Università degli Studi di Milano - Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea in Informatica

Sviluppo di applicazioni mobile con supporto di ChatGPT: UniFolder

Relatore: Prof. Micucci Daniela

Correlatore: Dott. Rossi Maria

Tesi di Laurea di:
Manuel Bosisio
Matricola 886370

Anno Accademico 2023-2024

Indice

1 Introduzione **3**
- 1.1 ChatGPT e sviluppo di applicazioni mobile . . . 4
- 1.2 Descrizione di UniFolder . . . 5
  - 1.2.1 Funzionalità . . . 5
  - 1.2.2 Obiettivi . . . 6
- 1.3 Utilizzo di ChatGPT . . . 6

2 Analisi dei requisiti **8**
- 2.1 Obiettivi utente . . . 8
- 2.2 Requisiti funzionali . . . 9
- 2.3 Requisiti non funzionali . . . 10
- 2.4 Requisiti di sistema . . . 12
  - 2.4.1 Requisiti Software . . . 12
  - 2.4.2 Requisiti Hardware . . . 13
- 2.5 Requisiti di interfaccia utente . . . 13
- 2.6 Dipendenze e vincoli . . . 14
  - 2.6.1 Dipendenze da terze parti . . . 14
  - 2.6.2 Vincoli tecnologici . . . 16
- 2.7 Casi d'uso . . . 17
  - 2.7.1 Caso d'Uso 1: Registrazione di un Nuovo Utente . . . 17
  - 2.7.2 Caso d'Uso 2: Login dell'Utente . . . 18
  - 2.7.3 Caso d'Uso 3: Ricerca di Materiale Didattico . . . 18
  - 2.7.4 Caso d'Uso 4: Caricamento di Materiale Didattico . . . 18
- 2.8 Conclusioni . . . 19

3 Descrizione dell'implementazione **20**
- 3.1 Architettura del Sistema . . . 20
  - 3.1.1 Panoramica dell'architettura . . . 20
  - 3.1.2 Architettura e componenti principali . . . 22
- 3.2 Dettagli dell'implementazione e funzionalità . . . 27
  - 3.2.1 Login . . . 28
  - 3.2.2 Signup . . . 28
  - 3.2.3 Home - Ricerca . . . 29
  - 3.2.4 Risultati di ricerca . . . 29
  - 3.2.5 Visualizzazione documento . . . 30
  - 3.2.6 Caricamento file . . . 30
  - 3.2.7 Profilo - Impostazioni . . . 31
- 3.3 Design . . . 31

3.4 Sviluppi futuri . . . 32
3.4.1 Nuovi obiettivi utente . . . 32
3.4.2 Miglioramento features . . . 32
3.5 Conclusioni . . . 32

4 Progetto di Testing **34**
4.1 Unit Testing . . . 35
4.1.1 Implementazione Unit Tests . . . 36
4.2 Integration Testing . . . 42
4.3 UI Testing . . . 42
4.3.1 Implementazione UI Tests . . . 43
4.4 Conclusioni . . . 45

5 Utilizzo di ChatGPT **46**
5.1 Analisi dei requisiti . . . 47
5.1.1 Considerazioni sull'utilizzo . . . 48
5.1.2 Conclusione . . . 48
5.2 Descrizione dell'implementazione . . . 49
5.2.1 Considerazioni sull'utilizzo . . . 50
5.2.2 Conclusione . . . 50
5.3 Progetto di Testing . . . 51
5.3.1 Considerazioni sull'utilizzo . . . 52
5.3.2 Conclusione . . . 53
5.4 Conclusioni . . . 53

6 Conclusioni **55**
6.1 L'importanza della definizione di un prompt . . . 55
6.2 Il non-determinismo di ChatGPT . . . 55
6.3 Utilità e percezione dello strumento . . . 56
6.3.1 Ethopoeia e ChatGPT: Un'Analisi . . . 57
6.4 Commento finale . . . 58

Capitolo 1

Introduzione

Negli ultimi anni, l'intelligenza artificiale è diventata un tema sempre più centrale, con crescente attenzione su come essa rivoluzionerà la nostra quotidianità nel prossimo futuro. In un'era segnata dall'innovazione continua e da tecnologie sempre più avanzate, i *Large Language Models* (LLM) rappresentano a oggi un capitolo fondamentale nella storia dell'intelligenza artificiale (IA).

I modelli linguistici di grandi dimensioni sono un tipo di intelligenza artificiale progettata per imitare le capacità di elaborazione del linguaggio umano. Utilizzano tecniche di *deep learning*, come le reti neurali, e vengono addestrati su enormi quantità di dati testuali provenienti da varie fonti, inclusi libri, articoli, siti web e altro. In particolare, l'addestramento estensivo consente agli LLM di generare testi altamente coerenti e realistici. Gli LLM analizzano i modelli e le connessioni all'interno dei dati su cui sono stati addestrati e utilizzano queste conoscenze per prevedere quali parole o frasi è probabile che compaiano successivamente in un contesto specifico. Questa capacità di comprendere e generare linguaggio è utile in vari campi dell'elaborazione del linguaggio naturale (NLP) come la traduzione automatica e la generazione di testi, e sta divenendo sempre più di supporto al lavoro umano nei più svariati ambiti [1].

L'utilizzo degli LLM come assistenti per i programmatori, ad esempio, è realizzato attraverso *Chat Support*, una chat all'interno della quale il modello fornisce suggerimenti, spiegazioni o completamenti di codice attraverso messaggi di testo. Gli LLM possono quindi generare codice completo in risposta a descrizioni di task o domande specifiche poste dai programmatori, aiutandoli a completare compiti di programmazione in modo più efficiente; i dati raccolti dalle interazioni possono a loro volta essere utilizzati per ottimizzare i suggerimenti forniti dal modello, ad esempio identificando quali suggerimenti vengono accettati o modificati più frequentemente [2].

Il campo degli LLM per le conversazioni di *Emotional Support* ha registrato notevoli progressi negli ultimi tempi, con vari modelli che hanno dimostrato capacità impressionanti nel dialogo a dominio aperto. I primi studi a riguardo si sono concentrati sulla raccolta e l'annotazione di serie di dati derivati dai social media, forum online, o trascrizioni di video di psicoterapia. Tuttavia, la strategia di supporto emotivo è trascurata e la qualità del dialogo è insoddisfacente in

questi set di dati. A tal fine, è stato possibile costruire un set di dati di conversazioni di supporto emotivo, che progetta con attenzione il processo di raccolta dei dati e meccanismi multipli per garantire l'efficacia delle strategie di supporto emotivo nelle conversazioni. In particolare, le sfide emotive più diffuse derivano da problemi di comunicazione, problemi derivanti da pressioni legate al lavoro o dalla disoccupazione. Mentre gli utenti mostrano spesso sentimenti negativi, gli assistenti propendono per espressioni positive, offrendo supporto emotivo [3].

Un altro caso di impiego per gli LLM riguarda il supporto alle pratiche cliniche. Il task a cui vengono sottoposti riguarda il comporre note mediche per pazienti ricoverati in terapia intensiva, dopo aver fornito informazioni su trattamenti, parametri di analisi e respiratori, in ordine casuale. Il modello è allora in grado di categorizzare correttamente la maggior parte dei parametri nelle sezioni appropriate, e sebbene non sia stato progettato per rispondere a domande di carattere medico e, pertanto, non disponga delle competenze mediche e del contesto necessari per comprendere appieno le complesse relazioni tra le diverse condizioni e i trattamenti, ha dimostrato la capacità di fornire suggerimenti significativi per ulteriori trattamenti sulla base delle informazioni fornite. La caratteristica migliore degli LLM rispetto a questo impiego è legata alla capacità di riassumere le informazioni, utilizzando un linguaggio tecnico per la comunicazione tra clinici e un linguaggio semplice per la comunicazione con i pazienti e le loro famiglie [1].

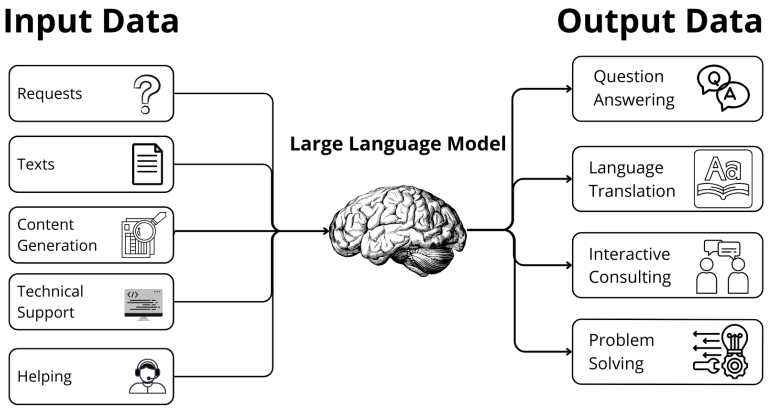


Figura 1.1: LLMs

1.1 ChatGPT e sviluppo di applicazioni mobile

Il progetto di stage presentato ha come obiettivo principale quello di valutare l'efficacia di ChatGPT-3.5 come supporto nello sviluppo di applicazioni mobile Android. In particolare, il progetto si focalizza sulla creazione di un'applicazione per lo scambio di appunti tra studenti in ambito universitario. L'idea alla base è

quella di esplorare se e come un modello di linguaggio avanzato come ChatGPT possa assistere i programmatori durante le varie fasi del ciclo di sviluppo, dalla progettazione alla scrittura del codice, fino al Testing e alla risoluzione dei bug.

Nello specifico ChatGPT fa parte della famiglia degli InstructGPT, quindi modelli formati tramite *deep learning* ma poi ottimizzati tramite il rinforzo umano (Tecnica nota come RLHF – *Reinforcement Learning From Human Feedback*). Tramite ChatGPT è possibile usare diversi LLM, tra i quali ChatGPT-3.5, uno dei più avanzati modelli di linguaggio di grandi dimensioni (LLM) sviluppati da OpenAI [4].
Questo gli consente di generare testo altamente coerente e realistico, analizzando modelli e connessioni nei dati su cui è stato addestrato e utilizzando tale conoscenza per predire le parole o le frasi successive in un dato contesto.
Grazie alla sua capacità di comprendere e generare linguaggio naturale, ChatGPT trova applicazione in diversi ambiti della elaborazione del linguaggio naturale (NLP), come la traduzione automatica, la generazione di testo, e in questo caso specifico, il supporto alla programmazione. Il nostro esperimento mira a determinare se ChatGPT può essere un valido strumento di supporto per i programmatori, migliorando l'efficienza e la qualità del processo di sviluppo di applicazioni mobile.

Nelle sezioni seguenti, descriveremo nel dettaglio il progetto di sviluppo dell'applicazione per lo scambio di appunti tra studenti, analizzando le interazioni con ChatGPT e valutando i risultati ottenuti.

1.2 Descrizione di UniFolder

UniFolder è un'applicazione Android pensata per la condivisione di materiale didattico tra studenti all'interno dell'università. Gli utenti possono registrarsi gratuitamente per avere accesso a dispense, appunti, presentazioni e esercizi relativi ai corsi universitari, oltre che a caricare il proprio materiale in modo da renderlo fruibile ai colleghi e agli utenti dell'app.

1.2.1 Funzionalità

- **Caricamento e condivisione di documenti**: Gli studenti possono caricare e condividere documenti PDF come dispense, appunti, esercizi relativi ai corsi universitari. Questa funzionalità rende il materiale didattico accessibile a tutti gli utenti dell'app, favorendo la collaborazione e lo scambio di risorse tra studenti.

- **Organizzazione e ricerca avanzata**: UniFolder consente agli utenti di organizzare i documenti in categorie o tramite l'uso di tag. Inoltre, l'app offre funzionalità di ricerca avanzata, permettendo di trovare rapidamente il materiale desiderato in base al corso, all'argomento o al titolo. Questo facilita l'accesso rapido e preciso alle risorse necessarie per lo studio.

1.2.2 Obiettivi

L'obiettivo principale di Unifolder è quello di creare una piattaforma centralizzata e accessibile per la condivisione di materiale didattico universitario. Gli obiettivi specifici includono:

- **Facilitare l'accesso al materiale didattico**: Fornire agli studenti un modo semplice e immediato per accedere a documenti utili per i loro corsi.
- **Promuovere la collaborazione tra studenti**: Incentivare la condivisione di risorse e informazioni tra studenti, migliorando così il processo di apprendimento collettivo.
- **Ottimizzare l'organizzazione dei documenti**: Permettere agli utenti di organizzare il materiale in modo strutturato e di trovarlo rapidamente grazie alle funzionalità di ricerca avanzata.

UniFolder vuole quindi fornire una piattaforma integrata per facilitare l'accesso alle risorse educative, promuovendo la collaborazione tra gli studenti e l'efficienza nello studio. L'app supporta gli studenti nel loro percorso accademico, fornendo loro gli strumenti necessari per una formazione continua e di qualità.

1.3 Utilizzo di ChatGPT

Durante la realizzazione del progetto dell'applicazione UniFolder, ChatGPT ha fornito supporto agli sviluppatori in tutte le fasi dello sviluppo. Inizialmente, nella fase di analisi dei requisiti, ChatGPT ha aiutato a chiarire e definire le funzionalità essenziali dell'applicazione, come la condivisione di materiali didattici, la gestione dei documenti e la personalizzazione delle interfacce utente. Grazie alle sue capacità di elaborazione del linguaggio naturale, ha facilitato la raccolta e l'organizzazione dei requisiti, garantendo che tutte le esigenze degli utenti fossero ben documentate e comprese.

Nella fase di disegno della soluzione, ChatGPT ha contribuito alla definizione dell'architettura dell'applicazione, suggerendo pratiche per la struttura del codice, l'organizzazione dei componenti e l'uso delle tecnologie più appropriate, seppur l'ultima decisione sia sempre spettata ai programmatori.
Durante l'implementazione, ha fornito frammenti di codice, risolto dubbi tecnici e proposto soluzioni più o meno efficaci ai problemi incontrati. Inoltre, ChatGPT ha giocato un ruolo significativo nel Testing e nel debugging dell'app, fornendo linee guida per l'uso di strumenti come Mockito o Robolectric per i test unitari e di integrazione. Infine, per la documentazione del progetto, ChatGPT ha assistito nella stesura di descrizioni chiare e concise delle funzionalità e delle decisioni progettuali, assicurando una documentazione completa e facilmente comprensibile. Complessivamente, ChatGPT ha offerto un supporto continuo e versatile, contribuendo in modo significativo al successo del progetto UniFolder, evolvendo di pari passo con gli sviluppatori.

Alla luce di quanto enunciato sopra, ChatGPT è stato ritenuto un valido supporto al lavoro svolto dai programmatori, nonostante non fosse impeccabile

in materia di dettagli implementativi o utilizzo di librerie esterne, e talvolta restituisse risposte non adeguate al contesto. Al contempo, è emerso quanto fondamentale sia istruire il ChatBot con un prompt il più accurato possibile: si entrerà nel merito della questione successivamente.

Capitolo 2

Analisi dei requisiti

L'analisi dei requisiti nello sviluppo software è un passaggio cruciale per garantire che l'applicazione soddisfi tutte le funzionalità e i requisiti di sicurezza previsti. Questo processo consente di definire in modo dettagliato le variabili di input/output, gli stati, le procedure di avvio/arresto e le modalità operative, delineando esattamente ciò che il sistema deve realizzare. Una specificazione chiara e non ambigua dei requisiti permette di identificare e risolvere eventuali conflitti tra obiettivi e vincoli del sistema nelle prime fasi del ciclo di sviluppo, riducendo così la necessità di costosi *redesign* e *recoding*.
Oltre a questo, l'analisi dei requisiti contribuisce alla robustezza del software documentando e verificando le assunzioni ambientali, assicurando l'affidabilità del sistema. La verifica delle assunzioni di valore sugli input e la specificazione dei vincoli temporali sono cruciali per prevenire fallimenti durante l'esecuzione. L'inclusione di loop di feedback nei requisiti consente di rilevare e correggere le carenze del sistema, migliorando la sicurezza e la capacità di gestire situazioni pericolose [5].

