



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:
Marco Ferioli
Matricola 879277

Anno Accademico 2023-2024

Indice

Introduzione	4
1 Analisi dei requisiti	5
1.1 Come nasce Communimib	5
1.2 Analisi dei casi d'uso	7
1.2.1 Diagramma dei casi d'uso	7
1.2.2 Attori	8
1.2.3 Descrizione dei casi d'uso in formato dettagliato	9
1.3 Organizzazione dei requisiti	20
1.4 Modello di dominio	23
2 Descrizione dell'implementazione	24
2.1 Analisi architetturale	24
2.1.1 Clean Architecture	25
2.1.2 Model-View-ViewModel (MVVM)	26
2.1.3 Diagramma dei package	28
2.2 Implementazione	29
2.2.1 Autenticazione	29
2.2.2 Bacheca	35
2.2.3 Segnalazioni	41
2.2.4 Profilo	48
2.2.5 Dettagli relativi all'uso di Firebase	51
3 Progetto di testing	53
3.1 Il ruolo dei test nello sviluppo Android	53
3.2 Test di unità	53
3.3 Test UI	54
4 Utilizzo dell'IA	56
4.1 ChatGPT v3.5	56
4.1.1 Ideazione ed analisi dei requisiti	57
4.1.2 Implementazione	58
4.1.3 Testing	61
4.1.4 Problematiche incontrate	61
4.1.5 Metodo di verifica ed analisi dei dati raccolti	62
4.2 Design del logo Communimib	64
Conclusioni	65
Ringraziamenti	66
Sitografia	67

Elenco delle figure

1.1	Diagramma dei casi d'uso	7
1.2	Diagramma del modello di dominio	23
2.1	Clean Architecture	25
2.2	Architettura a tre layer	26
2.3	Diagramma dell'architettura	27
2.4	Diagramma dei package	28
2.5	Classe Validation	29
2.6	Login	30
2.7	Registrazione	30
2.8	Conferma mail	31
2.9	Reset password	31
2.10	Diagramma classi autenticazione - UI layer	33
2.11	Diagramma classi autenticazione - Data layer	34
2.12	Bacheca	36
2.13	Avviso nuovo post	36
2.14	Creazione post	37
2.15	Commenti	37
2.16	Diagramma classi bacheca - UI layer	39
2.17	Diagramma classi bacheca - Data layer	40
2.18	Segnalazioni	42
2.19	UI dipendente	42
2.20	Selezione preferiti	44
2.21	Selezione filtri	44
2.22	Segnalazione dettagliata	45
2.23	Creazione segnalazione	45
2.24	Diagramma classi segnalazioni - UI layer	46
2.25	Diagramma classi segnalazioni - Data layer	47
2.26	Profilo	48
2.27	Modifica profilo	48
2.28	Diagramma classi profilo - UI layer	50
4.1	Splash Screen con il logo dell'app	59
4.2	Utilità di ChatGPT	62
4.3	Utilizzo di ChatGPT	62
4.4	Chat di successo e fallimento nei metodi di utilizzo di ChatGPT	63
4.5	Chat di successo e fallimento nei casi delle problematiche incontrate	63
4.6	Generazione e modifiche applicate al logo di Communimib	64

Elenco delle tabelle

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

Introduzione

La grande diffusione dell'IA (e dei modelli generativi) è un'importante argomento di discussione, perchè solleva rilevanti questioni etiche, economiche e sociali riguardanti l'automazione, la privacy dei dati ed il futuro del lavoro.

La seguente trattazione descrive dettagliatamente il processo che ha portato allo sviluppo dell'applicativo "Communimib", con lo scopo di valutare quanto le intelligenze artificiali basate sul testo (in particolare gli LLM) possano essere di supporto al programmatore durante le fasi che compongono il processo di creazione di un software.

Communimib è un'applicazione per dispositivi mobili Android, sviluppata utilizzando il linguaggio di programmazione Java, che si colloca nel contesto universitario dell'ateneo. Per rispondere alle esigenze degli studenti e del personale universitario, tale applicazione è stata strutturata in quattro sezioni principali: un servizio di autenticazione centralizzato (registrazione e login), un sistema di segnalazioni, un "social network" che simula il funzionamento delle bacheche universitarie ed un meccanismo di personalizzazione e gestione del profilo personale.

Durante le varie fasi che compongono il processo di sviluppo software, il team di sviluppo è stato affiancato e supportato principalmente da ChatGPT v3.5, che è stato consultato per rispondere a diverse operazioni; ad esempio risolvere dubbi implementativi, generare algoritmi ed identificare componenti. Ciò ha permesso di realizzare, in fase implementativa, un sistema basato sulla Modern App Architecture, ossia un'architettura scalabile e portata all'inclusione di nuove funzionalità. Inoltre il sistema di intelligenza artificiale è stato applicato anche durante la scrittura di un insieme di test automatici per validare il funzionamento dell'app. Al contrario, il sistema di IA ha trovato difficile impiego nelle sezioni di ideazione ed analisi dei requisiti dell'applicazione.

Al fine di valutare l'utilità dello strumento preso in considerazione, è stato compilato un "diario" durante lo sviluppo dell'applicazione, contenente tutte le chat effettuate con il sistema di intelligenza artificiale, le quali sono state catalogate sulla base di parametri come: fase del ciclo di vita, utilità, rielaborazione dell'input e dell'output. Ciò ha permesso, al termine dell'esperimento, di valutare nel suo complesso l'esperienza svolta, fornendo inoltre differenti considerazioni e spunti relativi all'utilizzo del sistema.

Capitolo 1

Analisi dei requisiti

1.1 Come nasce Communimib

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

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

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

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

La seconda funzionalità si configura in una piattaforma in cui gli utenti possono pubblicare post atti a raggiungere, in base alle proprie esigenze, gli altri membri della comunità universitaria. Tale piattaforma consente di ammodernare l'anacronistico uso delle bacheche fisiche presenti negli edifici, i cui annunci spesso non raggiungono i destinatari sperati; 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, si può pensare ad un semplice esempio: ogni studente, in base alla facoltà a cui è iscritto e alle lezioni che decide di seguire, tende a frequentare spesso lo stesso insieme (solitamente ristretto) di edifici, pertanto sarà probabilmente interessato a conoscere soltanto le segnalazioni attive in tali edifici. Contestualmente, è probabile che non tutti gli utenti facciano uso della bacheca allo stesso modo; a titolo esemplificativo, si pensi agli studenti che intendono prendere lezioni private per un superare un esame difficile, a coloro che sono alla ricerca di una casa in affitto, a chi ha semplicemente smarrito la giacca o chi invece vuole promuovere un'iniziativa benefica.

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

1.2 Analisi dei casi d'uso

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

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

1.2.1 Diagramma dei casi d'uso

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

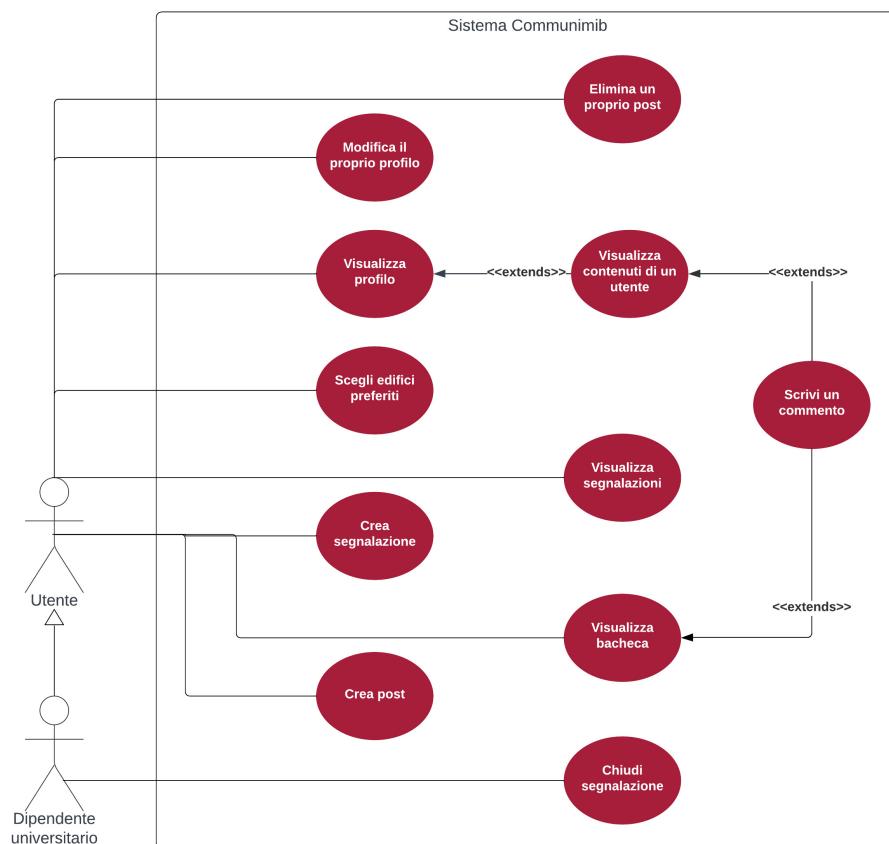


Figura 1.1: Diagramma dei casi d'uso

1.2.2 Attori

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

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

Tabella 1.1: Attore - Utente

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

Tabella 1.2: Attore - Dipendente universitario

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

1.2.3 Descrizione dei casi d'uso in formato dettagliato

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

Tabella 1.3: Caso d'uso - scegli edifici preferiti

Nome del caso d'uso	Scegli edifici preferiti
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate ed interessi	L'utente è interessato a selezionare un insieme di edifici preferiti tra quelli appartenenti all'ateneo.
Pre-condizioni	L'utente deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura gli edifici scelti dall'utente vengono registrati nel sistema ed associati al profilo personale.
Scenario principale di successo	<p>1. L'utente visualizza l'elenco degli edifici dell'ateneo.</p> <p>2. L'utente seleziona l'insieme degli edifici a cui è interessato.</p> <p>3. Il sistema memorizza gli edifici selezionati.</p>
Estensioni	Scenario alternativo di fallimento: <p>3.a. Se il sistema fallisce nel salvataggio degli edifici selezionati, l'utente visualizza il relativo messaggio d'errore.</p>
Requisiti speciali	Nessuno
Frequenza di ripetizione	Bassa

