilizza le informazioni necessarie e i passaggi da seguire per portare a termine un determinato compito. Tuttavia, nonostante le spiegazioni siano corrette, all'applicazione pratica, il codice generato talvolta rivela la presenza di errori, come valori non definiti che possono suscitare confusione, o errori in compilazione che impediscono l'esecuzione del programma. Per tale ragione in alcuni casi il confronto con ChatGPT si rivela inutile.

Al fine di presentare in modo efficace i risultati dell'esperimento, di seguito sono mostrati due grafici che evidenziano l'utilità del supporto di ChatGPT durante le diverse fasi dello sviluppo.

Per valutare l'efficacia delle interazioni con ChatGPT, queste sono state classificate in base al loro grado di utilità come "utile", "parzialmente utile" e "inutile". Su un totale di 23 utilizzi di ChatGPT, si è osservato che 13 sono stati giudicati utili, 7 parzialmente utili e 3 inutili.

Segue il grafico dei risultati.

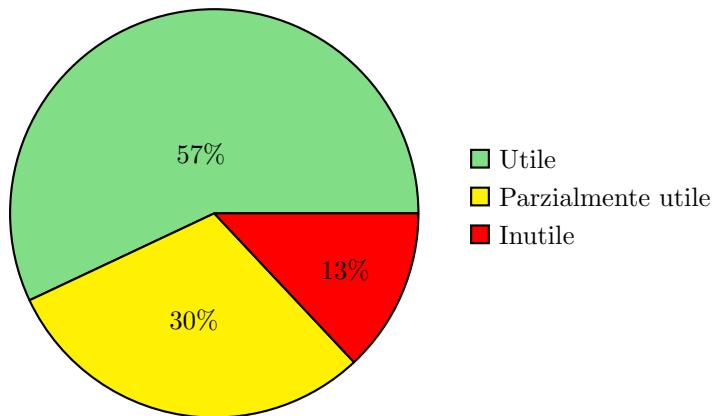


Figura 4.1: Grafico del grado di utilità

Nell'intero sviluppo del software, ChatGPT ha contribuito approssimativamente al 50% del lavoro, fornendo suggerimenti rispetto alle modalità ideali da adottare per l'implementazione di alcune funzionalità. Tale supporto è stato assente durante la fase di analisi dei requisiti, mentre è stato moderato durante l'implementazione, rappresentando circa il 40% del contributo totale, dato dagli spunti teorici offerti e i suggerimenti usati per il fixing. Durante il testing, il contributo è stato più significativo, corrispondente a circa il 60% del totale, grazie ai suggerimenti riguardanti l'uso di Mockito ed Espresso.

Il seguente grafico riassume le percentuali di utilizzo nelle diverse fasi.

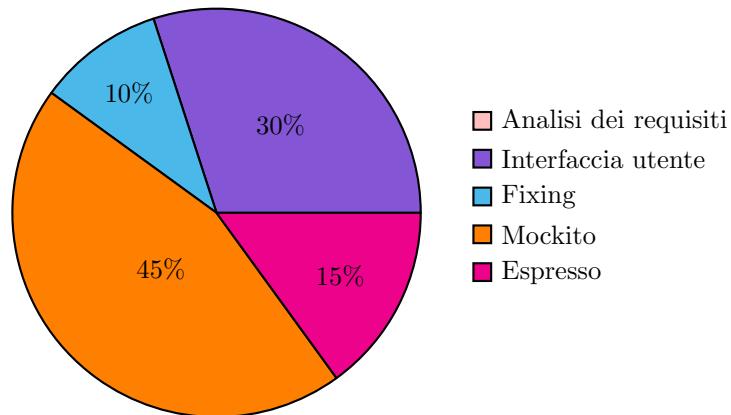


Figura 4.2: Grafico dell'utilità durante le fasi dello sviluppo

Conclusioni

In questa tesi è stato condotto un esperimento mirato a testare la capacità di un LLM, ChatGPT, nel supportare il programmatore durante lo sviluppo software, attraverso suggerimenti e infomazioni utili.

In particolare, il software prodotto durante l'esperimento è l'applicazione "Communimib", sviluppata nell'ambito dell'ateneo. Al fine di testare il supporto di ChatGPT e non la capacità di sostuiрsi al programmatore, l'interazione è stata ricercata solo nei casi in cui si siano manifestati dubbi durante lo sviluppo dell'applicativo e si sia ritenuto potenzialmente utile l'ausilio del chatbot. Inoltre, questi risultati fanno riferimento a un solo progetto software; risultati diversi potrebbero emergere includendo nell'analisi una più ampia varietà di progetti e linguaggi di programmazione.