L'approccio a questa fase, così come all'intero progetto, segue infatti una modalità "Agile" per lo sviluppo software. Questa, si basa su un processo incrementale, in cui i requisiti vengono definiti, revisionati e modificati man mano che il progetto avanza. Invece di cercare di prevedere tutti i requisiti all'inizio del progetto, l'Agile consente di adattarsi ai cambiamenti e di rispondere alle nuove esigenze che emergono durante lo sviluppo. Questo metodo ha dimostrato di migliorare la soddisfazione lavorativa e la produttività, sebbene l'adozione di metodi agili richieda una valutazione attenta delle caratteristiche del progetto [6].

In sintesi, questo capitolo vuole definire un'analisi accurata e completa dei requisiti, avvenuta attraverso le modalità descritte, in quanto elemento chiave per lo sviluppo di un'applicazione Android sicura, affidabile e funzionale.

2.1 Obiettivi utente

Per la realizzazione dell'applicazione UniFolder, sono stati individuati preliminarmente una coppia di obiettivi utente che potessero guidare l'intero sviluppo.

Gli "obiettivi utente" si riferiscono ai risultati specifici che gli utenti finali si aspettano di ottenere attraverso l'utilizzo dell'applicazione. Questi obiettivi rappresentano le necessità e le aspettative degli utenti, guidando la progettazione e lo sviluppo dell'applicazione per assicurarsi che risponda efficacemente alle loro esigenze [7]. Nello specifico, sono stati individuati i seguenti:

1. **Caricamento e condivisione di documenti**: Gli studenti possono caricare e condividere documenti PDF come dispense, appunti, esercizi relativi ai corsi universitari. Questa funzionalità rende il materiale didattico accessibile a tutti gli utenti dell'app, favorendo la collaborazione e lo scambio di risorse tra studenti.

2. **Organizzazione e ricerca avanzata**: UniFolder consente agli utenti di organizzare i documenti in categorie o tramite l'uso di tag. Inoltre, l'app offre funzionalità di ricerca avanzata, permettendo di trovare rapidamente il materiale desiderato in base al corso, all'argomento o al titolo. Questo facilita l'accesso rapido e preciso alle risorse necessarie per lo studio.

2.2 Requisiti funzionali

La modellazione dei requisiti spesso implica la modellazione del comportamento dinamico, nonché delle funzionalità specifiche che l'applicazione deve offrire per soddisfare le esigenze degli utenti. Durante questa fase, vengono definiti i comportamenti e le caratteristiche del sistema, come le operazioni che gli utenti possono eseguire e le condizioni in cui queste operazioni devono essere disponibili [8].
Al fine di soddisfare gli obiettivi utente e di offrire un'esperienza completa, sono stati individuati più requisiti funzionali:

- **Autenticazione e Autorizzazione Utente**
  - Login/Logout: Gli utenti possono effettuare il login e il logout utilizzando le proprie credenziali.
  - Registrazione: Gli utenti possono registrarsi con un'email valida.
- **Ricerca del Materiale Didattico**
  - Ricerca per Corso di Studi e tag: Gli utenti possono cercare materiale didattico per Corso di Studi (es. Economia, Giurisprudenza, Medicina), specificando opzionalmente un tag di tipologia documenti (appunti lezione, domande esame, altro).
  - Ricerca per titolo: Gli utenti possono cercare materiale didattico specificando una *query* per il titolo dei documenti.
  - Ricerca combinata: Gli utenti possono cercare materiale didattico creando filtri a loro piacimento, sfruttando una combinazione delle precedenti.
- **Caricamento del Materiale Didattico**
  - Upload Documenti: Gli utenti possono caricare documenti didattici dal proprio dispositivo.

  - Assegnazione di Metadati: Durante il caricamento, gli utenti possono assegnare metadati ai documenti (es. titolo, descrizione, corso di studi).

- **Visualizzazione Documenti**
  - Esplorazione Documenti: Gli utenti possono visualizzare i documenti didattici disponibili, ad esempio dopo aver effettuato una ricerca.
  - Visualizzazione Documenti Caricati: Gli utenti possono visualizzare i documenti che hanno caricato.

- **Cronologia dei documenti**
  - Ultimi documenti aperti: Gli utenti possono visualizzare una lista degli ultimi documenti visualizzati.
  - Ultimi caricamenti: Gli utenti possono visualizzare l'elenco dei loro *upload* avvenuti in app.

- **Gestione del Profilo Utente**
  - Visualizzazione Profilo: Gli utenti possono visualizzare le proprie informazioni di profilo.
  - Modifica Profilo: Gli utenti possono aggiornare le proprie informazioni personali e la foto del profilo.

2.3 Requisiti non funzionali

Questa fase consiste nell'identificare e documentare le caratteristiche e le qualità del sistema che non sono direttamente legate a specifiche funzionalità, ma che influenzano l'esperienza dell'utente e il comportamento del sistema.
I requisiti non funzionali (noti anche come requisiti di qualità) sono in genere più difficili da esprimere in maniera misurabile, il che li rende più difficili da analizzare [8].

In generale, includono aspetti come le prestazioni, la sicurezza, la scalabilità, l'usabilità e la manutenibilità del software. Definire chiaramente i requisiti non funzionali è essenziale per garantire che il sistema soddisfi gli standard di qualità desiderati e funzioni in modo efficiente e sicuro nell'ambiente previsto, di seguito sono riportati in maniera specifica per l'applicazione UniFolder:

- **Prestazioni**
  - Tempo di Risposta: L'applicazione garantisce tempi di risposta rapidi per tutte le operazioni principali, come la ricerca di documenti e il caricamento di file, grazie all'utilizzo di un servizio *back-end* dedicato (si veda *Dipendenze e vincoli*).

- **Usabilità**
  - Interfaccia Intuitiva: L'interfaccia utente è progettata per essere minimale ed intuitiva, fornendo una UX (*User Experience*) adatta anche ad utenti esperti, per incontrare le diverse competenze di studenti da ogni facoltà.

  - Design Responsivo: Il layout dell'applicazione si adatta automaticamente alle dimensioni dello schermo del dispositivo, garantendo una buona esperienza utente su più modelli di smartphone Android.

- **Affidabilità**
  - Robustezza: Implementazione di meccanismi per gestire in modo efficiente errori e malfunzionamenti, garantendo la continuità del servizio, scongiurando possibilità di *crash*.

- **Scalabilità**
  - Supporto per Utenti Concorrenziali: L'applicazione è progettata per supportare un elevato numero di utenti contemporanei senza degrado delle prestazioni, sempre grazie all'utilizzo di servizi *storage* dedicati. Questo è in linea con la grande mole di utenze previste nel contesto accademico.
  - Espandibilità: Architettura modulare che facilita l'aggiunta di nuove funzionalità senza compromettere quelle esistenti.

- **Sicurezza**
  - Autenticazione e Autorizzazione: Meccanismi sicuri di autenticazione e autorizzazione per garantire che solo gli utenti legittimi possano accedere alle funzionalità dell'app.

- **Manutenibilità**
  - Codice Pulito e Documentato: Codice sorgente ben strutturato, secondo le linee guida Android, e ampiamente documentato per facilitare la manutenzione e l'aggiornamento dell'applicazione.
  - Testing Automatizzato: Suite di test automatizzati per garantire che nuove modifiche non introducano regressioni o nuovi bug.

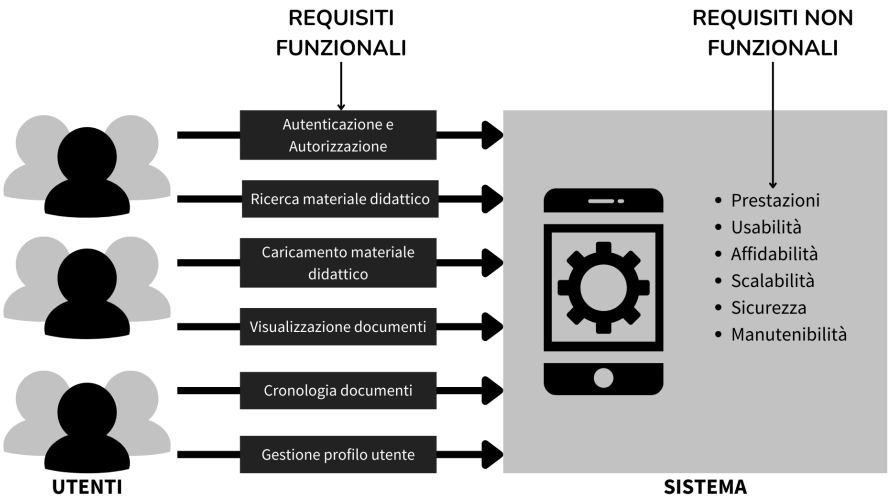


Figura 2.1: Confronto tra requisiti di tipo funzionale e non funzionale

2.4 Requisiti di sistema

Android è un software *open-source* per dispositivi mobili che comprende un sistema operativo, un *middleware* e applicazioni critiche. Da quando la piattaforma software è stata lanciata, ha subito numerosi miglioramenti in varie dimensioni, come le caratteristiche e l'hardware compatibile, per migliorare l'esperienza degli utenti. Inoltre, la piattaforma software si è estesa anche ad altri tipi di dispositivi oltre a quelli mobili a cui si rivolgeva quando è entrata nel mercato lanciando una piattaforma mobile. Basata sul modello architettonico del *kernel* Linux, la piattaforma Android è composta da diversi livelli che forniscono uno *stack* software completo [9].

Nel corso degli anni, sono state rilasciate numerose versioni di Android, ognuna con miglioramenti e nuove funzionalità. Di conseguenza, i requisiti di sistema per una determinata applicazione possono variare significativamente a seconda della versione di Android per la quale è stata sviluppata. Questo include requisiti sia hardware, come la quantità di RAM e la potenza del processore, sia software, come la necessità di specifiche API e compatibilità con versioni aggiornate dell'Android SDK.

2.4.1 Requisiti Software

Dal punto di vista software, è fondamentale disporre di driver compatibili per il *kernel* Linux, oltre a una versione aggiornata di Android SDK per lo sviluppo delle applicazioni. L'applicazione UniFolder è stata progettata per essere eseguita su sistemi operativi Android con un livello SDK pari o superiore a 24. La versione Android corrispondente viene denominata Nougat, è stata rilasciata nella seconda metà del 2016, ed ha costituito un grande passo in avanti verso i moderni sistemi operativi mobili; tra le molte novità si evidenziano il supporto *multi-window*, la modalità *Data Saver*, il compilatore JIT (che rende l'installazione delle app più veloce del 75%), supporto *picture-in-picture* ed avvisi sull'utilizzo della batteria [10].
Ad oggi, questo livello minimo SDK è supportato da più del 97% dei dispositivi Android, come testimoniato dalla successiva illustrazione.

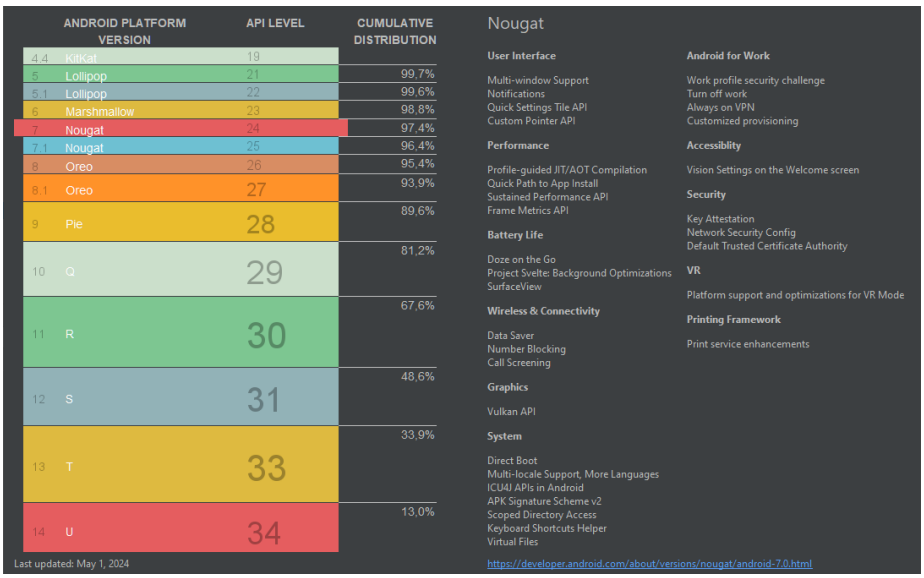


Figura 2.2: Distribuzione della versione Nougat e caratteristiche tecniche

2.4.2 Requisiti Hardware

Dal punto di vista hardware, l'applicazione UniFolder non necessità esplicitamente di alcuna componente dedicata. Gli unici vincoli fisici sono dettati dalla versione Android minima richiesta per il funzionamento dell'app.

Per supportare Android Nougat, infatti, gli smartphone devono soddisfare requisiti minimi di memoria e connettività. I dispositivi devono avere almeno 3 GB di memoria non volatile per i dati delle applicazioni. La memoria disponibile per il *kernel* e lo spazio utente (escludendo quella dedicata ai singoli componenti hardware) deve essere almeno di 512 MB o 896 MB per dispositivi a 32-bit, variabili rispetto alle dimensioni in dpi dello schermo, mentre per i dispositivi a 64-bit, i requisiti di memoria variano da 816 MB a 1824 MB.
Per quanto riguarda le funzionalità di rete, i dispositivi devono supportare almeno uno standard di rete dati con capacità di 200 Kbit/sec o superiore, come EDGE, HSPA, EV-DO, 802.11g, Ethernet o Bluetooth PAN. Devono inoltre includere uno *stack* di rete IPv6 con supporto per la comunicazione *dual-stack* (IPv4 e IPv6) su Wi-Fi, e per dispositivi con Ethernet. La connettività IPv6 deve essere mantenuta anche in modalità di risparmio energetico (*doze mode*) per garantire una comunicazione affidabile [11].

2.5 Requisiti di interfaccia utente

Il design dell'applicazione UniFolder vuole essere concepito per offrire un'esperienza utente moderna, intuitiva e coerente con le ultime tendenze del Material Design di Google. L'interfaccia deve essere progettata con un focus sulla semplicità e l'accessibilità, utilizzando una palette di colori chiari e scuri per adattarsi alle preferenze degli utenti in diverse condizioni di luce. L'applicazione suppor-

terà sia la modalità tema chiaro che la modalità *dark mode*, garantendo una visibilità ottimale e riducendo l'affaticamento visivo. La barra di stato e la barra di navigazione risulteranno personalizzate per integrare armoniosamente i colori principali del tema, fornendo un aspetto coeso e professionale. Le icone e le immagini scelte, infine, dovranno essere facilmente riconoscibili, contribuendo a una navigazione intuitiva e a una rapida accessibilità delle funzioni principali.

2.6 Dipendenze e vincoli

2.6.1 Dipendenze da terze parti

L'unico servizio esterno su cui poggia l'applicazione UniFolder è Firebase, di cui vengono sfruttati diversi moduli di servizi *back-end* necessari alla realizzazione delle funzionalità previste.

Firebase è una piattaforma per dispositivi mobili che consente di sviluppare rapidamente app di alta qualità, ampliare la base utenti e guadagnare di più. Firebase è costituito da funzionalità complementari che possono essere combinate in base alle diverse esigenze di progetto [12].

Firebase Authentication

La gestione dell'autenticazione tramite email e password è implementata in UniFolder facendo affidamento al servizio Firebase dedicato. Conoscere l'identità di un utente consente all'app di salvare in modo sicuro i dati dell'utente nel *cloud* e fornire la stessa esperienza personalizzata su tutti i dispositivi dell'utente. Firebase Authentication fornisce servizi di *back-end*, SDK di facile utilizzo e librerie dell'interfaccia utente già pronte per autenticare gli utenti [13].