Tabella 1.4: Caso d'uso - visualizza segnalazioni

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

Tabella 1.5: Caso d'uso - crea segnalazione

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

Tabella 1.6: Caso d'uso - chiudi segnalazione

Nome del caso d'uso	Chiudi segnalazione
Portata	Sistema Communimib
Livello	Obiettivo utente
Attore primario	Dipendente universitario
Parti interessate ed interessi	Il dipendente universitario è interessato a chiudere una segnalazione.
Pre-condizioni	Il dipendente universitario deve essere registrato ed autenticato nel sistema.
Garanzia di successo	Al completamento della procedura, il dipendente universitario ha correttamente chiuso la segnalazione desiderata.
Scenario principale di successo	<p>1. Il dipendente universitario sceglie la segnalazione che desidera chiudere.</p> <p>2. Il sistema avvia la chiusura della segnalazione.</p>
Estensioni	<p>Scenario alternativo di fallimento:</p> <p>2.a Se il sistema fallisce durante la chiusura della segnalazione, viene mostrato al dipendente universitario il relativo messaggio di errore.</p>
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>Scenario alternativo di successo:</p> <ol style="list-style-type: none"> 1. L'utente accede alla sezione dedicata alla visualizzazione della bacheca. 2. Il sistema mostra all'utente i post pubblicati nella bacheca.
Estensioni	<p>Scenario alternativo di fallimento:</p> <ol style="list-style-type: none"> 2.a Se il sistema fallisce durante la visualizzazione dei post, viene mostrato il relativo messaggio di errore.
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	<p>1. L'utente accede alla sezione che consente la pubblicazione di un nuovo post.</p> <p>2. L'utente inserisce i dati relativi al post che desidera pubblicare.</p> <p>3. Il sistema pubblica il post.</p>
Estensioni	<p>Scenari alternativi di fallimento:</p> <p>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.</p> <p>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.</p> <p>3.a Se il sistema fallisce nella creazione del post, viene mostrato all'utente il relativo messaggio di errore.</p>
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	<p>1. L'utente accede alla sezione per la visualizzazione del proprio profilo.</p> <p>2. Il sistema mostra i dati associati al proprio profilo.</p>
Estensioni	<p>Scenario alternativo di successo:</p> <p>1. L'utente richiede la visualizzazione del profilo di un altro utente.</p> <p>2. Il sistema mostra i dati associati al profilo desiderato.</p> <p>Scenario alternativo di fallimento:</p> <p>2.a Se il sistema fallisce nella visualizzazione dei dati, viene mostrato il relativo messaggio di errore.</p>
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	<p>1. L'utente accede alla sezione di modifica del profilo.</p> <p>2. L'utente sceglie di modificare i dati associati al proprio profilo.</p> <p>3. Il sistema salva i dati inseriti e mostra un messaggio di conferma.</p>
Estensioni	Scenario alternativo di fallimento: <p>3.a. Se il sistema fallisce nel salvataggio dei dati, viene mostrato il relativo messaggio di errore.</p>
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	Scenario alternativo di successo: <ol style="list-style-type: none"> 4.a. L'utente richiede l'annullamento dell'eliminazione del post. 4.b. Il sistema ripristina il post. Scenario alternativo di fallimento: <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	<p>1. L'utente accede alla sezione dedicata alla visualizzazione di un profilo (proprio o altrui).</p> <p>2. L'utente accede alla sezione per la visualizzazione dei post.</p> <p>3. il sistema mostra all'utente i post pubblicati.</p>
Estensioni	<p>Scenario alternativo di successo:</p> <p>1. L'utente accede alla sezione dedicata alla visualizzazione di un profilo (proprio o altrui).</p> <p>2. L'utente accede alla sezione dedicata alla visualizzazione delle segnalazioni.</p> <p>3. il sistema mostra all'utente le segnalazioni pubblicate.</p> <p>Scenario alternativo di fallimento:</p> <p>3.a. Se il sistema fallisce durante la visualizzazione dei post o delle segnalazioni, viene mostrato il relativo messaggio di errore.</p>
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	<p>1. L'utente accede alla sezione adibita alla visualizzazione della bacheca.</p> <p>2. L'utente seleziona il post che vuole commentare.</p> <p>3. L'utente scrive il commento e lo pubblica.</p> <p>4. Il sistema memorizza il commento e lo mostra nell'elenco dei commenti presenti.</p>
Estensioni	<p>Scenario alternativo di successo:</p> <p>1. L'utente accede alla sezione dedicata alla visualizzazione del profilo (proprio o altrui).</p> <p>2. L'utente accede alla sezione dedicata alla visualizzazione dei contenuti del profilo.</p> <p>3. L'utente clicca sul post che vuole commentare.</p> <p>4. L'utente scrive il commento e lo pubblica.</p> <p>5. Il sistema memorizza il commento e lo mostra nell'elenco dei commenti presenti.</p> <p>Scenario alternativo di fallimento:</p> <p>4.a. Se il sistema fallisce durante la memorizzazione del commento, viene mostrato il relativo messaggio di errore.</p>
Requisiti speciali	Nessuno
Frequenza di ripetizione	Bassa

1.3 Organizzazione dei requisiti

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

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

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

- **M**, "Must": descrive un requisito che deve essere soddisfatto nella soluzione finale, affinché essa sia considerata un successo.
- **S**, "Should": rappresenta un aspetto 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 [5].

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 con le relative relazioni.

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

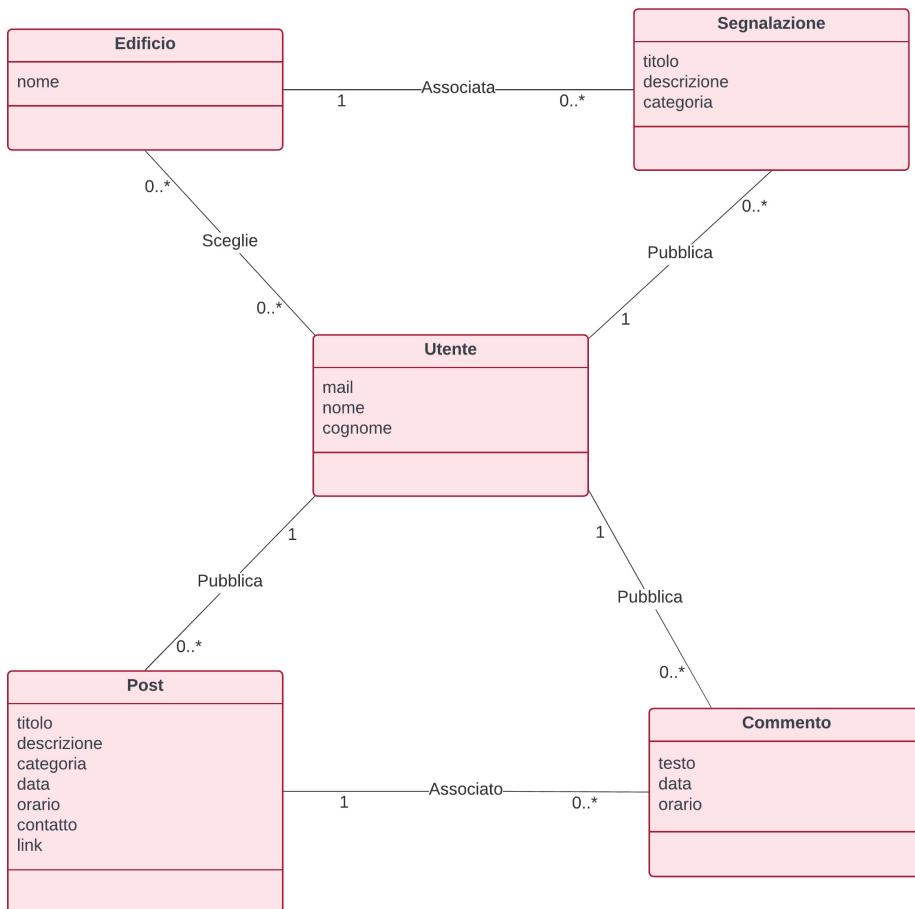


Figura 1.2: Diagramma del modello di dominio

Capitolo 2

Descrizione dell'implementazione

2.1 Analisi architetturale

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

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

- **Flessibilità.** Si tratta di una caratteristica estremamente importante, poichè 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 permette di gestire e risolvere tali problematiche senza generarne di nuove.
- **Scalabilità.** L’applicazione è estendibile, di conseguenza si presta facilmente all’aggiunta di nuovi contenuti. In generale un sistema non dovrebbe mai essere statico, chiuso e quindi limitato alle funzionalità implementate inizialmente. Questa caratteristica garantisce anche durabilità del software, poichè può essere modificato ed esteso nel tempo a seconda delle funzionalità richieste.
- **Riutilizzo.** Quando il contesto lo permette, è possibile riutilizzare il codice scritto, senza duplicarlo. Questo non solo riduce il tempo e gli sforzi necessari per lo sviluppo di nuove funzionalità, ma assicura anche una maggiore coerenza e qualità del codice, in quanto il riutilizzo di componenti già esistenti implica che questi siano già testati e ottimizzati.

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

2.1.1 Clean Architecture

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

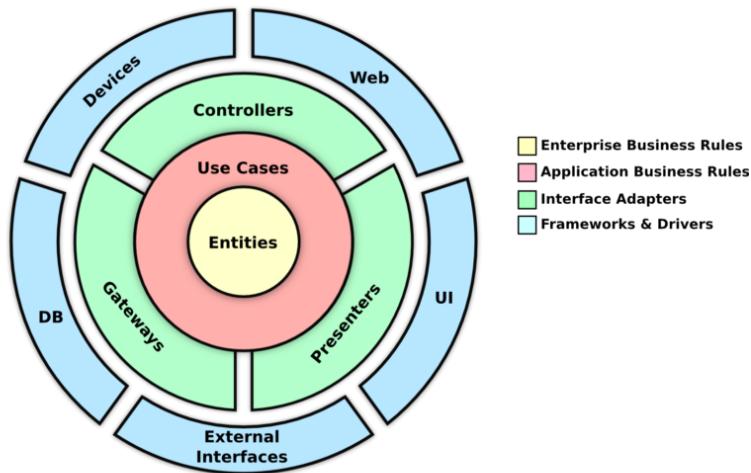


Figura 2.1: Clean Architecture

La regola fondamentale su cui si basa la Clean Architecture è la **Dependency Rule**, secondo la quale la direzione delle dipendenze deve 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 [1].

Tuttavia, la Clean Architecture tende 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].

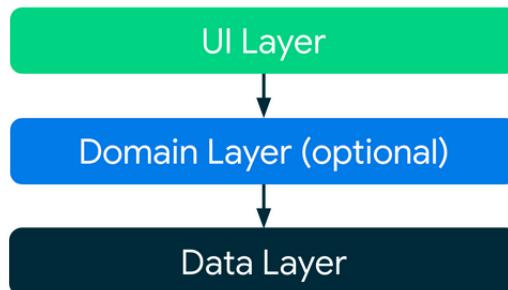


Figura 2.2: Architettura a tre layer

2.1.2 Model-View-ViewModel (MVVM)

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

Questo suggerisce la costituzione di tre componenti:

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

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

implementato il relativo ViewModel, che fornisce i dati da mostrare all'utente. Ogni ViewModel reperisce i dati dal Model sottostante, costituito dalle classi Repository e dalle classi Datasource. 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 diagramma seguente.