In generale, ChatGPT si è dimostrata un'ottima fonte di informazioni nel caso in cui la domanda posta sia generica, mirata a raccogliere informazioni su uno specifico tema, in quanto il chatbot è in grado di fornire una ricca panoramica rispetto all'argomento richiesto. Se, al contrario, la richiesta viene fatta da un utente che conosce il tema e necessita di una soluzione implementativa specifica, il chatbot potrebbe manifestare alcune difficoltà. Ciò è legato a una limitazione fondamentale di ChatGPT: la dipendenza dal contesto. L'efficacia di ChatGPT è infatti strettamente legata alla qualità delle richieste formulate dagli sviluppatori. Una formulazione imprecisa può portare a risposte meno accurate e poco utili. Si rende quindi necessario effettuare delle richieste dettagliate, che descrivano minuziosamente il problema, il contesto di applicazione, le variabili coinvolte, le cause e le conseguenze. Tuttavia, ciò può non essere sufficiente. Infatti, in alcuni casi è stato riscontrato che le soluzioni proposte non erano applicabili o, più precisamente, il codice non era corretto o risultava incompleto. La generazione del codice è uno dei casi in cui si manifestano maggiormente le limitazioni del chatbot, in quanto, in mancanza di un contesto ben definito, il codice proposto risulta talvolta di scarsa efficacia rispetto al problema.

Inoltre, è stata riscontrata un'incapacità del chatbot nella correzione dei propri errori. Nelle diverse interazioni, ogni qualvolta il codice presentava errori, è sempre stata segnalata a ChatGPT la presenza di tali imprecisioni; tuttavia, il chatbot non è sempre riuscito a correggerle. In alcuni casi infatti, il chatbot si è limitato a rigenerare lo stesso codice, senza apporre le necessarie modifiche.

Un aspetto interessante riscontrato durante l'esperimento riguarda le ottime prestazioni di ChatGPT quando viene sottoposto a un consulto, cioè nei casi in cui venga proposta al chatbot una possibile soluzione a un problema dato e

si chieda se questa sia applicabile e corretta o se, al contrario, sia da scartare. A conferma di ciò, due delle chat intervenute con ChatGPT hanno visto il chatbot affermare nella seconda chat che il codice generato durante la prima non era corretto. Nella prima chat era stato fornito un contesto dettagliato ed era stato posto un quesito, al quale ChatGPT ha risposto generando una porzione di codice. Nella seconda chat, è stato inserito nella richiesta fatta al chatbot il codice generato precedentemente; ChatGPT ha valutato il codice e ha affermato che non era corretto, fornendo una versione migliore. Ciò dimostra che le performance migliorano quando viene fornito al chatbot un termine di confronto su cui basare la risposta.

In conclusione, dai risultati dell'analisi condotta si deduce che gli LLM hanno le capacità necessarie per assumere un ruolo importante nello sviluppo software. Il contributo di ChatGPT è stato fondamentale in alcune circostanze, ha rappresentato un utile ausilio in alcune fasi ma è stato meno affidabile in altre.

Ad oggi, dunque, non è possibile affermare che ChatGPT sia uno strumento capace di sostituire uno sviluppatore. Sebbene ChatGPT sia utile e in grado di fornire supporto in molte aree dello sviluppo software, esso manca della capacità critica e dell'interpretazione contestuale che consentono a un programmatore di prendere decisioni informate e strategiche. Queste capacità umane sono fondamentali per valutare le diverse opzioni disponibili e scegliere la soluzione più appropriata rispetto al problema e al contesto di sviluppo dell'applicativo. Ne deriva che ChatGPT costituisce un potente alleato, uno strumento complementare alle capacità del programmatore che però non può sostituire il giudizio e l'esperienza che solo un utente umano dispone.

Ringraziamenti

Desidero esprimere i miei più sinceri ringraziamenti alle persone che mi hanno sostenuto durante il mio percorso universitario.

Prima di tutto, vorrei ringraziare la Prof. Daniela Micucci, per la sua guida e il suo supporto durante il progetto di stage e di tesi.

Ringrazio inoltre la Dott. Maria Teresa Rossi per il suo contributo e i suoi feedback che hanno migliorato il contenuto di questa tesi.

Ringrazio la mia mamma, la mia migliore amica e mia più grande sostentrice, la mia fonte di ispirazione, che ha gioito per le mie vittorie e mi ha incoraggiato nei momenti difficili offrendo saggi consigli e sostenendomi sempre.

Ringrazio il mio papà, che mi ha sempre spinta a impegnarmi e ad affrontare le difficoltà, senza arrendermi mai.

Ringrazio Giuseppe, la persona più intraprendente che conosca, che mi ha insegnato a sognare in grande e ad essere ambiziosa, a non accontentarmi mai.

Ringrazio i colleghi Luca e Marco, con i quali ho affrontato l'intero percorso di stage, per avermi offerto supporto e un confronto critico e stimolante.

Infine, ringrazio Andrea, il ragazzo più importante della mia vita, per essermi sempre stato accanto, per avermi ascoltato e supportato, offrendomi il suo sostegno ogni qualvolta ne avessi bisogno.

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).