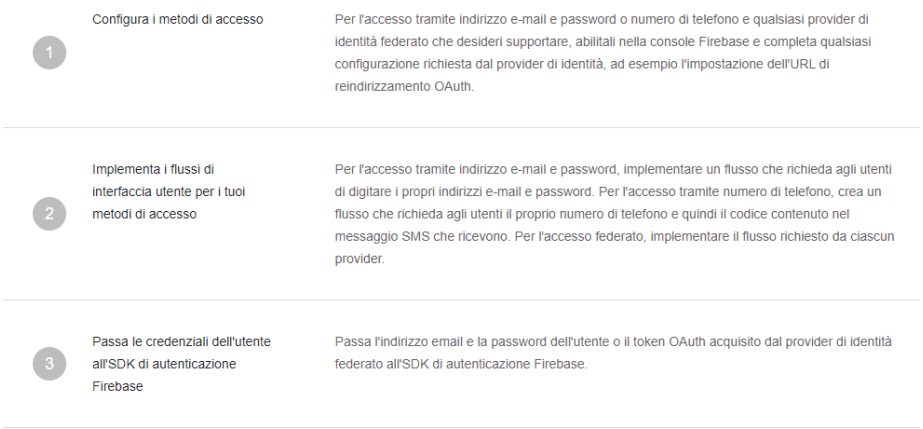


Figura 2.3: Funzionamento di Authentication

Firebase Realtime Database

Si tratta di un servizio di archiviazione e sincronizzazione dati basato su un database *cloud* NoSQL. I dati vengono archiviati come JSON e sincronizzati in

tempo reale su ogni client connesso [14].
A questo servizio sono affidati i dati relativi ai profili utente dell'applicazione UniFolder, come preferenze o elementi di personalizzazione.

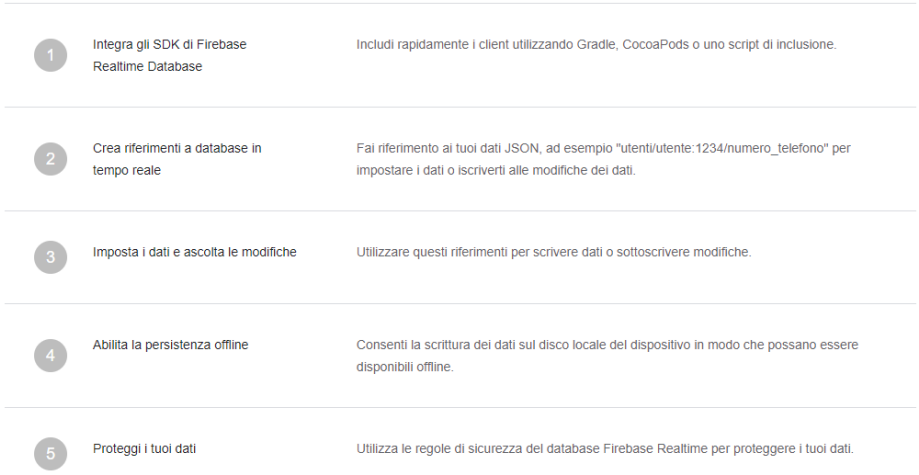


Figura 2.4: Funzionamento di Realtime Database

Firebase Cloud Firestore

Utilizza un database *cloud* NoSQL flessibile e scalabile, basato sull'infrastruttura Google Cloud, per archiviare e sincronizzare i dati per lo sviluppo lato client e server. Grazie a queste caratteristiche è in grado di ospitare l'intero insieme di documenti caricati dagli utenti di UniFolder, offrendo opzioni di indicizzazione per aumentare le performance rispetto a specifiche *query* di ricerca.
Come Firebase Realtime Database, mantiene i dati sincronizzati tra le app client tramite *listener* in tempo reale e offre supporto offline per dispositivi mobili e Web in modo da poter creare app reattive che funzionano indipendentemente dalla latenza di rete o dalla connettività Internet. Cloud Firestore offre inoltre un'integrazione perfetta con altri prodotti Firebase e Google Cloud, incluso Cloud Storage [15].

Figura 2.5: Funzionamento di Cloud Firestore

Firebase Cloud Storage

Cloud Storage per Firebase è basato sull'infrastruttura Google Cloud, rappresenta un'alternativa veloce e sicura per l'archiviazione e l'accesso a contenuti degli / per gli utenti e, nel caso di UniFolder, viene sfruttato per la gestione di file in formato PDF. In particolare, vengono creati URL di download che vengono collegati ad un documento non appena questo viene istanziato nel FireStore. Gli SDK Firebase per Cloud Storage aggiungono sicurezza Google ai caricamenti e ai download di file, indipendentemente dalla qualità della rete [16].

Figura 2.6: Funzionamento di Cloud Storage

2.6.2 Vincoli tecnologici

I vincoli tecnologici rappresentano le limitazioni e le restrizioni imposte dalla tecnologia e dalla piattaforma su cui si basa l'applicazione. Questi vincoli devono essere considerati attentamente durante la fase di sviluppo per garantire che l'applicazione funzioni correttamente all'interno delle specifiche date.

- **Visualizzazione dei Documenti Interna all'App**: L'applicazione deve essere in grado di visualizzare i documenti direttamente al suo interno, senza ricorrere a lettori esterni. Questo vincolo garantisce una migliore esperienza utente, evitando interruzioni e garantendo che tutti i contenuti siano accessibili direttamente dall'app.
- **Supporto Esclusivo per Documenti PDF**: L'applicazione deve accettare solo documenti in formato PDF. Questo vincolo semplifica la gestione dei documenti, assicurando che tutti i file siano in un formato standard e ampiamente supportato, riducendo così il rischio di problemi di compatibilità e visualizzazione.

Questi vincoli influenzano direttamente la scelta delle tecnologie e delle librerie da utilizzare, nonché la progettazione dell'architettura dell'applicazione.

2.7 Casi d'uso

Considerare il punto di vista dell'utente nella definizione dei requisiti è cruciale perché garantisce che l'applicazione soddisfi effettivamente le esigenze e le aspettative degli utenti finali. Questo approccio aumenta la probabilità di successo del progetto, migliorando l'usabilità, l'adozione e la soddisfazione degli utenti, e riducendo il rischio di sviluppare funzionalità non necessarie o poco intuitive.

Spesso gli utenti, però, hanno difficoltà ad articolare i propri requisiti. A tal fine, si può ricorrere alla raccolta di informazioni sui compiti che gli utenti svolgono attualmente e su quelli che potrebbero voler svolgere. Questi compiti possono spesso essere rappresentati in casi d'uso che possono essere utilizzati per descrivere i requisiti visibili dei sistemi. Più specificamente, l'ingegnere dei requisiti può scegliere un particolare percorso attraverso un caso d'uso, uno scenario, al fine di comprendere meglio alcuni aspetti dell'utilizzo di un sistema [8].

2.7.1 Caso d'Uso 1: Registrazione di un Nuovo Utente

Come utente non registrato, voglio poter creare un nuovo account, in modo da poter accedere a tutte le funzionalità dell'app.

Scenario:

1. L'utente apre l'app UniFolder.
2. L'utente seleziona l'opzione "Registrati".
3. L'utente inserisce le informazioni richieste (*username*, email, password, ecc.).
4. L'utente conferma la registrazione.
5. L'utente accede all'app con le nuove credenziali.

2.7.2 Caso d'Uso 2: Login dell'Utente

Come utente registrato, voglio poter effettuare il login, in modo da accedere al mio account e alle funzionalità dell'app.

Scenario:

1. L'utente apre l'app UniFolder.
2. L'utente seleziona l'opzione "Login".
3. L'utente inserisce email e password.
4. L'utente conferma l'accesso.
5. L'app autentica le credenziali e accede al profilo dell'utente.

2.7.3 Caso d'Uso 3: Ricerca di Materiale Didattico

Come utente registrato, voglio poter cercare e visualizzare materiale didattico specifico, in modo da trovare risorse utili per i miei studi.

Scenario:

1. L'utente accede all'app UniFolder.
2. L'utente naviga alla sezione "Cerca Materiale".
3. L'utente inserisce i criteri di ricerca (parola chiave, corso, tipo di documento, ecc.).
4. L'utente avvia la ricerca.
5. L'app visualizza i risultati corrispondenti.
6. L'utente seleziona un risultato per visualizzarne i dettagli.

2.7.4 Caso d'Uso 4: Caricamento di Materiale Didattico

Come utente registrato, voglio poter caricare file didattici (documenti, presentazioni, ecc.), in modo da condividerli con altri utenti.

Scenario:

1. L'utente accede all'app UniFolder.
2. L'utente naviga alla sezione "Carica Materiale".
3. L'utente inserisce i dettagli del file (titolo, descrizione, corso, ecc.).
4. L'utente seleziona il file da caricare dal proprio dispositivo.
5. L'utente conferma il caricamento.
6. L'app salva il file sul server e aggiorna l'elenco dei materiali disponibili.

2.8 Conclusioni

L'analisi dei requisiti è stata una fase cruciale per garantire che l'applicazione Android risponda alle esigenze degli utenti e soddisfi tutti i criteri di funzionamento previsti. Innanzitutto sono stati identificati chiaramente gli obiettivi utente, assicurando che l'applicazione fornisca un'interfaccia intuitiva e funzionalità che migliorino l'esperienza dell'utente. I requisiti funzionali sono stati definiti per specificare in dettaglio le funzionalità principali dell'app, come la gestione dei documenti e le interazioni con il sistema, mentre i requisiti non funzionali hanno delineato aspetti cruciali come la performance, la sicurezza e l'usabilità, garantendo che l'applicazione sia robusta, efficiente e sicura.
Grazie alla considerazione dei requisiti di sistema, sia hardware che software, si può assicurare quali diverse configurazioni dei dispositivi Android siano compatibili con l'applicazione. Le dipendenze esterne e i vincoli tecnologici sono stati valutati per identificare eventuali limitazioni o interazioni con altri sistemi, garantendo che l'applicazione possa integrarsi efficacemente nel contesto tecnologico esistente. Infine, i casi d'uso sono stati sviluppati per rappresentare scenari reali in cui l'applicazione verrà utilizzata, fornendo una guida pratica per il design e lo sviluppo.

La totalità dei requisiti identificati fino a questo punto, incontrerà il suo effettivo sviluppo nel processo di implementazione; la solida analisi dei requisiti svolta per la realizzazione dell'applicazione UniFolder, infatti, ha permesso di creare un'implementazione rigorosa, descritta nel capitolo seguente, in quanto a funzionalità, intenzioni e cura della percezione dell'utente finale.

Capitolo 3

Descrizione dell'implementazione

La fase di implementazione di un'applicazione Android richiede un'attenta pianificazione e una rigorosa adesione alle *best practice* architetturali per garantire che l'applicazione sia solida, scalabile e manutenibile. Una tipica app Android comprende vari componenti, come Activity, Fragment, servizi, content provider, che devono essere dichiarati nel file manifest dell'app. Questo complesso ambiente operativo, caratterizzato da risorse limitate e dalla possibilità che i processi dell'app vengano terminati in qualsiasi momento, pone significative sfide nello sviluppo. Pertanto, una corretta gestione dello stato dell'app e dei dati diventa essenziale per evitare la perdita di dati e garantire un'esperienza utente fluida [17].

L'intento di questo capitolo è di descrivere in dettaglio l'implementazione dell'applicazione sviluppata UniFolder, evidenziando come le linee guida e le *best practice* descritte saranno applicate per superare le sfide tipiche dello sviluppo su questa piattaforma. Attraverso una dettagliata disamina dell'architettura adottata, dell'organizzazione del codice e delle tecniche di gestione delle dipendenze e del flusso di dati, verranno illustrate le scelte progettuali che hanno contribuito a realizzare un'app solida e ben strutturata.

3.1 Architettura del Sistema

3.1.1 Panoramica dell'architettura

Per affrontare le sfide dello sviluppo di applicazione in ambiante Android, è cruciale seguire principi architetturali come la *Separation of concerns* ("separazione delle responsabilità"), l'uso di modelli di dati indipendenti dai componenti dell'interfaccia utente (UI), la definizione di un'unica fonte di verità (SSOT) per i dati dell'app e la definizione di un *Unidirectional Data Flow*. Questi principi non solo migliorano la scalabilità e la solidità dell'app, ma ne facilitano anche il Testing.

Inoltre, l'architettura raccomandata da Google per le app Android prevede almeno due livelli:

- Il livello UI che mostra i dati dell'applicazione sullo schermo, ricevendoli dal layer dati. Ogni volta che i dati cambiano, a causa dell'interazione dell'utente o di un input esterno, l'interfaccia utente deve aggiornarsi per riflettere le modifiche. Il livello UI è costituito da due elementi:
  - UI Elements: elementi dell'interfaccia utente che mostrano i dati sullo schermo.
  - State holders: come le classi ViewModel, contengono i dati, li espongono all'interfaccia utente e gestiscono la logica.
- Il livello dati che contiene la logica di business, gestisce i dati dell'app e li espone. Il livello dati è costituito da:
  - Repositories: definite per ogni diverso tipo di dati gestiti nell'app, ha la responsabilità di esporre dei dati al resto dell'app, centralizzare le modifiche ai dati, risolvere conflitti tra più Data sources e contenere la logica di business.
  - Data sources: sono il tramite tra l'applicazione e il sistema per le operazioni sui dati. Una classe di questo genere deve lavorare con una sola sorgente dati, che può essere un file, una risorsa di rete o un database locale.

In alcuni casi, può essere utile introdurre un ulteriore livello di dominio per semplificare e riutilizzare le interazioni tra l'interfaccia utente e i livelli dati [17].

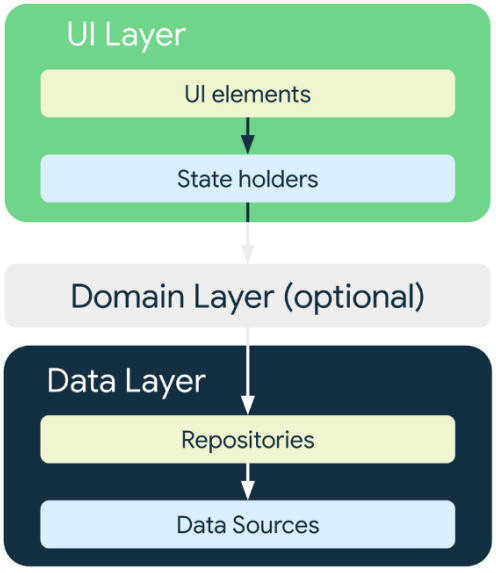


Figura 3.1: Il modello a livelli nell'architettura di un'applicazione Android

Al fine di descrivere più nello specifico l'architettura e le dipendenze delle effettive componenti dell'applicazione UniFolder, nella seguente sezione verranno presentati dei Class Diagram.

La notazione UML comprende i diagrammi delle classi per illustrare le classi, le interfacce e le relative associazioni. Essi sono utilizzati per la modellazione statica degli oggetti, ed in questa circostanza rappresentano il punto di vista software, ovvero l'effettiva realizzazione di queste classi all'interno del codice. Vengono tipicamente inclusi degli elementi di base, come [18]:

- **Classificatori**: in UML, un classificatore è un elemento di modello che descrive caratteristiche comportamentali e strutturali. I classificatori sono una generalizzazione di molti degli elementi di UML, comprese le classi, le interfacce, i casi d'uso e gli attori. Nei diagrammi delle classi, i due classificatori più comuni sono le classi e le interfacce.
- **Attributi delle classi**: mostrati mediante una notazione testuale e linee di associazione, omessi nella prossima sezione per questioni di compattezza della rappresentazione.
- **Estremità di associazioni della classe**: simboleggiano il punto di connessione tra una classe e un'associazione, indicando la relazione tra le due classi; sono mostrate mediante la notazione delle linee di associazione.