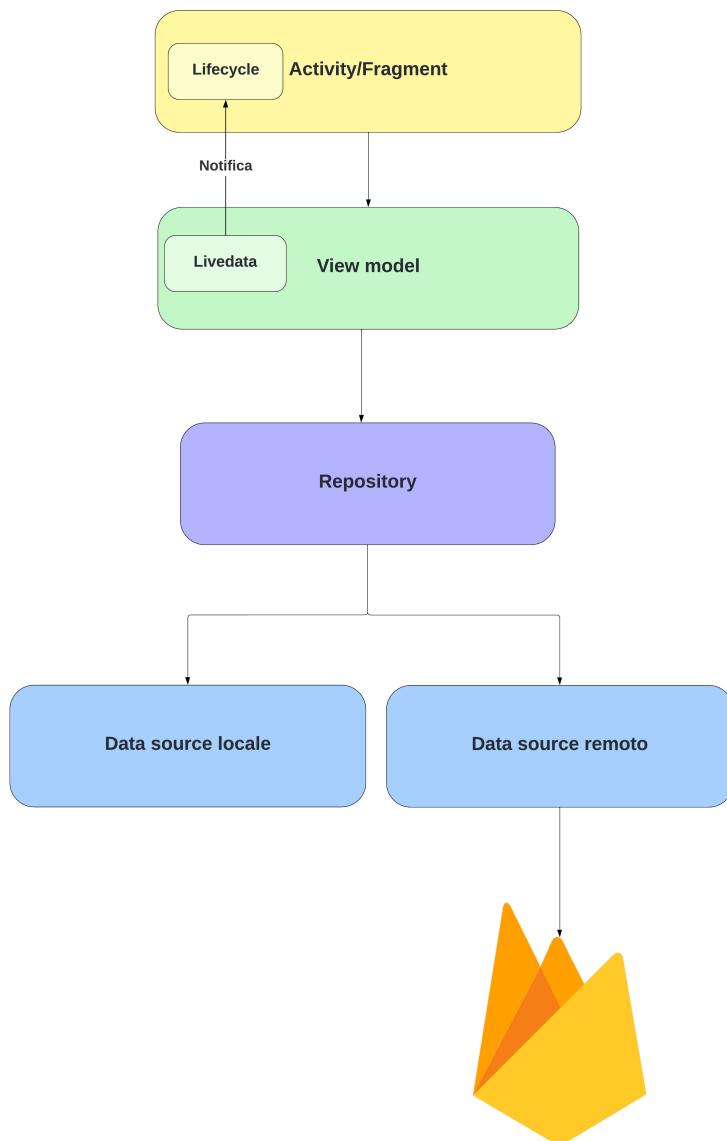


Figura 2.3: Diagramma dell'architettura

2.1.3 Diagramma dei package

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

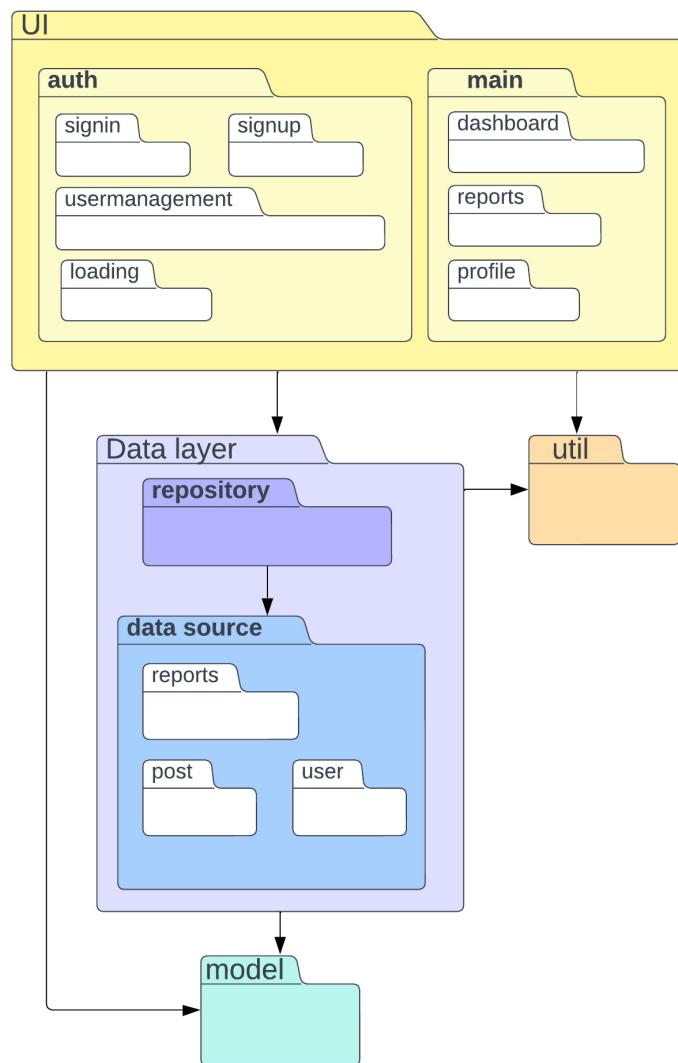


Figura 2.4: Diagramma dei package

2.2 Implementazione

Dopo aver fissato i requisiti e definito l'architettura, segue la fase di implementazione del software. In questa sezione verranno descritti i dettagli tecnici relativi alla realizzazione dell'applicazione mobile. 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

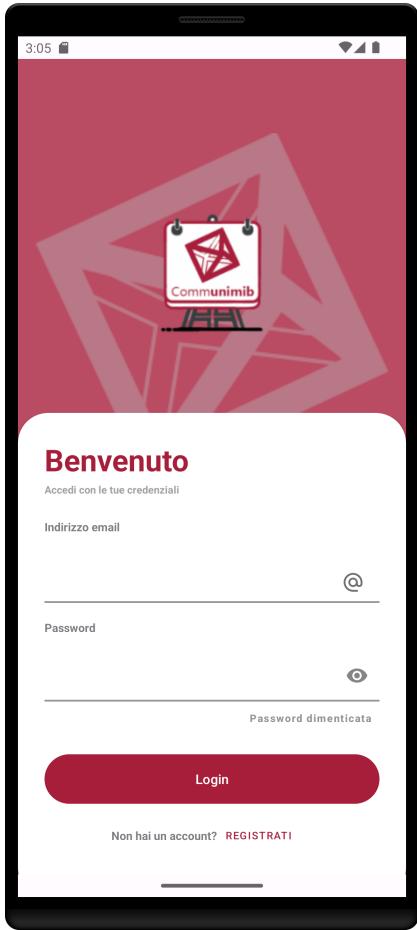


Figura 2.6: Login



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



Figura 2.8: Conferma mail

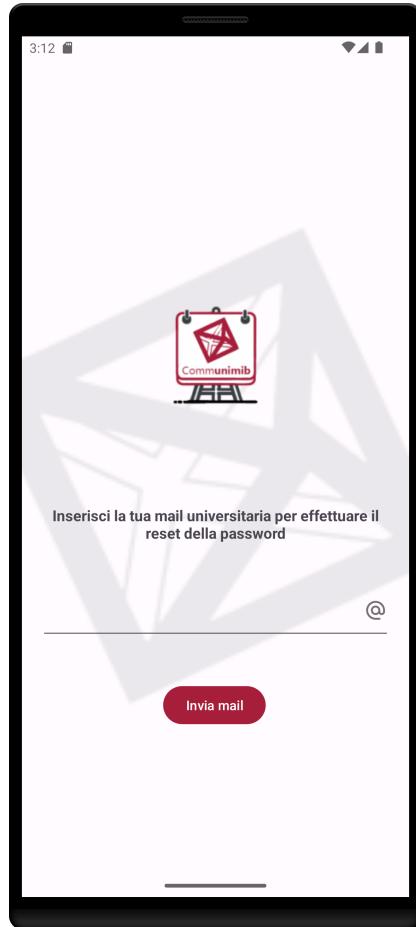


Figura 2.9: Reset password

Verifica e-mail

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

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

La procedura di verifica si compone di alcuni semplici passaggi:

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

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

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

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

Gestione della sessione

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. Dato che in alcuni casi l'esecuzione di tale meccanismo può richiedere molto tempo (ad esempio, se si utilizza una connessione lenta), è stato definito ed implementato uno splash screen che viene visualizzato all'avvio dell'applicazione e mostra un'interfaccia di caricamento fino al completamento delle operazioni.

Reset Password

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

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

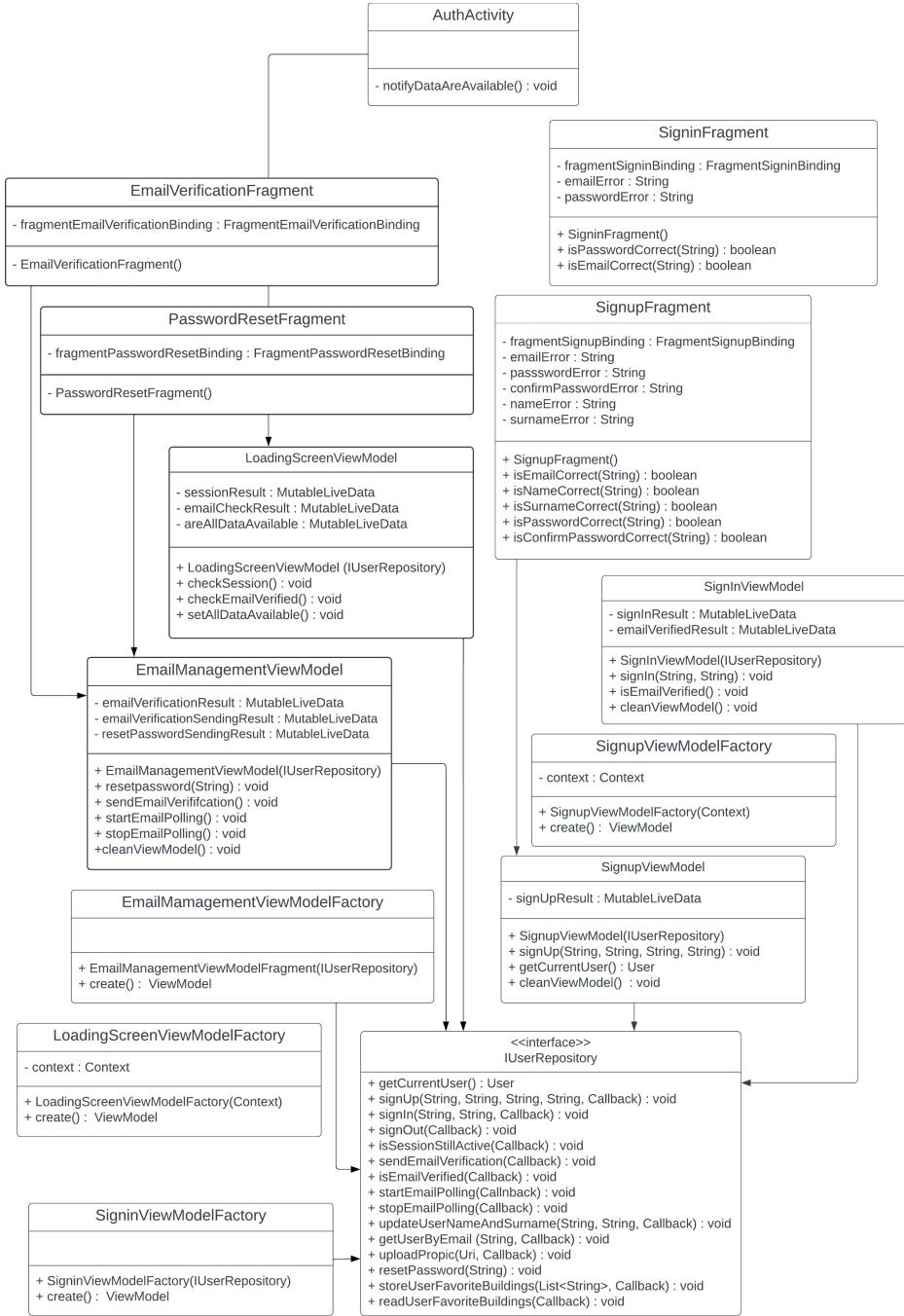


Figura 2.10: Diagramma classi autenticazione - UI layer

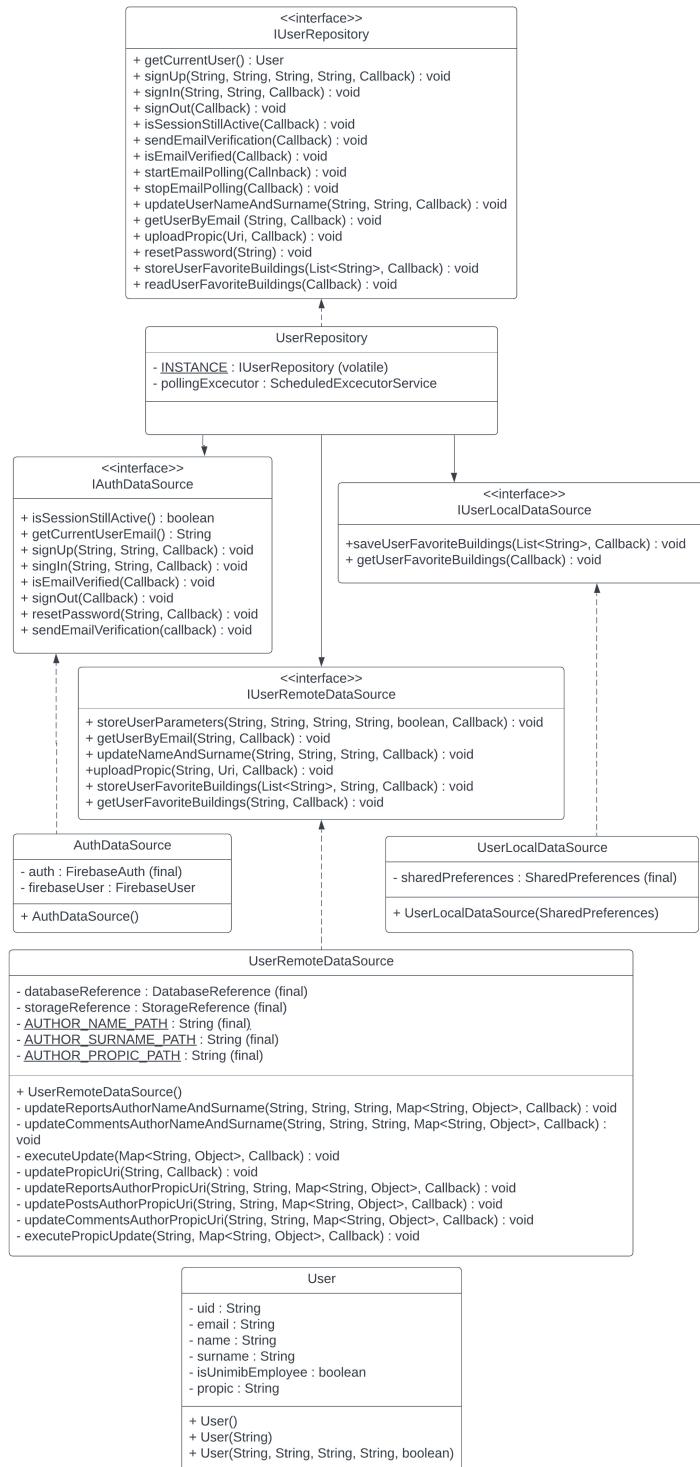


Figura 2.11: Diagramma classi autenticazione - Data layer

2.2.2 Bacheca

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

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

Visualizzazione dei post nella bacheca

La visualizzazione della bacheca e dei suoi post ha rappresentato un elemento chiave nello sviluppo di 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; quindi 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 per fare in modo che sia visibile e consultabile anche dagli altri utenti. Per massimizzare 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 solamente 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 un semplice blocco 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 stato 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` [3] 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 nel Firebase Realtime Database. Dal momento che quest'ultimo non supporta la memorizzazione delle immagini, viene quindi utilizzato Firebase Storage per effettuare tale operazione.