3.1.2 Architettura e componenti principali

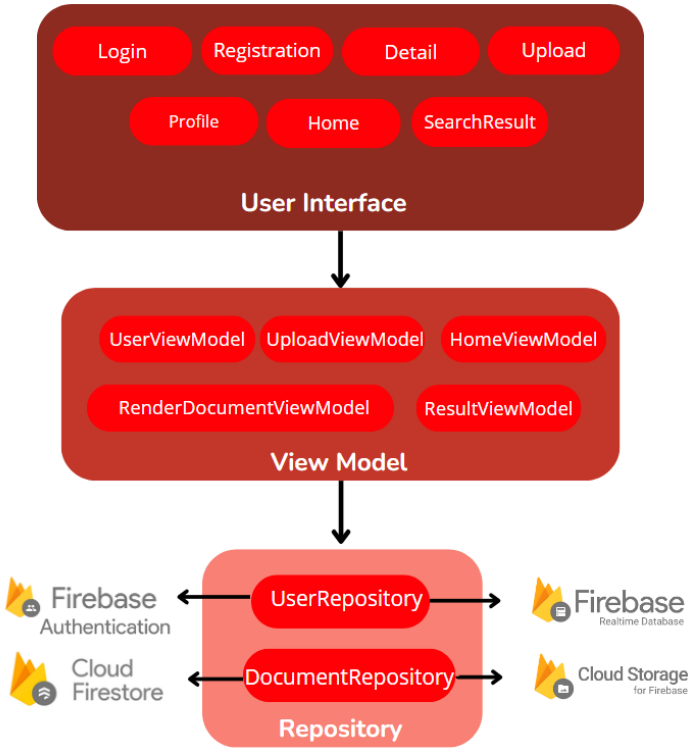


Figura 3.2: Diagramma architetturale dell'applicazione UniFolder

Layer UI

Il layer UI, costituito da Fragment ed Activity, costituisce la View nell'architettura MVVM ed i suoi componenti sono responsabili della presentazione dei dati e dell'interazione con l'utente:

- **Package Main**: responsabile dell'implementazione degli obiettivi utente, ovvero la ricerca/visualizzazione dei documenti ed il loro caricamento.
- **Package Welcome**: responsabile di autenticare o registrare gli utenti.

ViewModel

I ViewModel interagiscono con i rispettivi Repository e/o forniscono metodi di utilità per ottenere i dati da visualizzare nell'UI, utilizzando LiveData per rendere le View reattive. I ViewModel utilizzati sono i seguenti:

- **UserViewModel**: accede alla UserRepository per ottenere dati di utenti registrati all'applicazione.
- **ResultViewModel**: si interfaccia alla DocumentRepository per inviare *query* e, attraverso LiveData, offre i risultati ottenuti.
- **UploadViewModel**: si interfaccia alla DocumentRepository per l'*upload* di documenti, offrendo metodi di utilità invocati dal rispettivo Fragment.
- **RenderDocumentViewModel**: offre LiveData osservabili per restituire i render dei documenti, ottenuti grazie all'invocazione della DocumentRepository.
- **HomeViewModel**: utilizza LiveData per rendere la schermata Home reattiva alla ricezione delle informazioni che deve visualizzare, come ad esempio dettagli ed anteprime dei documenti nella cronologia, ottenuti dalla DocumentRepository, oppure informazioni sull'utente, grazie all'interfaccia verso la UserRepository, ed infine metodi di utilità per le schermate di *dialog*.

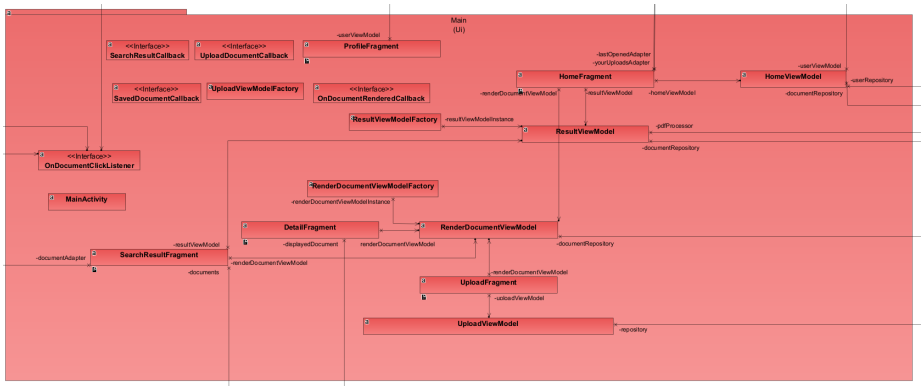


Figura 3.3: Class Diagram che mostra le dipendenze del *package* Main

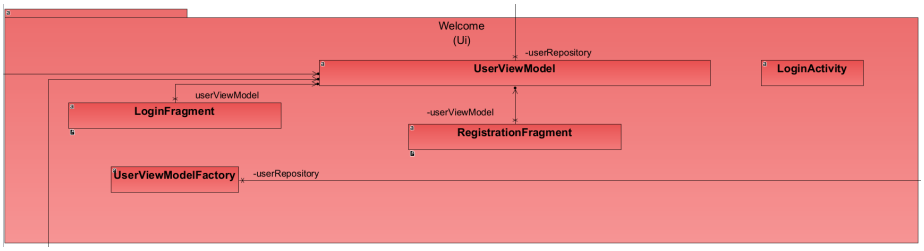


Figura 3.4: Class Diagram che mostra le dipendenze del *package* Welcome

Repository

Il Repository funge da intermediario tra il LocalDataSource e il RemoteDataSource, gestendo la logica di business e garantendo che i dati siano forniti in modo appropriato al ViewModel:

- **DocumentRepository**: si occupa di coordinare l'accesso ai dati, utilizzando sia il LocalDataSource che il RemoteDataSource secondo le necessità. Ad esempio, quando l'applicazione richiede un documento, il Repository prima cerca nel database locale e, se non è disponibile, lo recupera dal database remoto.
- **UserRepository**: si occupa delle operazioni relative alla gestione del profilo utente, così come del recupero degli user data.

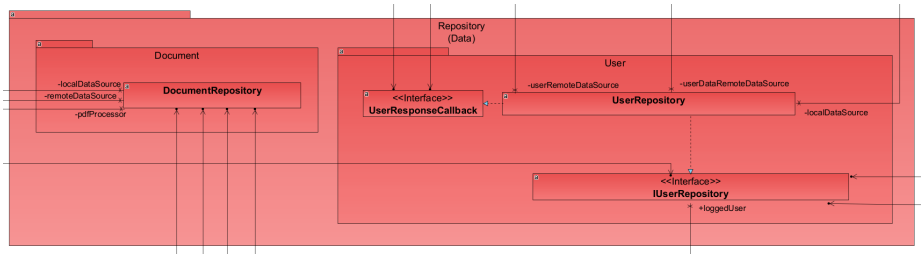


Figura 3.5: Class Diagram che mostra le dipendenze del *package* Repository

DataSource

I DataSource sono il tramite per le operazioni sui dati, ed ognuno di questi gestisce un'unica sorgente:

- **DocumentLocalDataSource**: gestisce l'accesso ai dati locali utilizzando Room per interagire con il database sul dispositivo dell'utente. Si occupa di operazioni come il recupero dei documenti salvati localmente o l'aggiunta di nuovi documenti.
- **RemoteDataSource**: gestisce l'accesso ai dati remoti utilizzando Firebase come *backend*. Si occupa di operazioni come il caricamento dei documenti su Firebase ed il recupero dei documenti da Firebase.

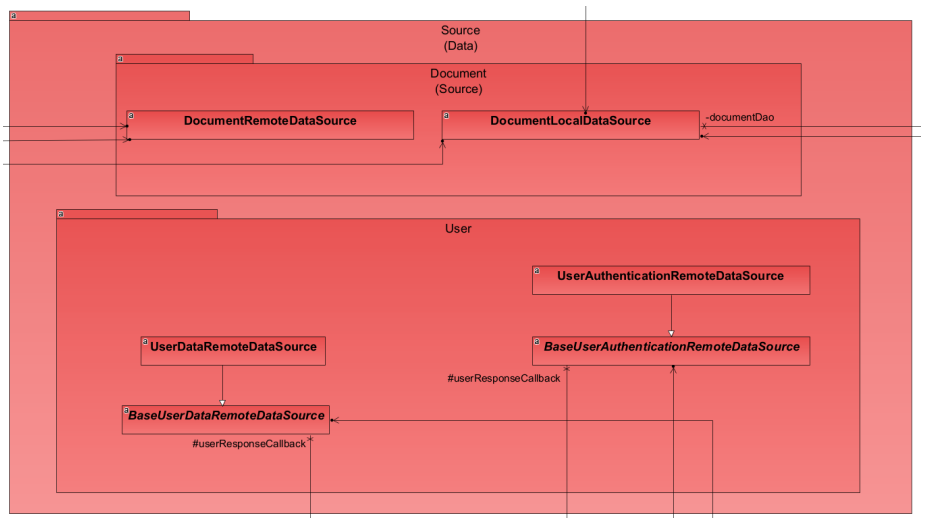


Figura 3.6: Class Diagram che mostra le dipendenze del *package* Source

Altre componenti

- **Model**: Il *package* model contiene le classi che rappresentano gli oggetti base dell'applicazione. Queste classi definiscono la struttura dei dati e forniscono i metodi necessari per la loro manipolazione, supportando le operazioni di creazione, lettura, aggiornamento e cancellazione (CRUD).

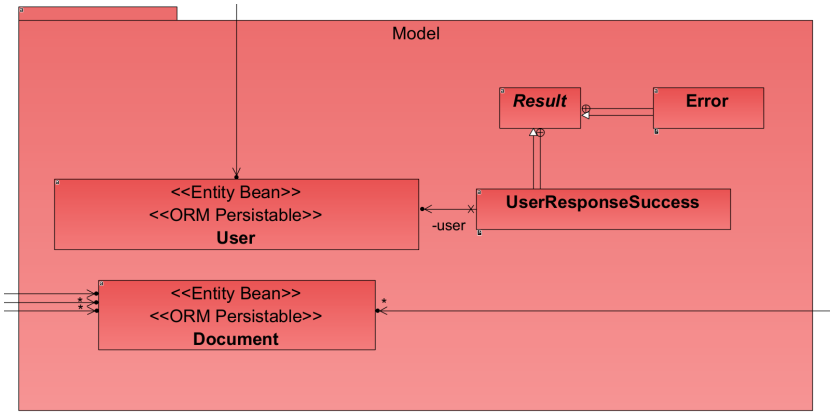


Figura 3.7: Class Diagram che mostra le dipendenze del *package* Model

- **Adapter**: Il *package* Adapter è responsabile della gestione e visualizzazione dei dati all'interno delle varie componenti dell'interfaccia utente, come liste e pagine. Le classi del *package* forniscono il *binding* tra i dati e le rispettive View, gestendo la visualizzazione dinamica e l'interazione con gli elementi della UI.

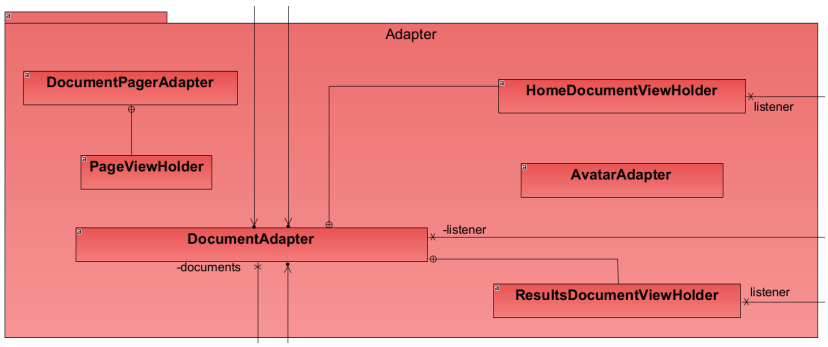


Figura 3.8: Class Diagram che mostra le dipendenze del *package* Adapter

- **Util**: Il *package* Util offre una serie di *utility* e classi di supporto per varie funzionalità comuni nell'applicazione. Include CourseUtil per la gestione dei corsi, LocalStorageManager per la gestione dello *storage* locale, Constants per le costanti globali, PdfProcessor per la gestione dei file PDF, e ServiceLocator per la localizzazione dei servizi, fornendo metodi e funzionalità riutilizzabili in diverse parti dell'app.

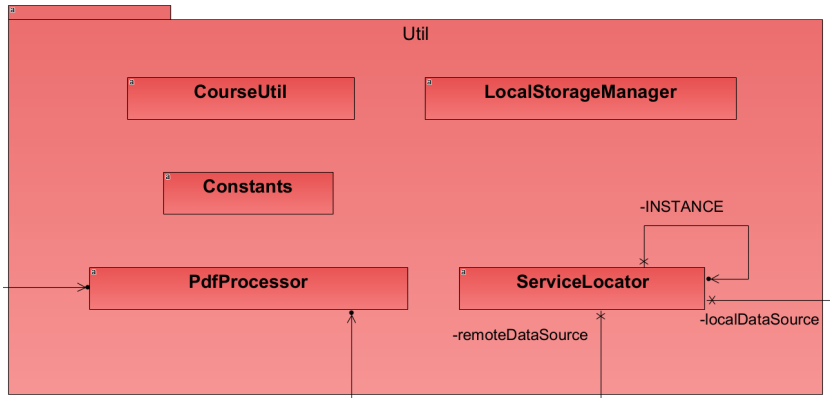


Figura 3.9: Class Diagram che mostra le dipendenze del *package* Util

- **Database**: Il *package* database è responsabile della gestione della persistenza dei dati nell'applicazione. Contiene DocumentDao, che definisce le operazioni di accesso ai dati (DAO) per i documenti, e DocumentDatabase, che fornisce l'implementazione del database locale, facilitando l'archiviazione, il recupero e la gestione dei dati persistenti.

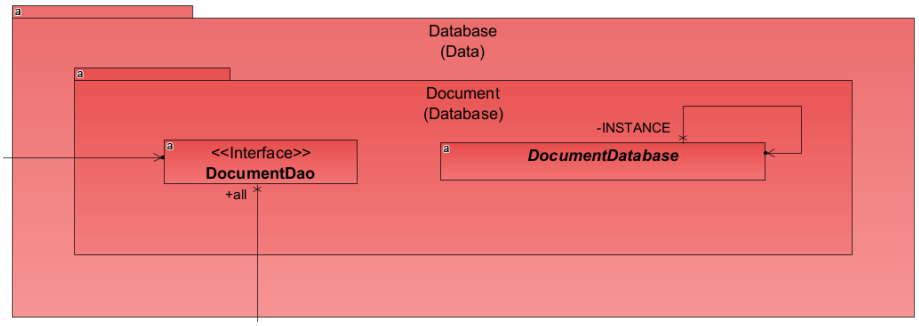


Figura 3.10: Class Diagram che mostra le dipendenze del *package* Database

3.2 Dettagli dell'implementazione e funzionalità

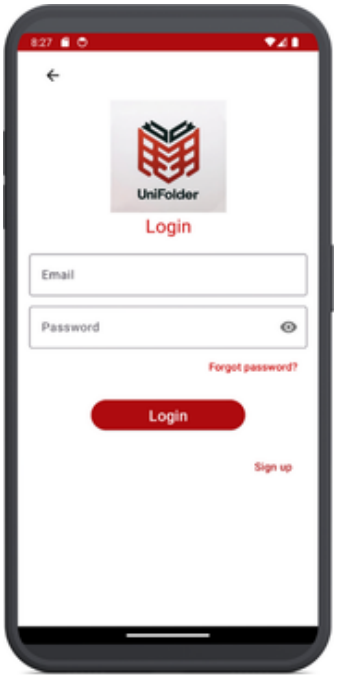
L'applicazione UniFolder presenta tre sezioni principali, accessibili facilmente dalla barra di navigazione. La schermata *Home* permette di avviare una ricerca di documenti attraverso *keyword* e filtri, allo stesso tempo presenta all'utente la cronologia degli ultimi documenti visualizzati e di quelli caricati.

La schermata di *Upload* vuole fornire un'interfaccia minimale ed intuitiva al fine di permettere all'utente di inserire dettagli utili alla catalogazione del contenuto che vuole caricare. A tal fine è stato introdotto un form che prevede l'inserimento di un titolo, la selezione del corso di studio attraverso una finestra di dialogo, la scelta di un tag relativo alla tipologia di documento e un *file picker* per l'inserimento di documenti in formato PDF dal proprio dispositivo.

La sezione *Profile*, infine, presenta all'utente le proprie informazioni, come lo *username*, l'indirizzo e-mail collegato ed un *avatar* personalizzabile. In più sono presenti uno *switch* per regolare il tema in modalità giorno/notte e pulsanti per il logout o la cancellazione del profilo.

Di seguito saranno analizzate nello specifico tutte le funzionalità implementate.

3.2.1 Login

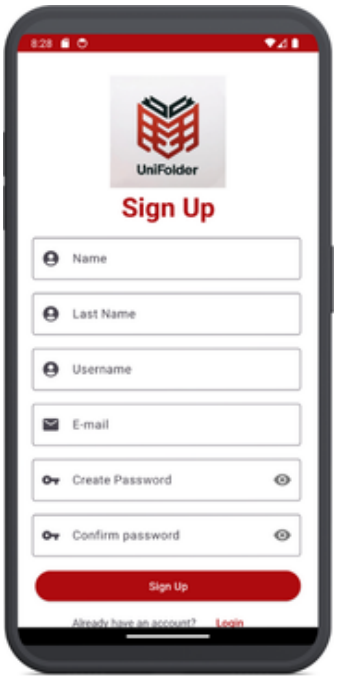


Al primo avvio della applicazione viene mostrata la schermata di login, la quale invita l'utente ad effettuare l'accesso per poter accedere a tutte le funzionalità, inserendo nei rispettivi campi le proprie credenziali (e-mail e password).

Se l'utente non dispone di un account, può registrarsi passando al corrispondente Fragment (si veda seguente), utilizzando il *button* "Sign up".

Dopo aver effettuato il primo accesso, riavviando l'applicazione verrà presentata la schermata Home con il profilo già collegato.

3.2.2 Signup



Da questa schermata, l'utente può compilare il form con le proprie informazioni in funzione di creare il proprio account, attraverso il quale potrà accedere a tutte le funzionalità.

In particolare, si richiede che l'indirizzo e-mail non sia già associato ad un altro account e che lo *username* scelto non sia già in uso.

Da questa schermata si può agevolmente passare a quella di login, nel caso in cui l'utente abbia già un account registrato.

3.2.3 Home - Ricerca

La schermata principale permette di effettuare una ricerca all'interno del database remoto, mediante la barra di ricerca nella parte superiore dallo schermo. Essa permette la ricerca per titolo ma anche con una serie di filtri accessibili dal *button* a lato: così facendo i risultati verranno filtrati per tag o corso di studi.

Inoltre, si presentano all'utente due RecyclerView che mostrano rispettivamente gli ultimi documenti aperti e i caricamenti effettuati: cliccando su un elemento si può visualizzare il documento desiderato con informazioni quali titolo, corso e tag. (si veda visualizzazione documento)

3.2.4 Risultati di ricerca

In questo particolare Fragment vengono mostrati i risultati della ricerca effettuata, presentati all'interno di una RecyclerView verticale. Il titolo della schermata permette all'utente di visualizzare la *keyword* cercata (se la ricerca la specificava).

In caso non fossero trovati risultati, un messaggio di errore avvisa l'utente consigliandogli di modificare i suoi parametri. A questo punto si può tornare alla schermata precedente cliccando sul pulsante "back" posizionato in alto.

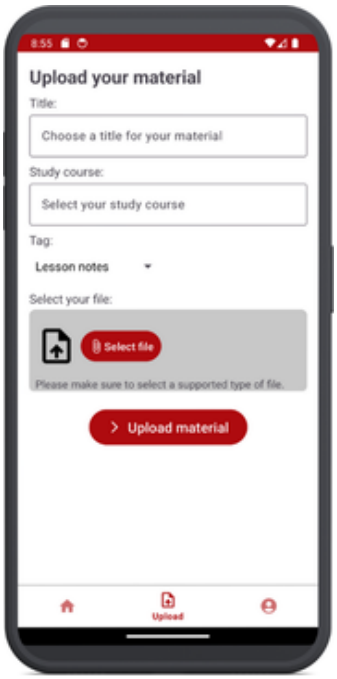
3.2.5 Visualizzazione documento

La schermata corrente, mostra i dettagli di un particolare documento selezionato: in alto sono specificati campi di titolo, corso e tag, mentre nella parte inferiore è presente il render del documento.

Il file è completamente sfogliabile in app; le pagine sono facilmente accessibili scorrendo orizzontalmente all'interno della View dedicata.

Al click del *button* in alto a destra la schermata si chiude lasciando spazio all'ultimo Fragment visualizzato in precedenza.

3.2.6 Caricamento file



La schermata di caricamento corrisponde alla seconda icona della barra di *bottom navigation*.

Da qui è possibile inserire i dettagli del documento da caricare e ovviamente scegliere comodamente il file (in formato PDF) da caricare dal proprio dispositivo, questo tramite il *button* "Select file". Una volta specificato un documento ne saranno visualizzati alcuni dettagli come nome del file e dimensione; sarà quindi possibile deselezionarlo e sceglierne uno diverso.

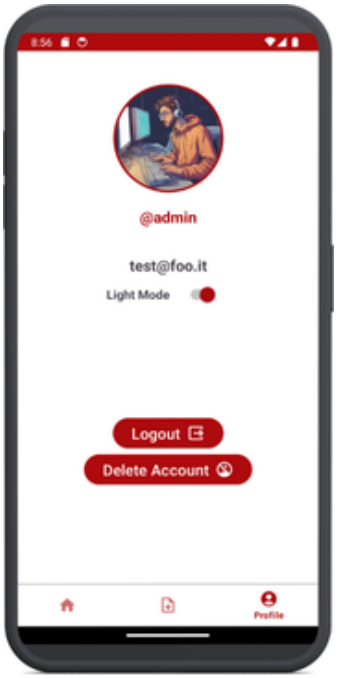
Al click del *button* "Upload material", se tutti i campi saranno stati specificati correttamente, il documento verrà creato e caricato online; al termine dell'operazione sarà visualizzato.

3.2.7 Profilo - Impostazioni

La schermata del profilo è accessibile grazie alla terza icona della barra di navigazione. All'interno di questa sono presentate tutte le informazioni disponibili dell'utente loggato.

Un'immagine *avatar* è presente per ciascun profilo, e può essere scelta tra l'insieme di quelle disponibili attraverso la finestra pop-up attivata al click su di essa.

Inoltre sono presenti comandi per regolare il tema dell'applicazione (*light mode* e *dark mode*) grazie allo *switch button*, oppure operazioni per la disconnessione dal profilo (*button* "Logout") e l'eliminazione del profilo stesso (*button* "Delete Account").



3.3 Design

UniFolder utilizza un layout grafico responsive, adattabile a una varietà di dispositivi mobili. La schermata principale presenta un'interfaccia pulita con una chiara suddivisione delle sezioni, come "Ultimi documenti aperti" e "I miei caricamenti", garantendo un rapido accesso ai contenuti più rilevanti per l'utente. Gli elementi interattivi, come i pulsanti di ricerca e filtro, sono posizionati in modo strategico per facilitare l'interazione, mentre l'uso di font leggibili e contrastanti assicura che le informazioni siano facilmente comprensibili. L'adozione del Material Design non solo migliora l'estetica dell'app, ma promuove anche la coerenza visiva e funzionale, offrendo agli utenti un'esperienza fluida e soddisfacente.

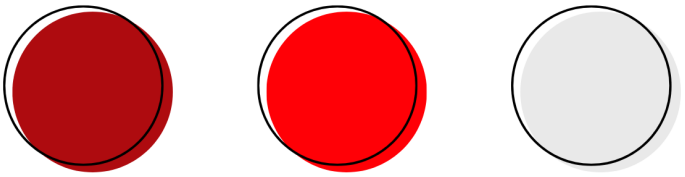


Figura 3.11: Palette principale di colori utilizzati

3.4 Sviluppi futuri

3.4.1 Nuovi obiettivi utente

Gli sviluppi futuri dell'applicazione UniFolder mirano ad ampliare le funzionalità esistenti per offrire un'esperienza utente sempre più completa e soddisfacente. Tra gli obiettivi principali sono considerati:

- **Gestione dei preferiti**: un utente può decidere di creare una propria collezione di documenti preferiti, così da poter accedere ai documenti più utili con maggior semplicità e coerenza;
- **Collaborazione e discussione**: integrare strumenti di collaborazione per permettere agli studenti di commentare, valutare, chiedere spiegazioni o suggerire miglioramenti per i materiali didattici in tempo reale;
- **Versioning documenti esistenti**: in linea con quanto descritto in precedenza, un documento dovrebbe poter essere aggiornato con eventuali correzioni, aggiornamenti e revisioni in modo da aumentare la qualità del materiale offerto.

3.4.2 Miglioramento features

UniFolder necessita di consolidare e migliorare le *features* già presenti in app, in modo da offrire una piattaforma più competitiva e funzionale all'importante *mission*. Pertanto, si pensa che possano essere migliorati:

- **Algoritmo di ricerca**: migliorare l'algoritmo di ricerca significa offrire maggiore rilevanza dei risultati, incontrando il bisogno dei singoli utenti di ottenere materiale mirato;
- **Catalogazione documenti**: integrare un sistema di tag personalizzabile, che renda ogni documento facilmente inquadrabile anche senza esplorarne il contenuto, oltre ad essere di supporto all'algoritmo di ricerca;
- **Esportazione e visualizzazione PDF**: allo stato attuale, UniFolder supporta esclusivamente una visualizzazione integrata dei file allegati; alcuni utenti potrebbero trovare utile avere a disposizione opzioni per la condivisione di link in app per rimandare più facilmente ad un certo documento, oltre a poter visualizzare questi ultimi con un *reader* esterno di sistema.

3.5 Conclusioni

Nella fase di descrizione dell'implementazione, si è delineata l'architettura del sistema sviluppato seguendo le specifiche Android per garantire un'applicazione robusta e scalabile. La panoramica dell'architettura ha incluso una dettagliata modellazione delle componenti principali del sistema, con l'uso di diagrammi delle classi e dei *package* che illustrano le interazioni e le dipendenze tra i vari elementi. Questo approccio metodico ha permesso di creare una struttura coerente e ben definita, facilitando la manutenzione e l'espansione futura dell'applicazione.

La descrizione dell'implementazione ha approfondito le caratteristiche principali dell'app, evidenziando le funzionalità offerte attraverso *screenshot* e spiegazioni dettagliate. Ogni *feature* è stata sviluppata per migliorare l'esperienza utente, tenendo conto delle *best practice* di design per Android. Sono stati definiti layout intuitivi e temi coerenti, che non solo migliorano l'estetica dell'app ma anche la sua usabilità, rendendo l'interazione con l'applicazione semplice e piacevole. Guardando al futuro, lo sviluppo dell'app prevede l'introduzione di nuovi obiettivi utente e il miglioramento delle funzionalità esistenti. L'obiettivo è continuare a migliorare l'applicazione per soddisfare al meglio le esigenze degli utenti, garantendo al contempo una piattaforma stabile e ricca di funzionalità innovative.

L'attenzione alla separazione delle responsabilità tra i componenti e la centralizzazione delle logiche di business hanno reso il codice non solo più manutenibile ma anche altamente testabile. L'ultimo aspetto pratico da trattare, proposto nel capitolo successivo, riguarda proprio il progetto di Testing, dove verranno descritte le metodologie utilizzate per garantire la qualità del software. L'implementazione di un'architettura modulare e l'adozione di pattern di design hanno facilitato l'automazione dei test e la verifica del codice, assicurando che ogni componente funzioni correttamente e che l'interazione tra i vari elementi avvenga senza problemi.

Capitolo 4

Progetto di Testing

Considerata la natura complessa delle applicazioni mobile e che queste richiedano un funzionamento in diversi contesti, è necessario avvalersi del meccanismo di Testing per poter verificare la correttezza, il comportamento funzionale e l'usabilità di UniFolder.

I test sono parte integrante del processo di sviluppo dell'appplicazione: eseguendo test su di essa in modo coerente, ne si può verificare la correttezza, il comportamento funzionale e l'usabilità prima di rilasciarla pubblicamente.
Il Testing nello sviluppo del software viene ampiamente utilizzato in ogni fase del ciclo di sviluppo ed in genere, viene dedicato al collaudo fino ad oltre il 50% del tempo di sviluppo. I test vengono solitamente eseguiti per i seguenti scopi:

- **Migliorare la qualità**: poiché i software possono essere utilizzati in applicazioni critiche, le conseguenze di un bug possono essere gravi. I bug possono causare perdite enormi, nei sistemi particolarmente critici hanno causato incidenti aerei, hanno permesso alle missioni dello Space Shuttle di andare in tilt, hanno bloccato le contrattazioni in borsa e altro ancora. Per qualità si intende la conformità ai requisiti di progettazione specificati. Essere corretti, il requisito minimo della qualità, significa funzionare come richiesto in circostanze specifiche. L'imperfezione della natura umana rende quasi impossibile correggere un programma moderatamente complesso al primo tentativo [19].

- **Per la verifica e la convalida (V&V)**: un altro scopo importante dei test è la verifica e la convalida (V&V), in questo caso i test possono servire come una sorta di "metrica". I tester possono fare affermazioni basate sull'interpretazione dei risultati dei test: il prodotto funziona in determinate situazioni, oppure non funziona, o si può confrontare la qualità tra diversi prodotti con le stesse specifiche, sulla base dei risultati dello stesso test.
  Non può essere testata la qualità direttamente, ma si può invece testare i fattori correlati per rendere visibile la qualità, ovvero: funzionalità, ingegneria e adattabilità. Questi tre gruppi di fattori possono essere considerati come dimensioni nello spazio della qualità del software ed ogni dimensione può essere scomposta nei fattori che la compongono e in considerazioni a livelli di dettaglio sempre più bassi. La tabella di seguito illustra alcune delle considerazioni sulla qualità più frequentemente citate.

Funzionalità (Qualità esterne)	**Ingegneria (Qualità interne)**	**Adattabilità (Qualità future)**
Correttezza	Efficienza	Flessibilità
Affidabilità	Testabilità	Riusabilità
Usabilità	Documentazione	Manutenibilità
Integrità	Struttura	

Tabella 4.1: Tipici fattori di qualità del Software

Un buon test prevede misure per tutti i fattori rilevanti, a seconda del contesto applicativo. Ad esempio, un qualsiasi sistema in cui sono in gioco vite umane deve dare estrema importanza all'affidabilità e all'integrità; nel tipico sistema aziendale, l'usabilità e la manutenibilità sono i fattori chiave. I test, per essere pienamente efficaci, devono essere orientati alla misurazione di ogni fattore rilevante, costringendo così la qualità a diventare tangibile e visibile.
Un progetto testabile è un progetto che può essere facilmente validato e mantenuto. Poiché i test sono uno sforzo rigoroso e richiedono tempi e costi significativi, la progettazione per la testabilità è anche una regola importante per lo sviluppo del software [19].

- **Per la stima dell'affidabilità**: l'affidabilità del software ha importanti relazioni con molti aspetti del software, tra cui la struttura e la quantità di test a cui è stato sottoposto. Sulla base di una stima della frequenza relativa di utilizzo dei vari input del programma, i test possono servire come metodo di campionamento statistico per ottenere dati sui guasti per la stima dell'affidabilità. Il collaudo del software può essere costoso, ma non collaudare il software è ancora più costoso, soprattutto se questo risulta essere critico nell'ambito in cui opera. Risolvere il problema del test del software non è più facile che risolvere il problema dell'*halting* di Turing: non si può mai essere sicuri che un software sia corretto [19].

4.1 Unit Testing

Gli Unit Test devono isolare un determinato componente da testare ed essere in grado di testarlo in modo ripetibile. Per questo motivo, i test unitari e gli oggetti Mock vengono solitamente affiancati. Si usano gli oggetti Mock per isolare l'unità dalle sue dipendenze, per monitorare le interazioni e anche per poter ripetere il test un numero qualsiasi di volte. Ad esempio, se il test cancella alcuni dati da un database, probabilmente non si vuole che i dati siano effettivamente cancellati e, quindi, non vengano trovati la volta successiva che il test viene eseguito. JUnit è lo standard di fatto per i test unitari su Android, si tratta di un semplice *framework open source* per automatizzare i test unitari [20].

Il metodo setUp()

Questo metodo viene chiamato per inizializzare la *fixture* (il test e lo stato del codice circostante). Sovrascrivendolo, si ha la possibilità di creare oggetti

e inizializzare campi che saranno utilizzati dai test. Vale la pena notare che questa configurazione avviene prima di ogni test [20].

Il metodo tearDown()

Questo metodo viene chiamato per finalizzare la *fixture*. Sovrascrivendolo, si possono rilasciare le risorse utilizzate dall'inizializzazione o dai test. Anche in questo caso, questo metodo viene invocato dopo ogni test. Ad esempio, si può rilasciare un database o chiudere una connessione di rete [20].