Figura 2.14: Creazione post

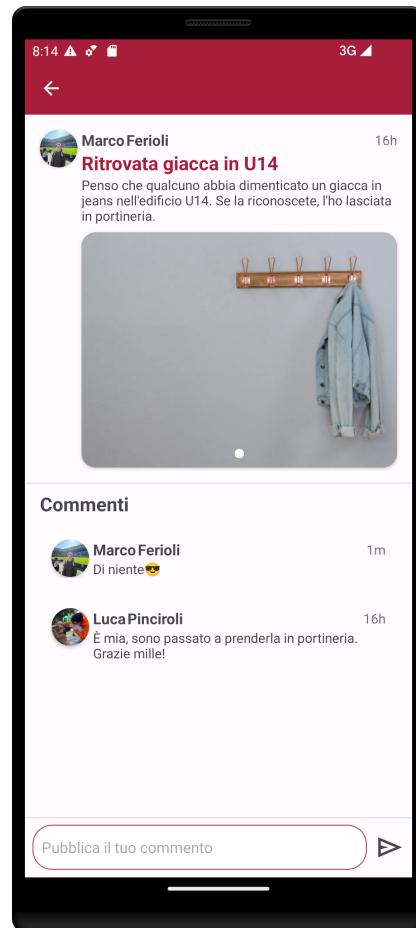


Figura 2.15: Commenti

Commenti

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

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

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

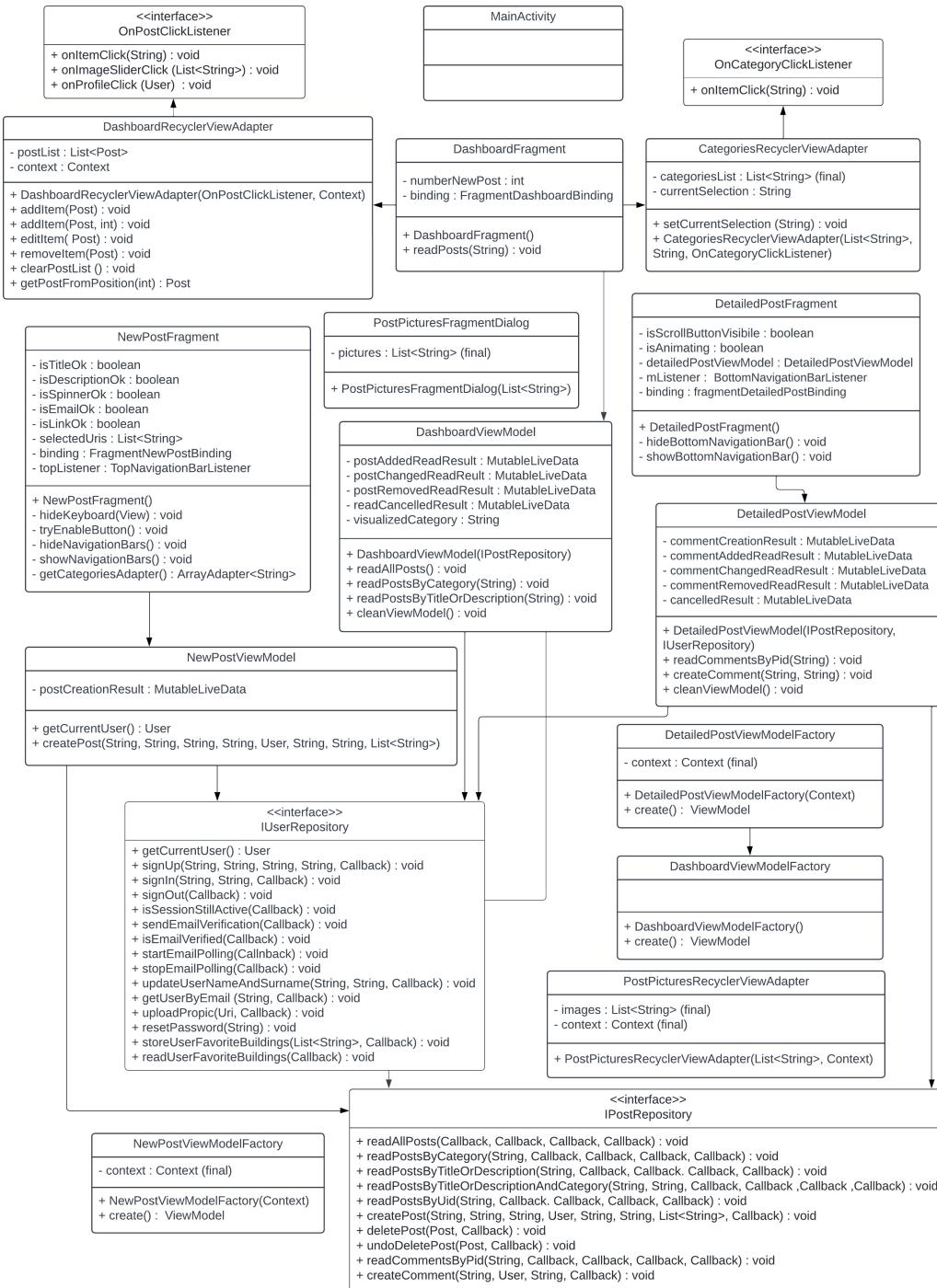


Figura 2.16: Diagramma classi bacheca - UI layer

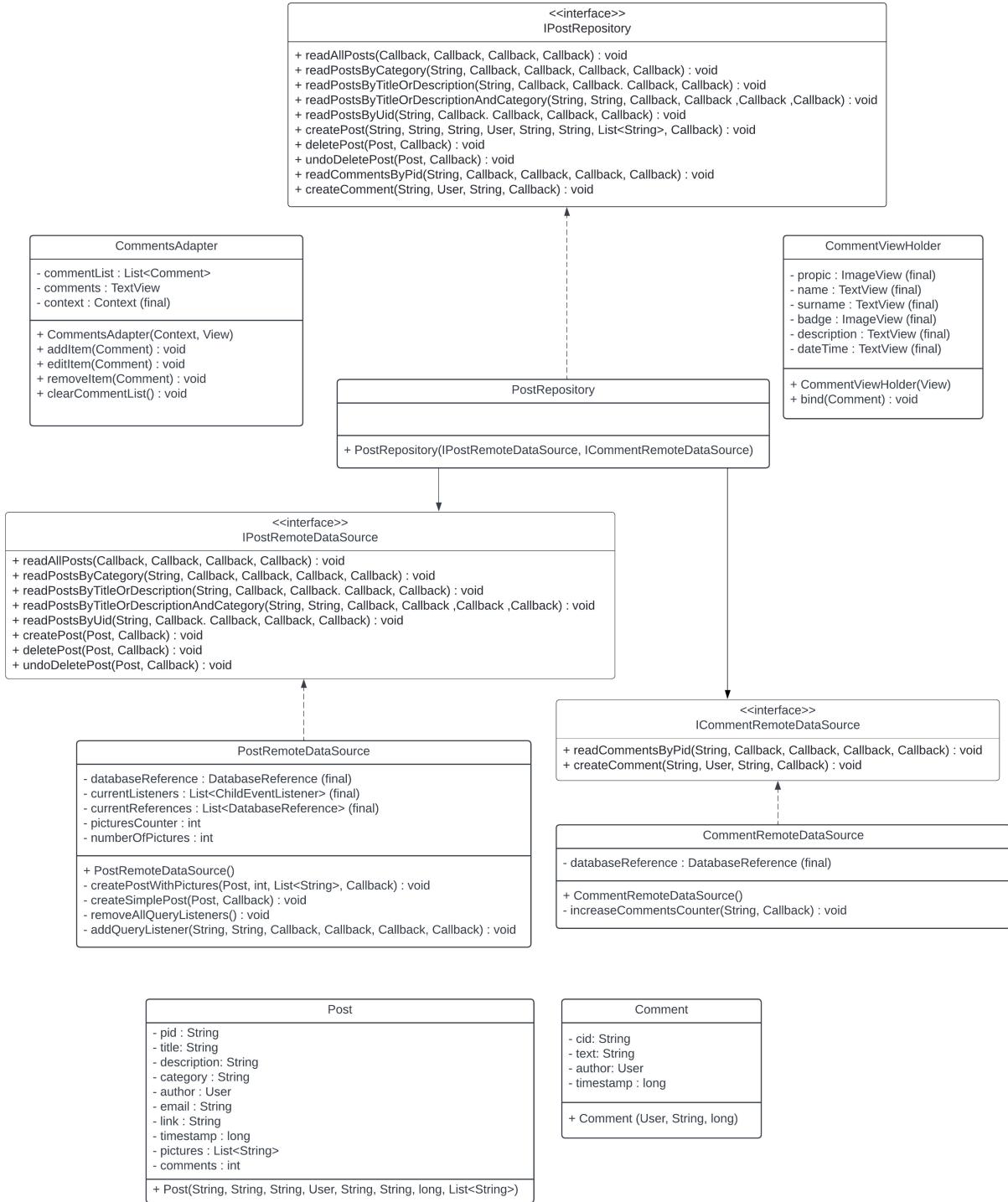


Figura 2.17: Diagramma classi bacheca - Data layer

2.2.3 Segnalazioni

Il sistema di segnalazione di Communimib nasce per rafforzare la collaborazione tra corpo studentesco e personale universitario. Nel rispetto del ruolo di ciascuna controparte, l'utilizzo di tale strumento può portare 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.