Fuori dal metodo di test

JUnit è progettato in modo tale che l'intero albero delle istanze di test sia costruito in un unico passaggio e che i test siano eseguiti in un secondo passaggio. Pertanto, il test *runner* mantiene forti riferimenti a tutte le istanze di test per tutta la durata dell'esecuzione del test. Ciò significa che, nel caso di test molto grandi e molto lunghi con molte istanze di test, nessuno dei test può essere sottoposto a *garbage collection* finché non viene eseguito l'intero test. Questo è particolarmente importante in Android e durante i test su dispositivi limitati, poiché alcuni test potrebbero fallire non a causa di un fallimento intrinseco, ma a causa della quantità di memoria necessaria per l'esecuzione dell'applicazione, oltre ai test che superano i limiti del dispositivo. Pertanto, se si allocano risorse esterne o limitate in un test, come Servizi o ContentProvider, si è responsabili della liberazione di tali risorse. L'impostazione esplicita di un oggetto a null nel metodo tearDown(), ad esempio, consente di liberarlo prima della fine dell'intera esecuzione del test [20].

All'interno del metodo di test

In JUnit 3, viene utilizzata l'introspezione per scovare i metodi di test, infatti tutti i metodi *public void* il cui nome inizia con test sono considerati come tali. Al contrario, JUnit 4 utilizza le annotazioni per scoprire i test [20].

Oggetti Mock

Come anticipato, gli oggetti Mock sono oggetti simulati utilizzati al posto degli oggetti di dominio reali, per consentire di testare le unità in modo isolato. In genere, questo viene fatto per verificare che vengano invocati i metodi corretti, ma possono anche essere di grande aiuto per isolare i test dal codice circostante ed essere in grado di eseguire i test in modo indipendente e garantire la riproducibilità.

I Mock non sono vere e proprie implementazioni, ma degli *stub*: l'idea è di estendere una di queste classi per creare un vero oggetto Mock e sovrascrivere i metodi che si desidera implementare. Tutti i metodi non sovrascritti verranno lanciati come UnsupportedOperationException [20].

4.1.1 Implementazione Unit Tests

Seguendo le *best practice*, sono stati implementati test per le seguenti unità:

ViewModels

- **HomeViewModelUnitTest**:

  Setup
  Nel metodo setUp(), viene configurato l'ambiente di test, inizializzando il contesto dell'applicazione e creando un'istanza del ViewModel.

  Test dei Metodi
  testFindIndex_Ok()

  - Funzione: Verifica che il metodo findIndex() di HomeViewModel ritorni l'indice corretto di un elemento in un array di stringhe.
  - Given: Un array di stringhe opts con valori predefiniti.
  - When: Invoca il metodo findIndex() su opts con vari valori (esistenti e non).
  - Then: Verifica che il metodo ritorni i corretti valori per le posizioni esistenti e -1 per valori non.

- **UploadViewModelUnitTest**:

  Setup
  Nel metodo setUp(), viene configurato l'ambiente di test, inizializzando il contesto dell'applicazione e creando un'istanza del ViewModel.

  Test dei Metodi
  testGetDocumentNameFromUri_Ok()

  - Funzione: Testa il metodo getDocumentNameFromUri() per verificare che recuperi correttamente il nome del documento da un URI.
  - Given: Un URI, un ContentResolver e un Cursor Mock.
  - When: Viene chiamato il metodo getDocumentNameFromUri().
  - Then: Verifica che il risultato sia il nome del file atteso.

  testGetDocumentSize_Ok()

  - Funzione: Testa il metodo getDocumentSize() per verificare che recuperi correttamente la dimensione del documento da un URI.
  - Given: Un URI, un ContentResolver e un ParcelFileDescriptor Mock.
  - When: Viene chiamato il metodo getDocumentSize().
  - Then: Verifica che la dimensione restituita sia corretta.

  testGetFilePathFromUri_Ok()

  - Funzione: Testa il metodo getFilePathFromUri() per verificare che recuperi correttamente il percorso del file da un URI.
  - Given: Un URI, un ContentResolver e un Cursor Mock.
  - When: Viene chiamato il metodo getFilePathFromUri().
  - Then: Verifica che il percorso del file restituito sia corretto.

testGetDocumentCreationDate_Ok()

- Funzione: Testa il metodo getDocumentCreationDate() per verificare che recuperi correttamente la data di creazione del documento da un URI.
- Given: Un URI, un ContentResolver e un ParcelFileDescriptor Mock.
- When: Viene chiamato il metodo getDocumentCreationDate().
- Then: Verifica che la data di creazione restituita sia corretta o un valore predefinito.

testGetFileSizeString_Ok()

- Funzione: Testa il metodo getFileSizeString() per verificare che converta correttamente le dimensioni del file in una stringa leggibile.
- Given: Diversi valori di dimensioni in byte.
- When: Viene chiamato il metodo getFileSizeString().
- Then: Verifica che le stringhe restituite corrispondano ai formati corretti (es. "0 B", "1.00 KB", "1.00 MB").

testCheckInputValuesAndUpload_Ok()

- Funzione: Testa il metodo checkInputValuesAndUpload() per verificare che accetti valori di input validi e avvii il caricamento.
- Given: Valori di input validi, un URI Mock e un *callback* Mock.
- When: Viene chiamato il metodo checkInputValuesAndUpload().
- Then: Verifica che il metodo restituisca *true*, indicando che il caricamento può procedere.

testCheckInputValuesAndUpload_Errors()

- Funzione: Testa il metodo checkInputValuesAndUpload() per verificare che gestisca correttamente valori di input non validi.
- Given: Valori di input non validi, un URI Mock e un *callback* Mock.
- When: Viene chiamato il metodo checkInputValuesAndUpload().
- Then: Verifica che il metodo restituisca false, indicando che il caricamento non può procedere a causa di input non validi.

- **RenderDocumentViewModelUnitTest**:

Setup
Nel metodo setUp(), viene configurato l'ambiente di test, disabilitando la registrazione dei Log, creando un Mock della Repository, inizializzando il ViewModel e configurando ArchTaskExecutor per eseguire tutte le operazioni sul Thread principale, simulando un ambiente di esecuzione sincrona per facilitare i test.

Test dei Metodi
testRenderDocument_Ok()

- Funzione: Verifica che il metodo renderDocument del RenderDocumentViewModel interagisca correttamente con il DocumentRepository e aggiorni i LiveData associati.
- Given: Un documento, un contesto Mock e una lista di bitmap Mock sono preparati e il Repository è configurato per restituire questi dati tramite un *callback*.
- When:Il metodo renderDocument del RenderDocumentViewModel viene chiamato con il documento e il contesto Mock.
- Then: Il metodo renderDocument del Repository è verificato come chiamato correttamente e i LiveData del ViewModel sono confermati come aggiornati con i valori attesi.

Cleanup
Nel metodo cleanUp(), si ripristina l'implementazione predefinita di ArchTaskExecutor, garantendo che le modifiche effettuate durante i test non influenzino altri test.

- **ResultViewModelUnitTest**:

Setup
Nel metodo setUp(), viene configurato l'ambiente di test, disabilitando la registrazione dei Log, creando un Mock della Repository, inizializzando il ViewModel e configurando ArchTaskExecutor per eseguire tutte le operazioni sul Thread principale, simulando un ambiente di esecuzione sincrona per facilitare i test.

Test dei Metodi
testSearchDocuments_Ok()

- Funzione: Questo test verifica che il ResultViewModel possa eseguire correttamente una ricerca di documenti basata solo su una *query* di ricerca (titolo del documento) e che aggiorni i LiveData con i risultati della ricerca.
- Given: Una *query* di ricerca e una lista di documenti Mock.
- When: Il metodo searchDocuments del ResultViewModel viene chiamato con la *query*.
- Then: I risultati della ricerca nel LiveData vengono verificati e il metodo di ricerca del Repository viene controllato come chiamato correttamente.

testSearchDocumentsWithTag_Ok()

- Funzione: Questo test verifica che il ResultViewModel possa eseguire una ricerca di documenti basata su una *query* di ricerca e un tag, e che aggiorni correttamente i LiveData con i risultati della ricerca.
- Given: Una *query* di ricerca, un tag e una lista di documenti Mock.
- When: Il metodo searchDocuments del ResultViewModel viene chiamato con la *query* e il tag.

– Then: I risultati della ricerca nel LiveData vengono verificati e il metodo di ricerca con filtro del Repository viene controllato come chiamato correttamente.

testSearchDocumentsWithTagAndCourse_Ok()

– Funzione: Questo test verifica che il ResultViewModel possa eseguire una ricerca di documenti basata su una *query* di ricerca, un tag e un corso, e che aggiorni correttamente i LiveData con i risultati della ricerca.

– Given: Una *query* di ricerca, un tag, un corso e una lista di documenti Mock.

– When: Il metodo searchDocuments del ResultViewModel viene chiamato con la *query*, il tag e il corso.

– Then: I risultati della ricerca nel LiveData vengono verificati e il metodo di ricerca con filtro e corso del Repository viene controllato come chiamato correttamente.

Cleanup
Nel metodo cleanUp(), si ripristina l'implementazione predefinita di ArchTaskExecutor, garantendo che le modifiche effettuate durante i test non influenzino altri test.

Data Layer

Richiede in particolare il test delle Repository, la maggior parte del livello dati dovrebbe essere indipendente dalla piattaforma.
I test Double sostituiscono i database e le fonti dati remote con oggetti Fake o Mock [21]:

- **DocumentRepositoryUnitTest**:

  Setup
  Nel metodo setUp(), viene configurato l'ambiente di test:

  – Mock: DocumentRemoteDataSource e DocumentLocalDataSource vengono mockati.

  – Spy: DocumentRepository viene inizializzato come uno Spy per poter monitorare le chiamate ai suoi metodi.

  – Context: Viene recuperato il contesto dell'applicazione.

  Test dei Metodi
  testSearchDocumentByTitle_Ok()

  – Funzione: Verifica che la ricerca di documenti per titolo funzioni correttamente.

  – Given: Una *query* di ricerca e una lista Mock di documenti.

  – When: Viene invocato il metodo searchDocumentByTitle() sul Repository.

- Then: Si verifica che il Data Source remoto venga chiamato correttamente e che il *callback* riceva i documenti mockati.

testSearchDocumentByFilter_Ok()

- Funzione: Verifica la ricerca di documenti per corso e tag.
- Given: Un corso e un tag di ricerca, e una lista Mock di documenti.
- When: Viene invocato il metodo searchDocumentByFilter() sul Repository.
- Then: Si verifica che il Data Source remoto venga chiamato correttamente e che il *callback* riceva i documenti mockati.

testSearchDocumentByTitleAndFilter_Ok()

- Funzione: Verifica la ricerca di documenti per titolo, corso e tag.
- Given: Una *query* di ricerca, un corso, un tag, e una lista Mock di documenti.
- When: Viene invocato il metodo searchDocumentByTitleAndFilter() sul Repository.
- Then: Si verifica che il Data Source remoto venga chiamato correttamente e che il *callback* riceva i documenti mockati.

testUploadDocument_Ok()

- Funzione: Verifica il caricamento dei documenti.
- Given: Un documento mockato e un *callback* mockato.
- When: Viene invocato il metodo uploadDocument() sul Repository.
- Then: Si verifica che il metodo di *upload* del Data Source remoto venga chiamato e che il documento venga salvato localmente.

testRenderDocument()

- Funzione: Verifica il *rendering* dei documenti.
- Given: Un documento mockato, un contesto mockato e un *callback* mockato.
- When: Viene invocato il metodo renderDocument() sul Repository.
- Then: Si verifica che il metodo getDocumentByIdAsync() venga chiamato e che il *callback* riceva il documento mockato e le bitmap mockate.

Classi di utilità

- **LocalStorageManagerUnitTest**:

  Setup
  Il metodo setUp() inizializza il contesto per i test.

  Test dei Metodi
  testSaveFileLocally()

- Funzione: Verifica che il metodo saveFileLocally() salvi correttamente un file in una directory specificata nel contesto Android e restituisca il percorso assoluto del file salvato.
- Given: Imposta il nome e il contenuto del file, crea un InputStream, mocka il metodo getExternalFilesDir() per restituire una directory temporanea e stubba il costruttore di FileOutputStream.
- When: Invoca il metodo saveFileLocally() con il contesto mockato, l'*input stream* e il nome del file.
- Then: Verifica che il percorso restituito dal metodo corrisponda al percorso assoluto del file salvato nella directory temporanea.

4.2 Integration Testing

I test di integrazione sono progettati per verificare il modo in cui i singoli componenti lavorano insieme. I moduli che sono stati testati indipendentemente vengono ora combinati insieme per testare l'integrazione. Di solito, le attività Android richiedono una certa integrazione con l'infrastruttura di sistema per poter essere eseguite. Hanno bisogno del ciclo di vita dell'attività fornito dall'ActivityManager e dell'accesso alle risorse, al filesystem e ai database. Gli stessi criteri si applicano ad altri componenti Android, come i servizi o i ContentProvider, che devono interagire con altre parti del sistema per svolgere il loro compito. In tutti questi casi, esistono classi di test specializzate fornite dal *framework* di Testing Android che facilitano la creazione di test per questi componenti [20].

4.3 UI Testing

L'implementazione di test UI rappresenta una strategia efficace per verificare il comportamento complessivo dell'applicazione in un ambiente che simula l'interazione reale dell'utente. I test UI sono una forma di test di tipo Instrumented, vengono quindi eseguiti direttamente su un dispositivo, fisico oppure un emulatore, offrendo una valutazione più accurata rispetto ai test eseguiti in ambienti di sviluppo isolati, poiché considerano le specifiche del dispositivo e le condizioni operative reali.

Utilizzando strumenti come Espresso per Android, i test UI Instrumented permettono di automatizzare l'interazione con l'interfaccia dell'applicazione, simulando input dell'utente come click, *swipe* e inserimento di testo, e verificando che le risposte dell'applicazione siano corrette e coerenti. Questo approccio non solo verifica l'aspetto e la funzionalità dell'interfaccia utente, ma consente anche di sussumere i test di integrazione, garantendo che le diverse componenti dell'app lavorino insieme senza problemi. L'esecuzione di questi test su dispositivi reali o emulatori assicura che l'applicazione sia robusta e pronta per l'uso in vari contesti e configurazioni di hardware e software [22].

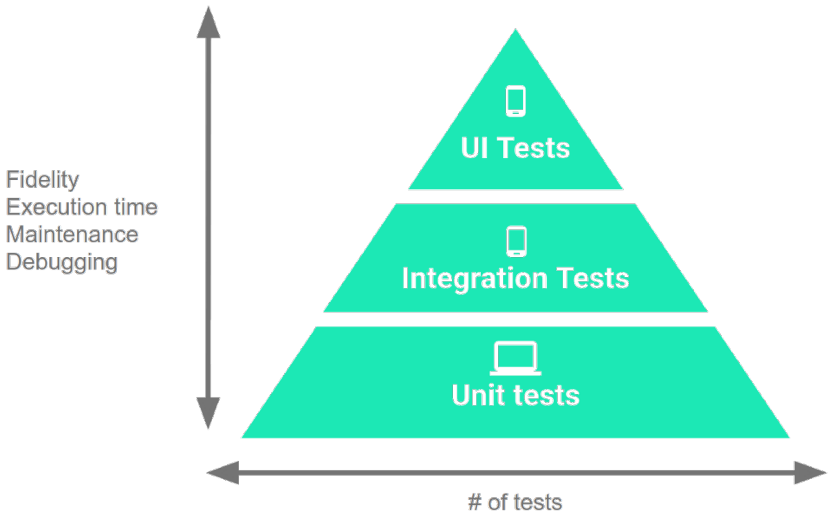


Figura 4.1: Relazione tra tipi di test

4.3.1 Implementazione UI Tests

Gli UI Tests sviluppati si ricollegano agli obiettivi utente principali:

- **SearchInstrumentedTest**: si occupa di testare innanzitutto la UI per l'inserimento dei parametri di ricerca (titolo e filtri), inoltre verifica la corrispondenza dei risultati visualizzati e l'eventuale schermata di risultati nulli; in più è testata la corretta visualizzazione dei dettagli di un documento al suo click dalla lista dei risultati.