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

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

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

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

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

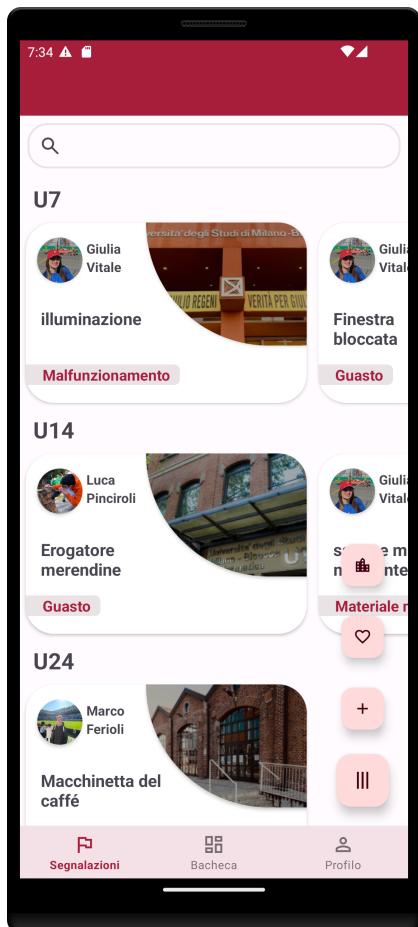


Figura 2.18: Segnalazioni

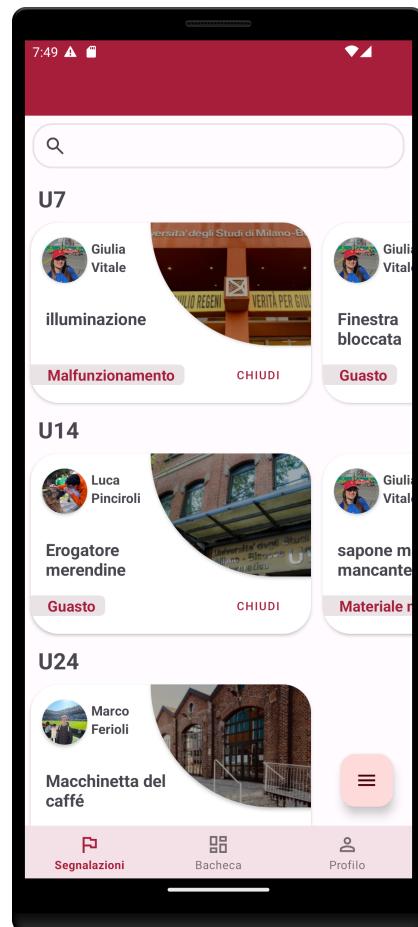


Figura 2.19: UI dipendente

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` (chiamati anche comunemente "dialog"). 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 `List` ed inviati al data layer. Qui viene effettuata una query verso Firebase Realtime Database per ottenere solo ed esclusivamente le segnalazioni associate agli edifici selezionati. Successivamente, queste segnalazioni vengono posizionate adeguatamente nella schermata, sfruttando 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

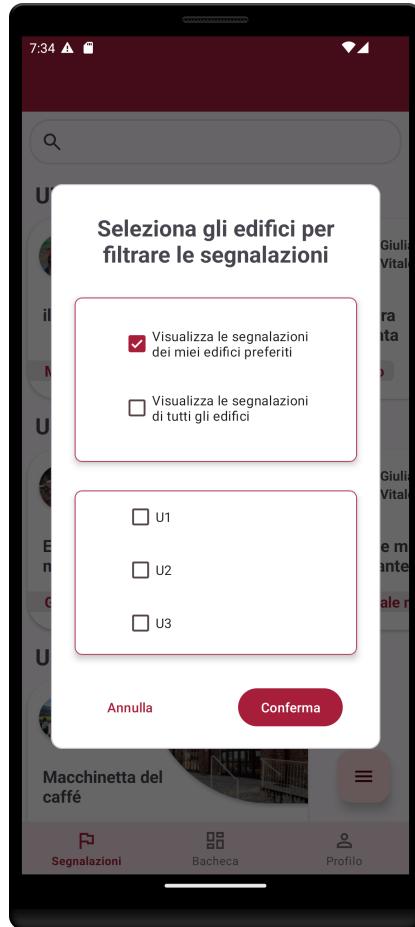


Figura 2.21: Selezione filtri

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, di fatto attuabile soltanto dai membri del personale di ateneo.

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

Dal momento che una delle due operazioni chiave è riservata al solo personale universitario, l'interfaccia grafica assume un comportamento differente in base alla categoria di utente che fa uso dell'applicazione; infatti 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

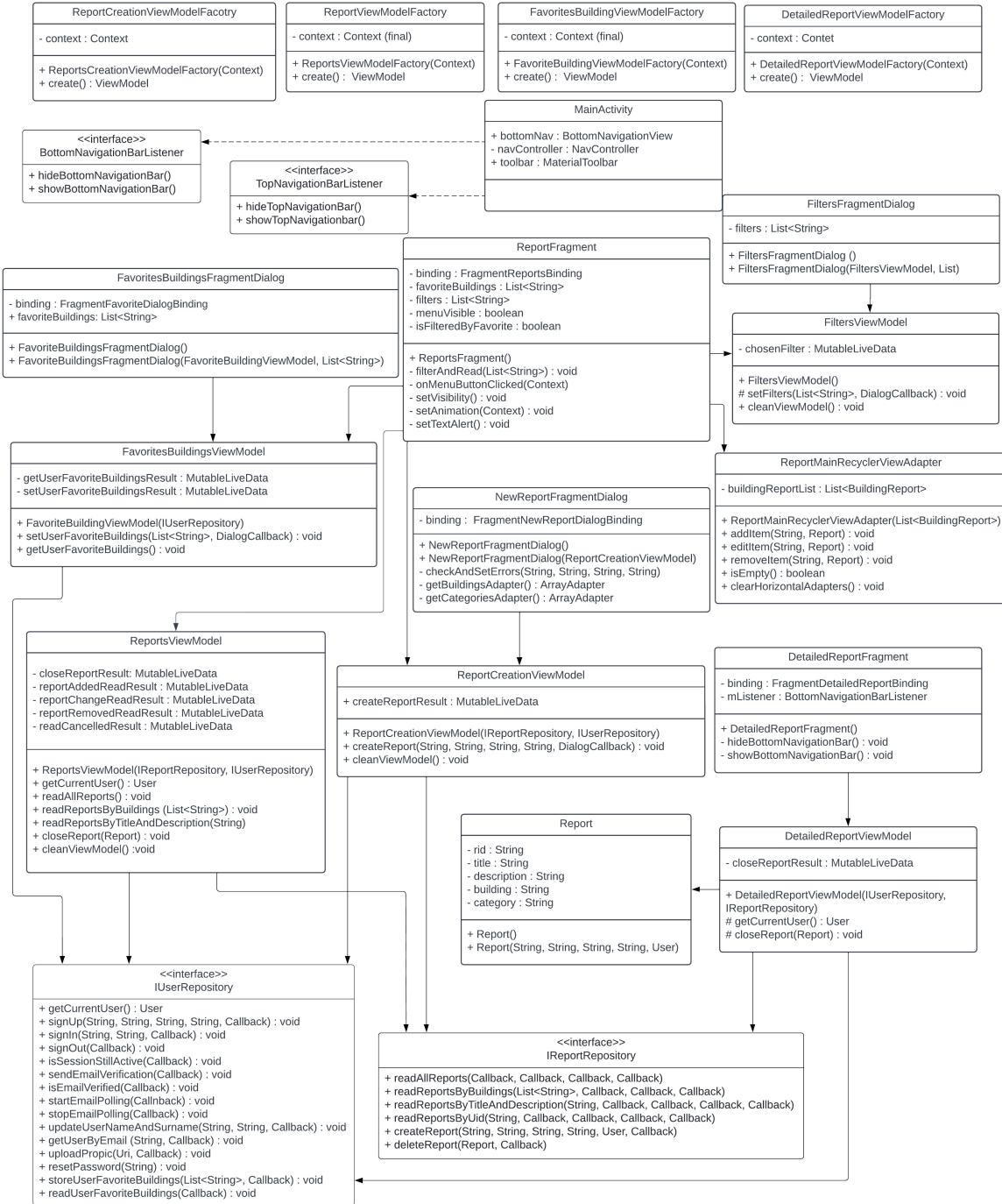


Figura 2.24: Diagramma classi segnalazioni - UI layer

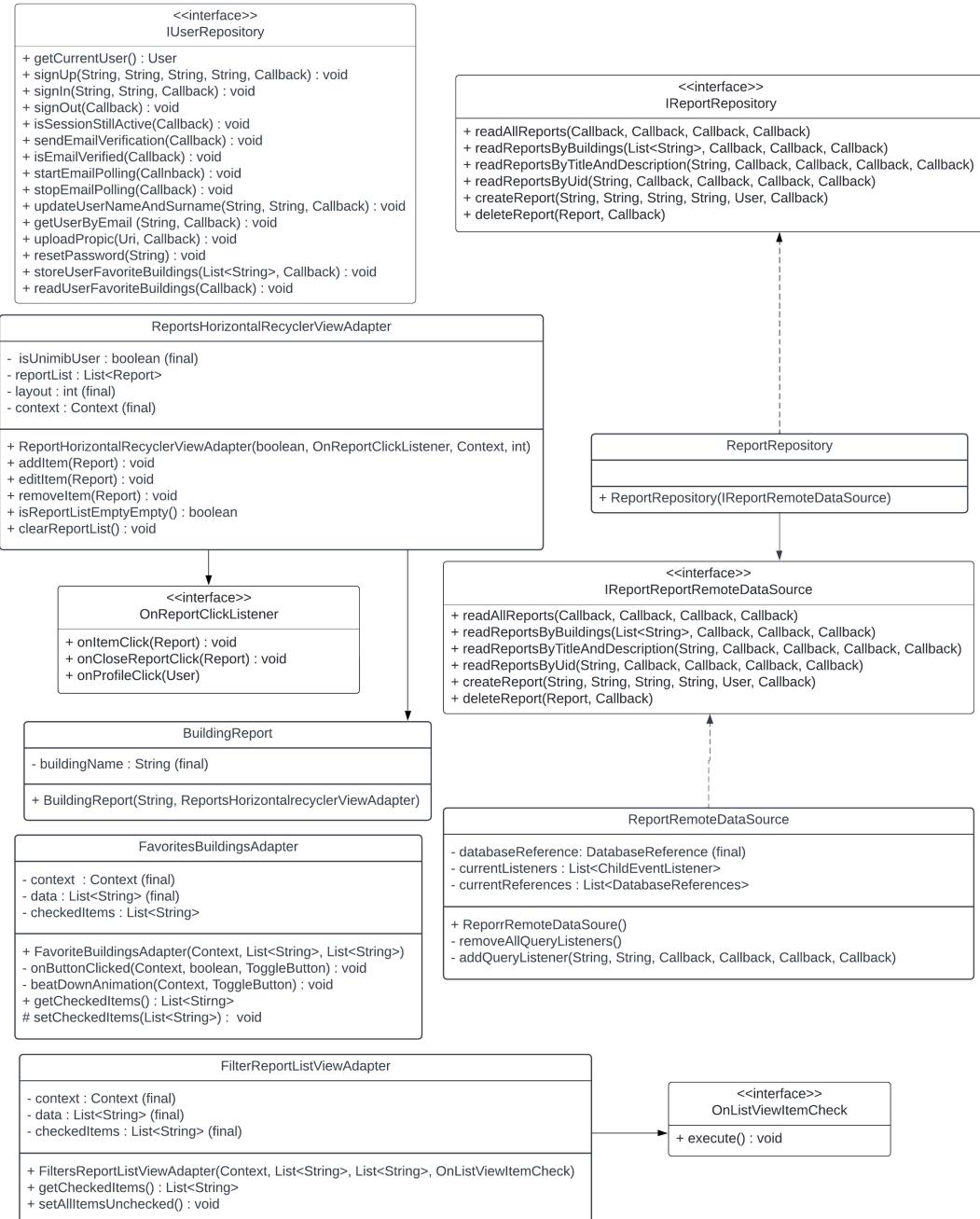


Figura 2.25: Diagramma classi segnalazioni - Data layer

2.2.4 Profilo

L'ultima sezione dell'applicazione è rappresentata dal profilo. Questa offre all'utente la possibilità di gestire il proprio account (si veda Figura 2.26), per apportare modifiche alle informazioni personali e visualizzare le segnalazioni o i post che ha pubblicato. La medesima schermata, a meno di alcuni particolari, viene utilizzata anche per la visualizzazione dei profili degli altri utenti. 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 ed alle segnalazioni.

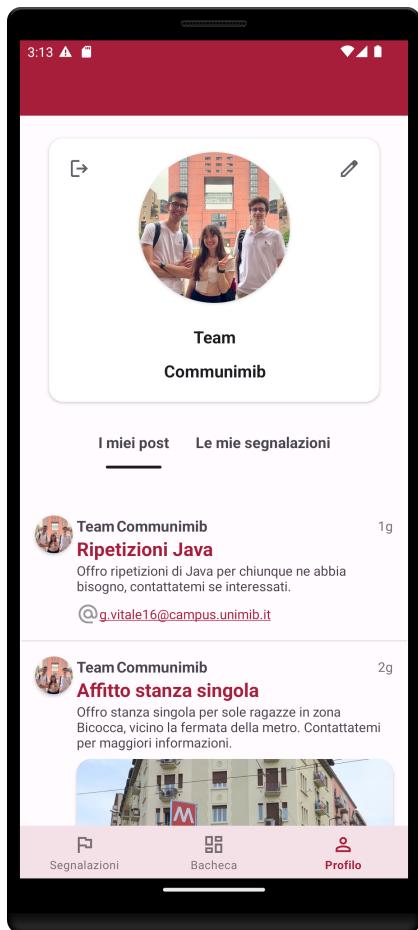


Figura 2.26: Profilo

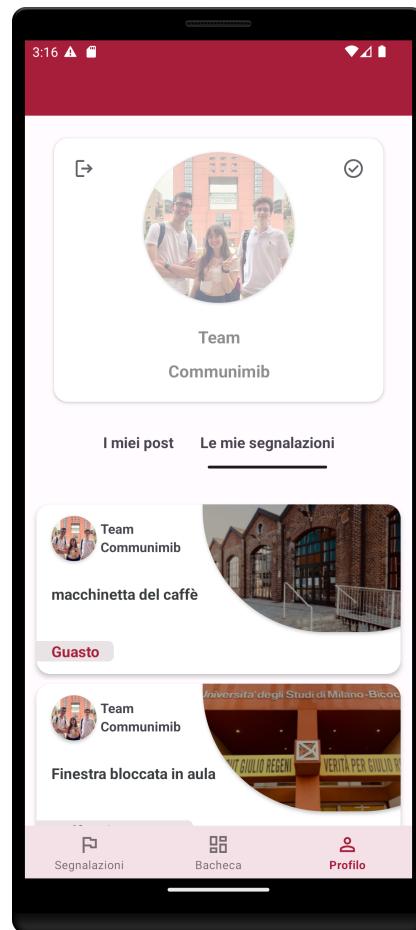


Figura 2.27: Modifica profilo

Immagine e parametri utente

Con lo scopo di fornire una maggiore libertà di personalizzazione, l'utente ha la possibilità di modificare arbitrariamente alcuni parametri che sono associati al profilo personale: nome, cognome ed immagine profilo (si veda Figura 2.27). Per ragioni estetiche, il team ha deciso di adottare una forma circolare per le immagini di profilo, rendendo 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 alla risoluzione del problema corrispondente, i post possono rimanere permanentemente memorizzati all'interno del sistema; questo comportamento potrebbe non essere sempre gradito all'utente, il quale potrebbe manifestare l'esigenza di rimuovere uno o più post dal suo profilo.

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

Un meccanismo 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.

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

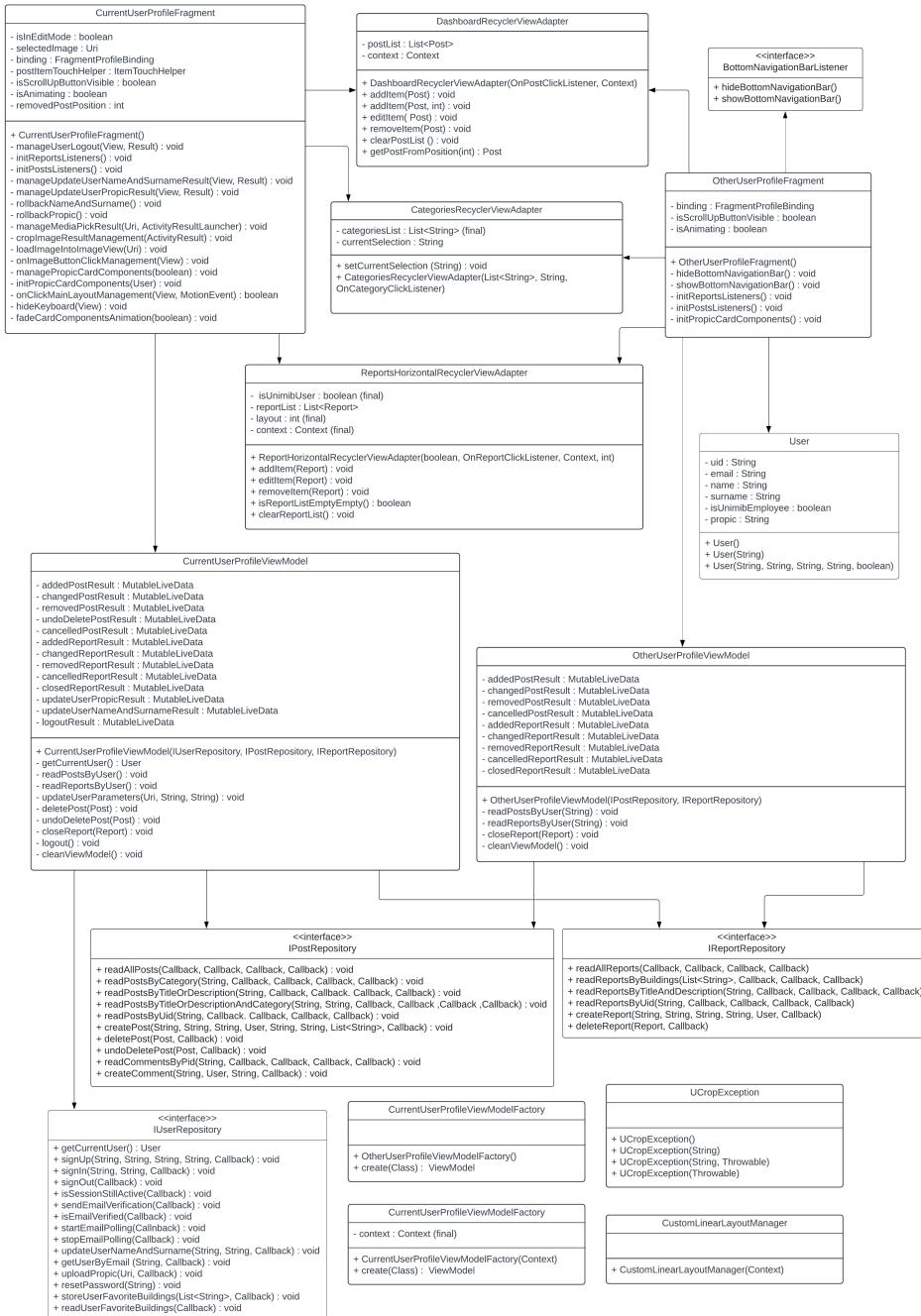


Figura 2.28: Diagramma classi profilo - UI layer

2.2.5 Dettagli relativi all'uso di Firebase

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

Firebase [10] è un servizio cloud offerto da Google che mette a disposizione un'ampia suite di strumenti a supporto degli sviluppatori mobile: comprende un sistema di gestione dell'autenticazione server-side, strumenti di analisi sul traffico dell'applicazione, uno spazio di archiviazione remoto, un servizio di AI generativa, ecc.

I servizi su cui Communimib si basa sono:

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

Firebase Authentication

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

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

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

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

Firebase Realtime Database

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

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

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

Il database messo a disposizione da Firebase Realtime Database è di tipo non relazionale e prevede la memorizzazione dei dati in formato JSON, richiedendo così un modus operandi ben diverso da quello richiesto dai classici database relazionali. Per questo motivo, 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; infatti se la stessa informazione è duplicata in punti diversi del database, si manifesta il rischio concreto di generare punti di inconsistenza che possono portare a malfunzionamenti nell'applicazione.

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

Firestore Storage

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

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

Capitolo 3

Progetto di testing

3.1 Il ruolo dei test nello sviluppo Android

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

Sebbene i test manuali effettuati dagli sviluppatori 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 [2]:

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

Infine, poichè 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 varie fasi implementative.

3.2 Test di unità

I test di unità isolano una singola porzione di codice testandola indipendentemente dal resto del sistema al fine di verificare che essa produca l'output previsto a partire da un dato input. Per l'implementazione di tali test è stato utilizzato

il framework Mockito [8], 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` così che 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 permette definire il comportamento della classe Mock, precisando i valori che devono essere assegnati alle callback utilizzate.
 2. **Chiamata al metodo da testare:** dopo aver definito i valori restituiti dalla classe mock, viene effettuata la chiamata al metodo interessato e si testa che i valori reali siano uguali ai valori specificati.
 3. **Invocazione metodo verify():** infine, si testa che il metodo esegua la chiamata alla classe Mock.

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

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

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 [7] (in combinazione con JUnit4), messa a disposizione da Android, per validare il funzionamento

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

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

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

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

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

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

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

Capitolo 4

Utilizzo dell'IA

4.1 ChatGPT v3.5

Il principale obiettivo prefissato all'inizio dell'attività di stage, che ha trovato riscontro nel progetto descritto nei capitoli precedenti, era valutare quanto un'intelligenza artificiale basata sul testo potesse essere utilizzata come supporto durante le varie fasi di sviluppo di un software.

Per condurre questo esperimento, sono stati presi in considerazione ed utilizzati principalmente i Large Language Models (LLM), ossia dei modelli di intelligenza artificiale la cui principale caratteristica è la capacità di comprendere e generare un testo in linguaggio naturale. Il principale strumento, che si basa su questa moderna tecnologia, consultato durante la creazione dell'applicativo, è ChatGPT nella versione 3.5. Tale scelta è stata effettuata sulla base di alcune considerazioni:

- **Capacità di comprensione e generazione del codice.** ChatGPT è in grado di comprendere un codice fornito in input e di generarlo in output; questa caratteristica è estremamente rilevante. Infatti, durante le varie fasi che compongono un processo di sviluppo software, le sezioni che occupano più tempo sono sicuramente quelle relative all'implementazione e testing dell'app. Di conseguenza, dal punto di vista programmatico, disporre di uno strumento che permetta di risolvere problemi o dubbi implementativi può rappresentare un notevole vantaggio in termini di efficienza e produttività.
- **Velocità e facilità nell'utilizzo.** Il sistema è molto semplice da utilizzare; viene fornita una finestra di testo in cui si deve inserire la "domanda" da porre all'intelligenza artificiale. A prescindere dalla lunghezza del quesito, la risposta del sistema è generalmente veloce e consiste in un'attesa di pochi secondi. L'utilizzo dello strumento da parte dell'utente si traduce in una semplice conversazione (chat) con il fine di identificare una soluzione di un problema.
- **Disponibilità ed accessibilità.** Al contrario di altri tipi di intelligenza artificiale basate sui LLM, ChatGPT è gratuito ed utilizzabile in Italia; inoltre è disponibile all'uso sia attraverso la piattaforma web, che tramite una comoda e semplice applicazione per desktop.

Oltre a queste caratteristiche, che costituiscono sicuramente dei vantaggi determinanti, è necessario tenere conto di alcune limitazioni associate all'uso degli LLM:

- **Rischio sulla correttezza dell'output.** Il sistema può generare delle risposte parzialmente o totalmente errate; ad esempio potrebbe fornire una spiegazione non pertinente alla domanda effettuata o una soluzione implementativa non funzionante.
- **Limitazione sui dati di addestramento.** La versione di ChatGPT utilizzata è limitata alle informazioni elaborate in fase di addestramento. Il modello potrebbe quindi non essere in grado di fornire risposte a determinati quesiti, oppure potrebbe generare delle soluzioni implementative deprecate o basate su componenti non più utilizzabili.

Quindi, sulla base di queste considerazioni, è possibile affermare che ChatGPT rappresenta uno strumento straordinariamente versatile e avanzato. Tuttavia, è fondamentale che il programmatore lo utilizzi con attenzione e discrezione.

ChatGPT è stato utilizzato come supporto nella realizzazione dell'applicativo descritto precedentemente. Nelle sezioni successive vengono evidenziati e descritti alcuni dei casi più particolari riguardo all'utilizzo del modello, fornendo, inoltre, alcune considerazioni su come è stato utilizzato lo strumento preso in esame.