  Setup
  Nel metodo setUp(), viene configurato l'ambiente di test, eseguendo il login all'applicazione simulando l'inserimento delle credenziali e il clic sul pulsante di login.

  Test dei Metodi
  testTitleSearchOkInteraction()

  - Funzione: Verifica che la ricerca per titolo funzioni correttamente mostrando i risultati pertinenti.
  - Given: Imposta un titolo di documento che dovrebbe produrre corrispondenze.
  - When: Esegue la ricerca inserendo il titolo e premendo il tasto Enter.
  - Then: Verifica che il primo elemento visualizzato nella RecyclerView contenga il titolo cercato.

  testTitleSearchNoResultInteraction()

  - Funzione: Verifica che la ricerca per titolo che non produce risultati mostri correttamente un messaggio di errore.
  - Given: Imposta un titolo di documento che non dovrebbe produrre corrispondenze.

- When: Esegue la ricerca inserendo il titolo e premendo il tasto Enter.
- Then: Verifica che venga visualizzato il layout di errore per nessun risultato.

testFilterSearchOkInteraction()

- Funzione: Verifica che la ricerca con filtri (tag, macroarea e corso) funzioni correttamente mostrando i risultati pertinenti.
- Given: Imposta i valori dei filtri (tag, macroarea e corso) e un titolo che dovrebbe produrre corrispondenze.
- When: Applica i filtri e esegue la ricerca inserendo un titolo.
- Then: Verifica che il primo elemento visualizzato nella RecyclerView contenga il titolo, il tag e il corso cercati.

testCombinedSearchOkInteraction()

- Funzione: Verifica che la ricerca combinata (titolo e filtri) funzioni correttamente mostrando i risultati pertinenti.
- Given: Imposta i valori dei filtri e un titolo di documento che dovrebbe produrre corrispondenze.
- When: Applica i filtri, esegue la ricerca inserendo il titolo e premendo il tasto Enter.
- Then: Verifica che il primo elemento visualizzato nella RecyclerView contenga il titolo, il tag e il corso cercati.

testSearchResultAndDetailOkInteraction()

- Funzione: Verifica che la visualizzazione dei dettagli di un documento funzioni correttamente dopo aver eseguito una ricerca combinata.
- Given: Imposta i valori dei filtri e un titolo di documento che dovrebbe produrre corrispondenze.
- When: Esegue la ricerca combinata, clicca sul primo risultato e verifica i dettagli del documento.
- Then: Verifica che i dettagli del documento visualizzato siano corretti.

- **UploadInstrumentedTest**: effettua il Testing della UI per quanto riguarda l'inserimento di un nuovo documento, in particolare verifica i messaggi di errore dovuti alla mancata o scorretta specificazione di alcuni campi.

Setup
Nel metodo setUp(), viene configurato l'ambiente di test, eseguendo il login all'applicazione simulando l'inserimento delle credenziali e il clic sul pulsante di login.

Test dei Metodi
testUploadWithTitleErrorInteraction()

- Funzione: Verifica che venga mostrato un messaggio di errore quando si tenta di caricare un documento senza inserire un titolo.
- Given: Nessun titolo inserito.
- When: Viene eseguito il caricamento del documento.
- Then: Verifica che venga visualizzato un messaggio di errore relativo al titolo.

testUploadWithCourseErrorInteraction()

- Funzione: Verifica che venga mostrato un messaggio di errore quando si tenta di caricare un documento senza selezionare un corso.
- Given: Nessun corso selezionato.
- When: Viene eseguito il caricamento del documento.
- Then: Verifica che venga visualizzato un messaggio di errore relativo al corso.

testUploadWithFileErrorInteraction()

- Funzione: Verifica che venga mostrato un messaggio di errore quando si tenta di caricare un documento senza allegare un file.
- Given: Nessun file allegato.
- When: Viene eseguito il caricamento del documento.
- Then: Verifica che venga visualizzato un messaggio di errore relativo al file.

4.4 Conclusioni

La fase di "Progetto di Testing" ha rappresentato un elemento cruciale nell'assicurare la qualità e la robustezza dell'applicazione sviluppata. Inizialmente questa ha compreso l'implementazione di test di unità (Unit Testing), che mirano a verificare il funzionamento corretto delle singole unità di codice, come metodi e classi. Questi test sono stati progettati per isolare ogni componente, consentendo di identificare rapidamente eventuali bug o comportamenti anomali. Utilizzando *framework* come JUnit, è stato possibile automatizzare questi test, garantendo che ogni unità del codice funzionasse come previsto e migliorando la manutenibilità del codice nel lungo termine.
Successivamente, i test di UI (UI Testing) sono stati implementati per verificare che l'interfaccia utente dell'applicazione sia intuitiva, funzionale e priva di errori. Questi test, eseguiti su dispositivi fisici o emulatori, simulano le interazioni dell'utente con l'app, garantendo che l'esperienza utente sia fluida e soddisfacente. Grazie a strumenti come Espresso si è realizzata in maniera automatica l'interazione con l'interfaccia, verificando che ogni elemento visivo e funzionale risponda correttamente agli input. I test UI non solo verificano l'aspetto e il comportamento della UI, ma sussumono anche aspetti dei test di integrazione, assicurando che l'intero sistema funzioni armoniosamente.

Capitolo 5

Utilizzo di ChatGPT

Come preannunciato nella sezione introduttiva, ChatGPT è stato impiegato in ognuna delle fasi di sviluppo per l'applicazione UniFolder. In generale, l'approccio assunto nei confronti dello strumento di supporto è stato costruito a partire dall'utilizzo di un'unica chat per le interazioni inerenti alla realizzazione dell'app, così da mantenere un contesto e un background solido, e dalla definizione di un prompt adeguato.

Definire un prompt significa creare un'istruzione o una serie di istruzioni specifiche che guidano un modello di linguaggio, come un ChatBot o un'intelligenza artificiale, a produrre una risposta desiderata. I prompt possono essere semplici, come una singola domanda, o complessi, includendo dettagli specifici che il modello deve considerare nella sua risposta. La definizione di un prompt è cruciale perché rappresenta il modo in cui gli utenti interagiscono con i modelli di linguaggio, influenzando direttamente la qualità e la pertinenza delle risposte generate [23].

Rispetto all'utilizzo del ChatBot, questo è stato sollecitato in virtù della definizione di implementazioni specifiche secondo una logica già definita e collaudata, ad esempio seguendo i principi della progettazione Android (le *best practice*), oppure per esplorare alternative differenti, sia da un punto di vista di modelli che di funzionamenti particolari, nonché per avere supporto nelle gestioni di dipendenze esterne.
I suggerimenti del ChatBot, a seconda di ogni caso, possono risultare consoni agli scopi presentati o fuorvianti, ed è il chiaro motivo per il quale ChatGPT viene presentato solamente come uno strumento di supporto al lavoro del programmatore; quest'ultimo possiede la capacità di rielaborare soluzioni, nozioni o suggerimenti presentatogli, includendo poi nel progetto effettivo i soli principi e il codice selezionati con il criterio di raggiungere gli obiettivi prestabiliti.

Nelle seguenti sezioni verranno approfondite le diverse questioni che hanno caratterizzato le interazioni con ChatGPT, per alcune delle fasi descritte fino ad ora.

5.1 Analisi dei requisiti

Durante la fase di "Analisi dei requisiti" per la realizzazione dell'applicazione UniFolder, ChatGPT è stato utilizzato per vari scopi cruciali. Di seguito sono riportati i principali modi in cui la chat è stata utilizzata:

- **Raccolta delle Esigenze degli Utenti**
  - Descrizione: La chat ha fornito supporto nell'identificare e raccogliere le esigenze specifiche degli utenti finali dell'applicazione. Questo è stato realizzato attraverso la simulazione di scenari d'uso e la formulazione di domande che aiutano a chiarire le funzionalità richieste.
  - Esempio: Discussioni su come gli studenti e i docenti vorrebbero caricare, cercare e visualizzare i documenti accademici.
- **Definizione delle Funzionalità Chiave**
  - Descrizione: Utilizzando la chat, è stato possibile esplorare e definire in dettaglio le funzionalità chiave dell'applicazione, come la gestione dei documenti, la ricerca avanzata, i filtri e le interazioni utente.
  - Esempio: Analisi delle funzionalità di ricerca per garantire che gli utenti possano trovare documenti rilevanti utilizzando titoli, tag e corsi.
- **Chiarimento dei Requisiti Tecnici**
  - Descrizione: La chat ha aiutato a chiarire i requisiti tecnici, incluso l'uso delle librerie e *framework* necessari per implementare le funzionalità desiderate.
  - Esempio: Discussioni sulle librerie di Espresso per i test di interfaccia utente e come implementare test di integrazione efficaci.
- **Consulenza e Best Practices**
  - Descrizione: La chat ha offerto consulenza sulle *best practices* per lo sviluppo di applicazioni mobili, inclusi suggerimenti su come gestire l'interazione con l'utente e l'implementazione di un'interfaccia utente intuitiva.
  - Esempio: Consigli su come implementare i pattern di design Material Design per migliorare l'esperienza utente.

5.1.1 Considerazioni sull'utilizzo

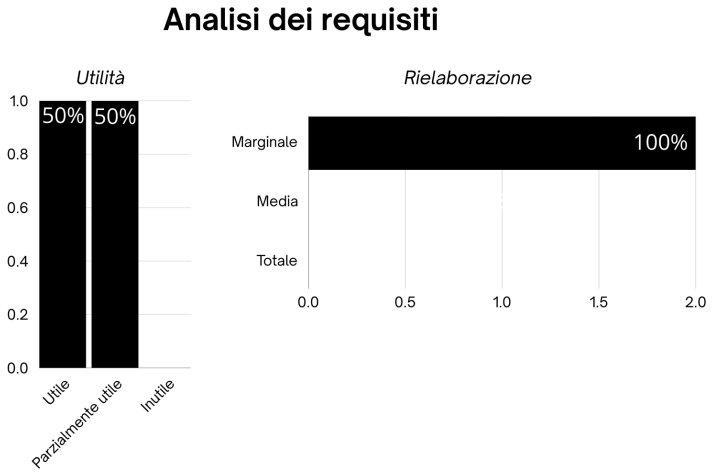


Figura 5.1: Statistiche delle interazioni con ChatGPT durante la fase di Analisi dei requisiti

Tra le fasi in esamina, questa fase presenta il minor numero di interazioni con ChatGPT registrate in totale, perlopiù avvenute per definire in maniera concisa obiettivi utente e requisiti dell'applicazione, nonché per analizzare i diversi campi di impiego di diverse applicazioni, prima della scelta di UniFolder. Tuttavia, si può notare come la valutazione delle risposte sia altamente sbilanciata a favore dell'utilità delle stesse e verso una rielaborazione minima dei loro contenuti; insieme a questo, il fatto che il numero medio di richieste per la generazione di nuove risposte sia minimo, è indice di una spiccata efficacia (almeno percepita) e completezza nella generazione di contenuti da parte del ChatBot in questa categoria.

Contributi dell'Utilizzo della Chat

- **Chiarezza e Dettaglio**: La chat ha aiutato a chiarire e dettagliare i requisiti in modo strutturato, riducendo ambiguità e migliorando la comprensione condivisa tra i membri del team.
- **Efficienza**: Ha permesso di ottenere risposte rapide e precise a domande specifiche, accelerando il processo di analisi dei requisiti.
- **Qualità**: Ha contribuito a identificare e risolvere potenziali problemi in fase di progettazione, migliorando la qualità complessiva del progetto.

5.1.2 Conclusione

L'utilizzo di questa chat durante la fase di "Analisi dei requisiti" ha giocato un ruolo fondamentale nell'assicurare che l'applicazione UniFolder fosse progettata in modo efficace e rispondesse alle esigenze degli utenti finali, integrando al

contempo *best practices* e standard di qualità elevati.
Nonostante il ristretto numero di interazioni registrate, si può certamente definire positivo il feedback fornito da ChatGPT in merito alla fase di "Analisi dei requisiti", in quanto a qualità e pertinenza delle risposte.

5.2 Descrizione dell'implementazione

Durante la fase di "Descrizione dell'implementazione" per la realizzazione dell'applicazione UniFolder, ChatGPT è stato utilizzato in diversi modi per supportare lo sviluppo del progetto. Di seguito sono riportati i principali modi in cui la chat è stata utilizzata durante questa fase:

- **Implementazione delle Funzionalità**
  - Descrizione: La chat è stata utilizzata per guidare l'implementazione delle funzionalità chiave, offrendo esempi di codice che seguisse le *best practices*.
  - Esempio: Esempi di implementazione per la gestione degli input utente nei campi di caricamento dei documenti e nei filtri di ricerca.
- **Integrazione con Servizi Esterni**
  - Descrizione: La chat ha aiutato a integrare l'applicazione con servizi esterni, come server *backend* e servizi di autenticazione.
  - Esempio: Supporto per l'implementazione del login tramite server *backend* e la gestione dei servizi Firebase stessi.
- **Refactoring del Codice**
  - Descrizione: La chat ha fornito suggerimenti per il Refactoring del codice per migliorare la manutenibilità e la leggibilità del codice.
  - Esempio: Suggerimenti su come suddividere grandi metodi in metodi più piccoli e modulari per migliorare la chiarezza e la riutilizzabilità del codice.
- **Gestione degli Errori e delle Eccezioni**
  - Descrizione: La chat ha fornito supporto nella gestione degli errori e delle eccezioni, suggerendo strategie per catturare e gestire correttamente gli errori in modo da migliorare la robustezza dell'applicazione.
  - Esempio: Consigli su come gestire le eccezioni durante il caricamento dei file e la visualizzazione degli errori appropriati agli utenti.

5.2.1 Considerazioni sull'utilizzo

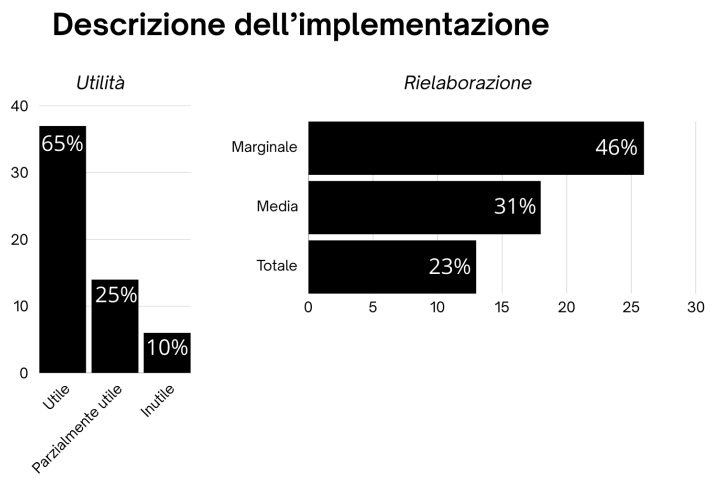


Figura 5.2: Statistiche delle interazioni con ChatGPT durante la fase di Descrizione dell'implementazione

La fase di "Descrizione dell'implementazione" ha visto un utilizzo piuttosto attivo di ChatGPT, utile ad esempio nella definizione di layout XML, ma anche di *coding* di parti di classi o metodi a partire da una specifica piuttosto completa. Rispetto a questa fase dello sviluppo, le interazioni con ChatGPT hanno registrato il più alto numero di feedback sia per la categoria "Utile" che per una rielaborazione dei contenuti "Marginale". La presenza di una buona percentuale di valutazioni differenti da quelle enunciate, può essere interpretata in una buona parte dei casi come la scelta del programmatore di fornire implementazioni (più o meno) differenti da quelle proposte, in quanto più consone agli scopi progettuali, o piuttosto talvolta come una vera e propria lacuna delle soluzioni offerte dal ChatBot, in quanto a funzionamento o utilizzo di librerie deprecate.

Contributi dell'Utilizzo della Chat

- **Implementazione Immediata**: Ha permesso di ottenere assistenza immediata su implementazioni specifiche, riducendo il tempo di *coding* necessario per implementare soluzioni anche note.
- **Miglioramento delle Competenze**: Ha contribuito a migliorare le competenze degli sviluppatori attraverso consigli e suggerimenti dettagliati.
- **Qualità del Codice**: Ha aiutato a migliorare la qualità del codice attraverso suggerimenti su *best practices* e strategie di Refactoring.