4.1.1 Ideazione ed analisi dei requisiti

Durante le fasi di ideazione ed analisi, ChatGPT è stato consultato sommariamente; l'ideazione dell'applicativo rappresenta l'operazione che dà origine all'idea su cui si fonda l'intero sistema poi progettato ed implementato.

L'intelligenza artificiale ha trovato difficile impiego in questa sezione a causa di una motivazione fondamentale; il concetto stesso di "idea", sui cui si basa un'applicazione, deriva da un'operazione di ragionamento comune al team di sviluppo. L'obiettivo principale da raggiungere in questa fase era infatti trovare un'idea che fosse accattivante, innovativa (nel contesto dell'ateneo) e soprattutto personale. Per questa ragione ChatGPT è stato inizialmente consultato per proporre delle idee, che sono successivamente state scartate.

Al contrario, durante la procedura di analisi e stesura dei requisiti, il modello è stato consultato per chiarire dei dubbi riguardanti i sistemi utilizzati per formalizzare le idee definite in fase di ideazione. Ad esempio, è stato utilizzato nella fase di stesura del diagramma dei casi d'uso e modello di dominio con lo scopo di rappresentare correttamente le varie strutture all'interno dei rispettivi diagrammi. Tuttavia, come nel caso precedente, ChatGPT non è stato utilizzato per identificare i requisiti funzionali e non funzionali che caratterizzano l'applicativo, i quali sono stati invece scoperti ed ampliati dal team di sviluppo durante le varie fasi della creazione dell'app.

4.1.2 Implementazione

Come anticipato precedentemente, ChatGPT è stato utilizzato, nella maggioranza dei casi, durante la fase di implementazione, che consiste nella costruzione, dal punto di vista programmatico, dell'architettura dell'app e nella scrittura del codice relativo alla logica applicativa. In questo contesto il modello si è rivelato estremamente utile, perché ha permesso di risolvere problemi implementativi, correggere errori nella scrittura del codice, migliorare la user experience ed identificare idee e concetti da implementare nell'app.

Styling dell'interfaccia

Un'interfaccia con uno stile grafico ben definito e comune tra le varie schermate rende l'utilizzo dell'applicazione molto più gradevole e semplice per l'utente. Da un punto di vista puramente programmatico, i componenti che costituiscono l'interfaccia grafica sono rappresentati dai Fragment (e dai ViewModel); durante la loro definizione e costruzione, ChatGPT ha permesso di integrare molteplici funzionalità e dettagli, che permettono di migliorare l'esperienza utente.

L'esempio preso in considerazione è relativo allo stile delle schermate. Come è possibile osservare, nella Figura 2.14 e Figura 2.23 i vari componenti di tipo `EditText` presentano il medesimo stile caratterizzato da un semplice contorno rosso con sfondo bianco. L'obiettivo, in questo caso, era mantenere coerenza tra le varie view; per effettuare questa operazione ChatGPT ha suggerito di definire un file, all'interno della directory relativa alle risorse, che svolgesse la funzione di styling del componente. Il chat-bot ha inoltre definito i vari parametri, che permettono di personalizzare la schermata come richiesto. Infine, la discussione effettuata con il sistema di intelligenza artificiale ha permesso di comprendere il funzionamento dei file associati alle operazioni di styling, che sono stati utilizzati anche in altre occasioni, ad esempio nella definizione delle proprietà grafiche dei `FragmentDialog`.

Gestione delle animazioni

All'interno delle varie schermate che compongono l'app sono presenti diversi componenti grafici, come pulsanti, menu, card, ecc. Nel contesto di un'applicazione per dispositivi mobili, una schermata deve disporre di dettagli grafici che la rendano accattivante, ma non confusionaria per l'utente. Sulla base di questo proposito ChatGPT ha permesso di comprendere la gestione delle animazioni dei componenti presenti a schermo, le quali permettono di muovere gli elementi grafici in risposta a specifiche azioni compiute dall'utente.

Per utilizzare un'animazione è sufficiente definire un file con una serie di parametri all'interno della directory "anim", presente tra le risorse dell'app. Tali parametri indicano la tipologia di animazione da eseguire (movimento, ingrandimento, trasparenza, ecc). Successivamente è sufficiente dichiarare un oggetto di tipo `Animation` all'interno del codice dove caricare l'animazione ed invocare il metodo `startAnimation()` per eseguire l'animazione creata su un componente specifico.

Alcuni esempi di utilizzo delle animazioni all'interno dell'applicazione descritta precedentemente si trovano nelle schermate relative alla visualizzazione delle segnalazioni (ad esempio l'apertura e la chiusura del menu che si vede in Figura 2.18) ed alla selezione degli edifici preferiti (il cuore nella Figura 2.20 si allarga e si restringe per simulare un battito).

Splash screen

Come descritto in precedenza, quando l'utente apre l'applicazione, il sistema deve effettuare una serie di controlli per verificare lo stato della sessione. Ad esempio se l'utente ha effettuato l'operazione di logout, deve essere indirizzato alla schermata di autenticazione; al contrario, se la sessione è ancora attiva, l'utente deve essere indirizzato verso la bacheca.

Il componente Splash Screen (si veda Figura 4.1) è stato suggerito da ChatGPT per "nascondere" in modo elegante il processo di caricamento dei dati. Il modello è stato inoltre di grande aiuto durante l'implementazione di questo sistema, perché ha permesso di identificare un meccanismo semplice ed intuitivo per coordinare le operazioni di controllo della sessione con l'animazione compiuta dallo Splash Screen nell'interfaccia utente. L'intelligenza artificiale ha inoltre saputo adattarsi alle domande effettuate, personalizzando come richiesto le animazioni messe in atto dallo Splash Screen, creando una schermata semplice, ma appetibile graficamente per l'utente.

Shared Preferences e Safe Args

Il chat-bot si è rivelato utile nella costruzione del meccanismo utilizzato per memorizzare localmente gli edifici di interesse dell'utente tramite le Shared Preferences. Le chat condotte hanno permesso di comprendere il funzionamento di questo sistema, che è molto simile alla scrittura e lettura di un file. Tali operazioni vengono inoltre messe in atto in combinazione con l'utilizzo della libreria GSON [6], che permette di effettuare la serializzazione (e deserializzazione) di un dato o oggetto complesso. In relazione a ciò, ChatGPT ha fornito i metodi richiesti per la corretta gestione dell'intero meccanismo utilizzando i componenti indicati precedentemente, che hanno consentito di implementare correttamente le procedure di salvataggio e recupero degli edifici preferiti.

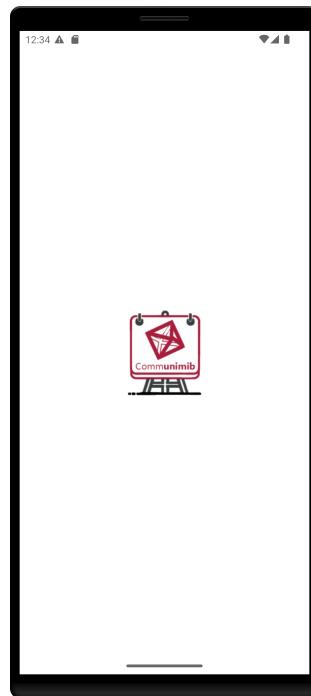


Figura 4.1: Splash Screen con il logo dell'app

Il modello ha inoltre fornito le soluzioni implementative per integrare correttamente il meccanismo delle Safe Args, utilizzato per trasferire dati complessi (come oggetti) da un Fragment ad un altro. Per effettuare questa semplice operazione è necessario personalizzare il file di tipo **navigation**, che permette di gestire il passaggio da una schermata ad un'altra, integrando il campo **argument**, che definisce il tipo di dato da trasferire. Infine, quando si effettua l'operazione di navigazione, è sufficiente utilizzare il rispettivo componente di tipo **Directions** per trasferire l'argomento desiderato, per poi recuperarlo nel Fragment di destinazione, utilizzando il rispettivo oggetto di tipo **Args**. Questa procedura è stata utilizzata per gestire la visualizzazione dettagliata dei post della bacheca (si veda Figura 2.15) e delle segnalazioni (si veda Figura 2.22).

Scrittura di algoritmi e risoluzione generale di problematiche

Una delle caratteristiche principali di ChatGPT è la capacità di adattamento a fronte delle varie richieste poste. In molti casi, è possibile fornire una descrizione testuale di un algoritmo e richiedere la relativa implementazione nel linguaggio di programmazione specificato. Questa operazione è stata applicata diverse volte durante la fase di scrittura del codice, ad esempio nella costruzione della schermata relativa ai filtri applicabili alle segnalazioni (si veda Figura 2.21) e la schermata di scelta degli edifici preferiti (si veda Figura 2.20).

Nel primo caso si è rivelato necessario implementare un meccanismo per la gestione delle selezioni effettuate nella schermata; ad esempio un utente non può selezionare contemporaneamente le prime due Checkbox presenti nell'interfaccia grafica. In un primo momento, l'implementazione di questo semplice comportamento si è rivelato difficoltoso a causa delle proprietà del componente **Checkbox**. Tuttavia, dopo aver fornito il codice prodotto a ChatGPT, quest'ultimo si è rivelato in grado di risolvere il problema, suggerendo la soluzione implementativa corretta, rimodellando il codice fornito in input.

Nel secondo caso, invece, il componente **ListView**, contenente la lista degli edifici presenti nell'ateneo, non effettuava il salvataggio grafico delle preferenze scelte dall'utente. Questo problema veniva generato da alcune proprietà del componente **ListView**, che non è in grado di mantenere memorizzate, dopo l'operazione di "scroll", le operazioni compiute dall'utente. ChatGPT è stato in grado di identificare la sorgente del problema e ha fornito la soluzione implementativa corretta per risolverlo.

Identificazione di componenti

Durante le fasi implementative è spesso capitato di voler includere una specifica funzionalità nell'app senza sapere quale componente fosse responsabile della sua gestione, rendendo necessaria la sua identificazione.

ChatGPT è un sistema estremamente utile per effettuare questo tipo di operazione; a partire da una descrizione fornita come input, il modello è stato in grado, nella maggioranza dei casi, di identificare il componente, fornendo inoltre degli esempi di utilizzo. Ciò ha permesso, ad esempio, di identificare componenti grafici complessi come i **DialogFragment** ed elementi grafici unitari, come i **FloatingActionButton**, le **Checkbox**, lo **Slider** utilizzato nelle schermate

relative alla bacheca (si veda ad esempio Figura 2.15) e l’ImagePicker utilizzato nella sezione del profilo (si veda Figura 2.26). ChatGPT ha, infine, suggerito dettagli sulle rispettive librerie.

È importante precisare che questa ”feature” del modello di intelligenza artificiale non può sostituire la consultazione della documentazione e si presta esclusivamente all’identificazione del componente, piuttosto che alla corretta implementazione.

4.1.3 Testing

Durante la fase di testing, che consiste nella verifica e validazione dei componenti implementati, ChatGPT è stato utilizzato principalmente come supporto durante la scrittura dei test automatici relativi all’interfaccia utente. Il modello ha innanzitutto suggerito l’uso della libreria Espresso, in combinazione con JUnit4 ed ha fornito alcuni esempi di utilizzo, utili a comprenderne il funzionamento. Dopo aver consultato la documentazione relativa alle librerie consigliate, i test UI sono stati integrati correttamente nel sistema.

Inoltre, il maggiore contributo del modello in questa fase è avvenuto durante la stesura del meccanismo utilizzato per gestire i casi di test relativi alla `MainActivity`, attraverso l’oggetto di tipo `ActivityScenario` ed in combinazione con il metodo `setUp()` descritti nel capitolo precedente. ChatGPT ha fornito, dopo una lunga discussione, la corretta soluzione per gestire la problematica riscontrata.

4.1.4 Problematiche incontrate

Prendendo in considerazione solo ed esclusivamente le considerazioni precedenti, che rappresentano alcuni tra gli esempi di utilizzo più rilevanti, potrebbe sembrare che ChatGPT sia uno strumento praticamente perfetto. Tuttavia, nonostante l’alto grado di precisione nelle risposte fornite, in alcuni casi produce degli errori. Durante le varie fasi di analisi ed implementazione dell’applicativo sono state identificate alcune situazioni ”comuni” che portano il sistema a commettere errori o fornire delle soluzioni inadatte all’integrazione:

- **Posizionamento.** Nella maggioranza dei casi, ChatGPT non è stato in grado di riconoscere e rispondere a richieste relative al posizionamento di componenti grafici sullo schermo. L’output non rispecchia quanto richiesto oppure utilizza delle proprietà di posizionamento non esistenti o non funzionanti.
- **Dimensione.** Come nel caso descritto precedentemente, il modello interpreta con difficoltà le richieste relative alla gestione della dimensione dei componenti grafici.
- **Styling complesso.** ChatGPT è in grado di gestire richieste più o meno complesse. Tuttavia non è in grado di rispondere a richieste eccessivamente elaborate, soprattutto se queste riguardano lo stile grafico dei componenti. In questo caso è molto più semplice ”scomporre” la domanda posta in quesiti più semplici al fine di identificare le proprietà da implementare.

- **Ottimizzazione.** Dopo aver scritto un lungo blocco di codice, il programmatore potrebbe volerlo migliorare, ad esempio rimuovendo operazioni inutili con lo scopo di facilitarne la comprensione e migliorarne le prestazioni. In alcuni casi sono state fornite delle procedure a ChatGPT al fine di effettuare l'operazione di ottimizzazione, ma il modello ha prodotto dei risultati insoddisfacenti; in alcuni casi il comportamento dell'algoritmo fornito è stato completamente stravolto.

Per cercare di massimizzare l'utilità delle discussioni, può rivelarsi utile scrivere delle domande che abbiano un contenuto di facile comprensione, cercando soprattutto di fornire un contesto per "aiutare" ChatGPT a produrre delle risposte coerenti ed utili.

4.1.5 Metodo di verifica ed analisi dei dati raccolti

In relazione a quanto descritto nelle sezioni precedenti, durante lo sviluppo dell'applicativo Communimib, è stato compilato un diario contenente tutte le discussioni effettuate con ChatGPT. Ciascuna chat è stata etichettata sulla base della fase del processo di sviluppo software, fornendo inoltre un commento sull'utilità. Al fine di valutare complessivamente l'utilizzo dello strumento proposto, sono stati prodotti i seguenti grafici.

Durante lo sviluppo dell'app sono state condotte 86 chat differenti con il modello di intelligenza artificiale. Tra queste, 47 si sono rivelate utili, 15 hanno trovato media utilità e 24 hanno avuto poco impatto. Il grafico in Figura 4.2 permette di mettere in evidenza tale relazione, che certifica come il modello sia generalmente utile durante le fasi che compongono lo sviluppo del software.

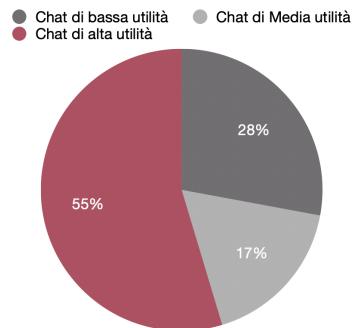


Figura 4.2: Utilità di ChatGPT

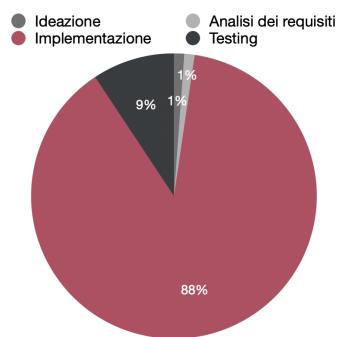


Figura 4.3: Utilizzo di ChatGPT

Il grafico rappresentato nella Figura 4.3 mette in evidenza la suddivisione delle chat nelle varie fasi che compongono lo sviluppo del software. Come è possibile notare, a supporto delle affermazioni precedenti, la maggior parte delle discussioni con ChatGPT sono identificabili nelle fasi di implementazione e testing; al contrario durante le fasi di ideazione ed analisi dei requisiti è presente una minoranza delle chat.

Infine, i grafici esposti in Figura 4.4 e Figura 4.5 mettono in evidenza i casi di successo e fallimento in relazione ai metodi di utilizzo di chatGPT in fase implementativa ed i problemi descritti precedentemente.

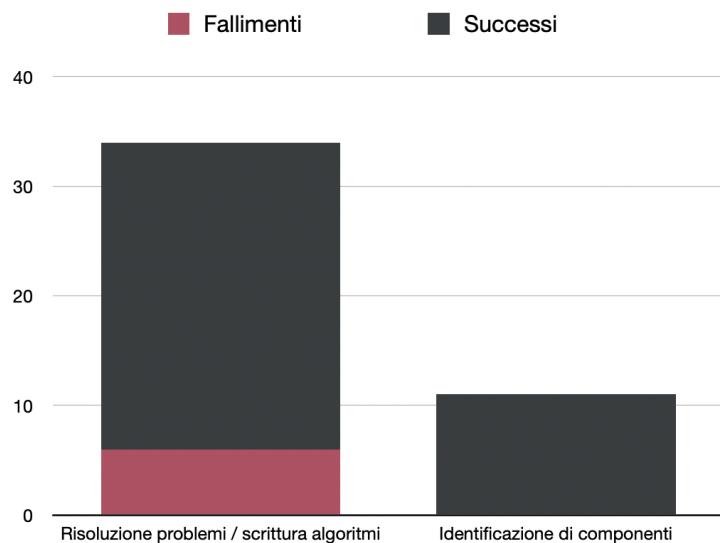


Figura 4.4: Chat di successo e fallimento nei metodi di utilizzo di ChatGPT

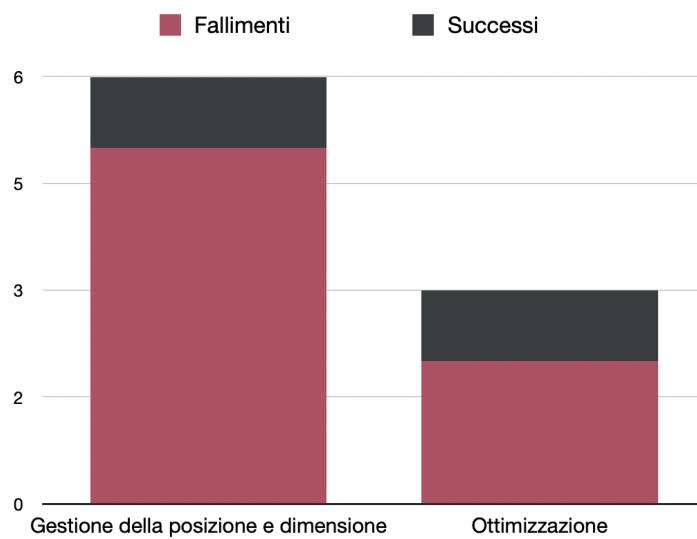


Figura 4.5: Chat di successo e fallimento nei casi delle problematiche incontrate

4.2 Design del logo Communimib

Uno degli elementi che permettono di caratterizzare maggiormente un'applicazione è il logo. Quest'ultimo deve avere uno stile riconoscibile e deve presentare degli elementi che permettano, a grandi linee, di rappresentare graficamente le funzionalità messe in atto dall'applicativo. Si tratta di un componente molto importante, perché permette di rendere l'applicazione riconoscibile ed identificabile all'utente; inoltre può anche essere utilizzato all'interno dell'app per personalizzare le schermate di caricamento migliorando l'interfaccia grafica.



Figura 4.6: Generazione e modifiche applicate al logo di Communimib

L'utilizzo dell'intelligenza artificiale ha permesso di realizzare un modello di base per il logo utilizzato all'interno dell'app, che è stato successivamente migliorato, utilizzando uno strumento di editing delle immagini, come si può notare nella Figura 4.6.

Per effettuare questa operazione è stato utilizzato il modello di intelligenza artificiale Copilot - Designer. Come nel caso di ChatGPT, anche in questo caso è sufficiente fornire al modello una descrizione testuale dell'immagine che si vuole ottenere; inoltre è possibile fornire dei dettagli per caratterizzare l'immagine, come i colori dei vari componenti oppure lo stile. Ad esempio, per generare l'immagine rappresentata precedentemente, sono stati forniti alcuni dei colori utilizzati per costruire la paletta dell'applicativo; inoltre è stato specificato uno stile minimalista.

La differenza sostanziale con ChatGPT riguarda invece la tipologia di modello. ChatGPT si basa sulla tecnologia LLM, mentre Copilot - Designer, così come la maggior parte delle intelligenze artificiali che permettono di generare immagini, suoni e video, è strutturato sulla base della tecnologia GANs (Generative Adversarial Networks).

Nonostante questa differenza, tale tipologia di modello di intelligenza artificiale può essere di grande supporto per il programmatore, perché permette di generare immagini o altri contenuti a partendo da un semplice linguaggio naturale.

Conclusioni

Sulla base dei dati raccolti e delle considerazioni effettuate nei capitoli precedenti, è possibile affermare che ChatGPT (e l'IA in generale) sia uno strumento incredibilmente utile da utilizzare come supporto durante le fasi che compongono lo sviluppo di un software. La caratteristica principale di ChatGPT è sicuramente la versatilità di utilizzo; come esposto precedentemente, il modello di IA è stato utilizzato principalmente per implementare meccanismi e risolvere problemi in fase di implementazione. Tuttavia ciò non significa che ChatGPT non possa essere utilizzato in modo ancora più vario, a seconda delle esigenze e della fantasia del programmatore.

Nonostante ciò, il tema dell'uso dell'intelligenza artificiale è al centro di molteplici dibattiti riguardanti la sua effettiva utilità ed i possibili metodi di applicazione. I principali dubbi riguardano sicuramente l'evoluzione del mondo del lavoro; uno strumento con la capacità di generare il codice, in futuro, sostituirà la figura lavorativa del programmatore? Sicuramente, prima o poi, il ruolo del programmatore cambierà e lo farà a prescindere dall'utilizzo o meno delle intelligenze artificiali; l'innovazione tecnologica è infatti un fattore determinante, che porta a continui cambiamenti all'interno del mondo informatico.

ChatGPT è uno strumento dalle grandi potenzialità, ma, al momento, è strettamente legato all'utilizzo da parte del programmatore. Quest'ultimo è in grado di pensare, ragionare, immaginare e sognare; l'IA può essere quindi di aiuto per realizzare e mettere in atto queste capacità. L'adozione dell'intelligenza artificiale nel processo di sviluppo software, non solo permette di migliorare efficienza e produttività, ma apre anche nuove frontiere nella creatività ed innovazione del programmatore. Per questa ragione è importante considerare ChatGPT (e l'IA in generale) non come un pericolo, ma come un'opportunità che è possibile utilizzare per migliorare il nostro futuro.

"C'è potenziale nell'evoluzione tecnologica: abbracciamola senza timori, per innovare e progredire insieme!"

(generata da ChatGPT v3.5)

Ringraziamenti

Desidero esprimere la mia sincera e profonda gratitudine a tutti coloro che mi hanno supportato durante il mio percorso accademico. In particolare, un sentito ringraziamento va ai miei relatori, la Prof.ssa Daniela Micucci e la Dott.ssa Maria Teresa Rossi, per l'opportunità di stage offertami e per la loro costante disponibilità.

Vorrei inoltre ringraziare i miei colleghi ed amici Luca Pincioli e Giulia Vitali, con i quali ho avuto il piacere di intraprendere e condividere il percorso di stage. La nostra collaborazione, unita al sostegno reciproco, è stata fondamentale per la realizzazione di questo grande progetto. Il loro impegno ed entusiasmo sono stati una grande fonte di ispirazione e motivazione.

Desidero infine ringraziare i miei familiari ed i miei amici. Senza il loro costante e incondizionato incoraggiamento e sostegno, non sarei riuscito ad affrontare e superare le numerose sfide che ho incontrato lungo il cammino. La loro presenza e il loro affetto mi hanno dato la forza necessaria per arrivare fino in fondo.

Sitografia

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