5.2.2 Conclusione

L'utilizzo di questa chat durante la fase di "Descrizione dell'implementazione" ha giocato un ruolo cruciale nel supportare lo sviluppo dell'applicazione UniFolder. Ha fornito soluzioni rapide ai problemi, migliorato le competenze del

team di sviluppo, e contribuito a garantire che il codice fosse di alta qualità e che l'applicazione rispondesse efficacemente ai requisiti definiti.
La qualità delle risposte generate da ChatGPT all'interno di questa fase, così come l'integrazione dei contenuti *as-they-are*, si possono ritenere di discreto livello, essendo risultati percentualmente in maggioranza all'interno delle interazioni registrate.

5.3 Progetto di Testing

Durante la fase di "Progetto di Testing" per la realizzazione dell'applicazione UniFolder, ChatGPT è stato utilizzato in vari modi per supportare e migliorare il processo di Testing. Ecco una descrizione dettagliata di come è stata utilizzata la chat:

- **Definizione della Strategia di Testing**
  - Descrizione: La chat è stata utilizzata per aiutare a definire una strategia di Testing completa, che includeva la scelta degli strumenti di Testing appropriati e la definizione dei casi di test.
  - Esempio: Discussioni sulle *best practices* per il Testing delle applicazioni Android e sulla configurazione di Espresso per i test di interfaccia utente.
- **Scrittura dei Test di Unità**
  - Descrizione: La chat ha fornito supporto nella scrittura di test di unità, aiutando a garantire che le diverse classi e metodi dell'applicazione funzionassero correttamente, grazie soprattutto alla definizione di oggetti Mock.
  - Esempio: Esempi di codice per test di unità che verificano la correttezza di classi ViewModel o Repository per la ricerca di documenti.
- **Implementazione dei Test di Interfaccia Utente**
  - Descrizione: La chat ha assistito nell'implementazione dei test di interfaccia utente utilizzando Espresso, fornendo indicazioni su come simulare le azioni degli utenti e verificare i risultati.
  - Esempio: Indicazioni dettagliate su come utilizzare Espresso per simulare il login dell'utente, eseguire ricerche e verificare che i risultati siano visualizzati correttamente.
- **Gestione delle Risorse di Testing**
  - Descrizione: La chat ha fornito consigli su come gestire le risorse necessarie per i test, come l'uso delle risorse di attesa (Idling Resources) per sincronizzare i test con lo stato dell'applicazione.
  - Esempio: Suggerimenti per assicurarsi che i test attendano il completamento delle operazioni asincrone.
- **Debugging dei Test**

– Descrizione: La chat è stata utilizzata per il debugging dei test, fornendo supporto per risolvere i problemi riscontrati durante l'esecuzione dei test.
– Esempio: Soluzioni ai problemi relativi alla visibilità degli elementi durante i test di interfaccia utente e alla sincronizzazione dei test con l'interfaccia utente dell'applicazione.

5.3.1 Considerazioni sull'utilizzo

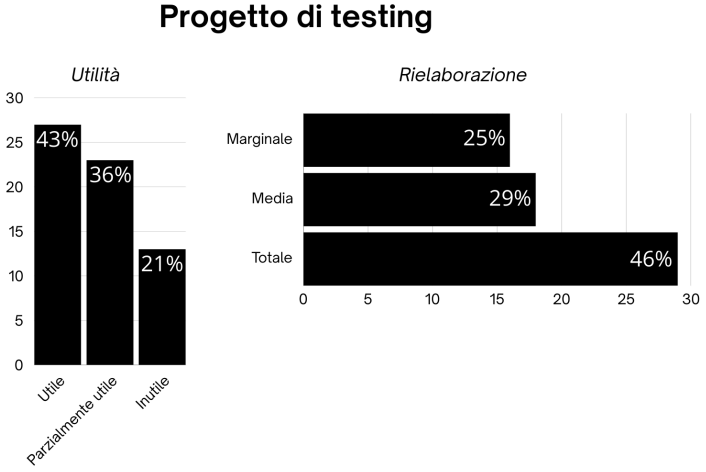


Figura 5.3: Statistiche delle interazioni con ChatGPT durante la fase di Progetto di Testing

Tra le categorie in esamina, la fase di "Progetto di Testing" ha fatto registrare il maggior numero di interazioni con ChatGPT, esplicitando fin da subito il maggiore supporto richiesto dal programmatore in queste circostanze. Infatti, lo strumento di ChatBot è stato interrogato rispetto alle diverse tecniche di Testing, in modo da scovare la più idonea, e soprattutto rispetto all'utilizzo di diverse librerie per l'implementazione, aspetto risultato essere particolarmente impegnativo.
In quanto ad utilità si può ritenere quantomeno soddisfacente il livello raggiunto dalle risposte generate, al contrario si nota una rielaborazione "Marginale" sostanzialmente solo per un caso su quattro, così come il più alto tasso registrato di rigenerazione dei responsi tra tutte le categorie.
Interpretando questi dati, è possibile dedurre che i contenuti generati dal ChatBot risultassero almeno concettualmente (in parte o del tutto) corretti, ma d'altro canto inadatti ad essere inclusi nel progetto *as-they-are*. Un'ulteriore considerazione riguarda le modalità di richiesta per la generazione: sebbene non vi siano dati numerici registrati rispetto a quest'ultimo fattore, si è potuto notare come il ChatBot stesso fosse più propenso a fornire soluzioni accettabili non appena riuscisse a fornire una prima risposta "in linea" con le aspettative ed il contesto; a questo punto, continuando a ragionare sulla stessa lunghezza d'onda si sono registrati una serie consecutiva di risposte di qualità.

Contributi dell'Utilizzo della Chat

- **Supporto Dettagliato**: Ha fornito supporto dettagliato e specifico per ogni fase del Testing, dai casi di test ai problemi di sincronizzazione.
- **Miglioramento della Qualità del Codice**: Ha contribuito a migliorare la qualità del codice attraverso test rigorosi e verifiche continue.
- **Copertura del Testing**: Ha migliorato la copertura di codice del Testing, garantendo che tutti gli aspetti dell'applicazione fossero testati accuratamente.

5.3.2 Conclusione

L'utilizzo di questa chat durante la fase di "Progetto di Testing" ha giocato un ruolo cruciale nel garantire che l'applicazione UniFolder fosse testata in modo completo e rigoroso. Ha fornito supporto per la definizione della strategia di Testing, la scrittura e l'implementazione dei test, la gestione delle risorse di Testing, il debugging e l'automazione dei test. Questo ha contribuito a garantire che l'applicazione fosse di alta qualità e rispondesse ai requisiti definiti, migliorando l'affidabilità e la robustezza dell'applicazione.
L'utilizzo di ChatGPT in questa fase è sicuramente risultato essere il più ostico, come dimostrato dai dati; nonostante ciò, si può affermare di aver raggiunto comunque un discreto livello di qualità per la fase di "Progetto di Testing", grazie all'esplorazione di più possibilità e tecniche, fino alla scoperta della modalità più consona allo sviluppo dei diversi tipi di test.

5.4 Conclusioni

L'analisi dettagliata delle diverse fasi del progetto UniFolder ha messo in luce un utilizzo variegato e strategico di ChatGPT, adattato alle specifiche esigenze del programmatore in ciascuna fase del ciclo di vita dello sviluppo software. Durante la fase di "Analisi dei requisiti", ChatGPT ha avuto un ruolo di rilievo nonostante il numero relativamente ridotto di interazioni, dimostrando un'alta efficacia nella generazione di contenuti utili e concisi. La fase di "Descrizione dell'implementazione" ha visto un maggiore impiego dello strumento, utilizzato per la definizione di layout XML e per la scrittura di codice a partire da specifiche dettagliate. Anche se le interazioni con ChatGPT in questa fase hanno mostrato una buona percentuale di feedback positivi, il team ha spesso optato per implementazioni personalizzate per meglio rispondere alle esigenze progettuali. Infine, durante la fase di "Progetto di Testing", ChatGPT ha registrato il maggior numero di interazioni, indicando un alto livello di supporto richiesto per la definizione e l'implementazione delle tecniche di Testing. Nonostante il tasso di rigenerazione delle risposte fosse il più alto, le soluzioni fornite si sono rivelate concettualmente utili, sebbene richiedessero ulteriori adattamenti per essere completamente integrate nel progetto.

In sintesi, l'utilizzo di ChatGPT nel progetto UniFolder ha evidenziato una notevole flessibilità e adattabilità dello strumento, che è stato impiegato con successo in diverse fasi del ciclo di vita del progetto. Le modalità di utilizzo

hanno variato dal supporto nella definizione di requisiti e obiettivi iniziali, passando per l'implementazione di componenti specifici, fino alla complessa fase di Testing. La quantità di interazioni con ChatGPT ha rispecchiato l'intensità e la complessità delle esigenze in ciascuna fase, con un picco significativo durante il Testing. Questo ha permesso di sfruttare al meglio le capacità di ChatGPT, migliorando la qualità del codice, ottimizzando i processi di sviluppo e garantendo che l'applicazione rispondesse in modo efficace ai requisiti definiti. Nel complesso, il feedback ottenuto ha confermato l'utilità e la pertinenza di ChatGPT come strumento di supporto nello sviluppo di applicazioni software, con un impatto positivo sulla produttività e sulla qualità del prodotto finale.

Capitolo 6

Conclusioni

Per concludere quanto trattato, questo capitolo vuole appuntare le *lesson learned* dall'esperienza descritta, con un particolare focus sull'utilizzo pratico di ChatGPT e verso le implicazioni comportate sullo stesso.

6.1 L'importanza della definizione di un prompt

Un prompt ben definito è essenziale per ottenere risposte utili e accurate dal modello di linguaggio. Quando i prompt sono vaghi o mal strutturati, il modello potrebbe produrre risposte che non rispondono adeguatamente alle esigenze dell'utente. Inoltre, un prompt chiaro e specifico riduce il rischio di interpretazioni errate da parte del modello, migliorando l'affidabilità delle risposte. La precisione nella formulazione del prompt facilita anche la valutazione dell'efficacia del modello, poiché rende più evidente se il modello sta seguendo correttamente le istruzioni fornite.
Un prompt scritto con criterio può portare a risposte più pertinenti, utili e coerenti, migliorando significativamente l'interazione con il modello di linguaggio. Ad esempio, un prompt ben strutturato può aiutare il modello a mantenere un tono coerente durante una conversazione, a seguire istruzioni complesse e a evitare errori comuni. Questo non solo aumenta la soddisfazione dell'utente, ma può anche rendere l'uso del modello di linguaggio più efficiente e produttivo. Inoltre, promuove un migliore apprendimento del modello, poiché le risposte generate basate su prompt chiari e dettagliati forniscono dati più utili per il miglioramento continuo del modello stesso [23].

6.2 Il non-determinismo di ChatGPT

Il concetto di "non-determinismo" di ChatGPT si riferisce alla tendenza del modello a restituire risultati diversi o variazioni significative nel codice generato per la stessa istruzione di input. Tale fenomeno può influenzare la riproducibilità dei risultati, mettendo in discussione la validità delle conclusioni scientifiche ottenute attraverso l'analisi di *Large Language Models* come ChatGPT.
Si evidenzia che l'alto non determinismo di ChatGPT nella generazione di codice presenta rischi significativi per gli sviluppatori software e i ricercatori. Nel caso specifico, gli sviluppatori devono essere consapevoli delle limitazioni di

ChatGPT e dei potenziali rischi nell'utilizzare codice generato in produzione, con una particolare allerta di testare attentamente il codice generato prima del rilascio e di implementare processi di Testing e validazione più robusti per garantirne la determinismo e l'affidabilità. Inoltre, si sottolinea che trovare un equilibrio tra determinismo e creatività è essenziale per ottimizzare le prestazioni di *Large Language Models* per diverse applicazioni pratiche [24].

6.3 Utilità e percezione dello strumento

L'utilizzo di ChatGPT come strumento di supporto allo sviluppo dell'applicazione UniFolder ha richiesto l'impiego dello stesso su diversi fronti, per accompagnare il lavoro del programmatore in ognuna delle fasi esaminate. Risulta chiaro come il ChatBot fornisca una solida piattaforma sulla quale fare affidamento, e durante la realizzazione del progetto è emerso in maniera naturale come il ruolo di ChatGPT sia confinato (allo stato attuale delle cose) al supporto ad un programmatore umano, per quanto dimostrato dai dati raccolti su utilità e rielaborazione delle risposte, di seguito in figura.

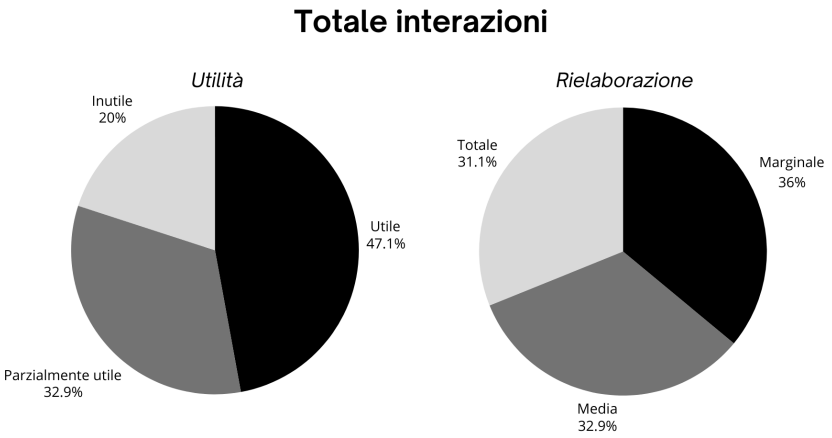


Figura 6.1: Statistiche delle interazioni totali con ChatGPT per lo sviluppo di UniFolder (campione +200 items).

Emerge in maniera piuttosto chiara come l'utilità dei contenuti sia apprezzata in una buona parte dei casi, ma per quanto riguarda la rielaborazione dei contenuti ci sia grande incertezza, con una distribuzione quasi equa tra i tre livelli di distinzione definiti, e spetta proprio al programmatore rimaneggiare i riscontri ottenuti attraverso l'interazione con il ChatBot; sulla base di ciò, è lecito dichiarare ChatGPT un ottimo alleato allo sviluppo di applicazioni mobile, ma non (ancora?) in grado di guidare l'intero sviluppo da sé.

6.3.1 Ethopoeia e ChatGPT: Un'Analisi

Il concetto di Ethopoeia

Nell'ambito della CASA (*Computers as Social Actors*) Theory, con il termine Ethopoeia si descrive il fenomeno per cui le persone reagiscono automaticamente e inconsciamente alle macchine nello stesso modo in cui reagiscono agli altri esseri umani. Il cervello umano si è sviluppato in un'epoca in cui solo gli esseri umani erano in grado di mostrare un comportamento sociale e in cui ogni persona e ogni luogo erano persone e luoghi reali. Per affrontare con successo la vita quotidiana, il cervello umano ha sviluppato risposte automatiche, che sono ancora in uso oggi.
Questo comportamento cosiddetto "*mindless*" può essere meglio inteso come una risposta automatica agli spunti sociali contestuali, provenienti da qualsiasi agente, che non prevede l'elaborazione attiva di tali spunti, ossia una sorta di incapacità di tracciare nuove distinzioni.
Inoltre, si ipotizza che le persone non solo rispondano senza pensieri, ma abbiano anche la tendenza a usare scorciatoie cognitive ed euristiche, e che quindi utilizzino le regole sociali facilmente accessibili dell'interazione uomo-uomo (HHI) e le applichino all'interazione uomo-macchina (HMI) - a causa della somiglianza funzionale percepita tra esseri umani e computer.
Esempi di questa somiglianza funzionale (o spunti sociali) sono l'uso del linguaggio naturale, l'interattività e l'assunzione di ruoli sociali tradizionalmente ricoperti dagli esseri umani, proprio come nel caso di ChatGPT [25].

Il fenomeno percepito in ChatGPT

Durante il corso delle interazioni (oltre 200 in totale), si è andato sempre più a delineare un fattore di Ethopoeia percepita nei confronti del ChatBot; in effetti, quest'ultimo non sembrava essere percepito strettamente come un LLM, bensì come *una seconda mente* sulla quale fare affidamento, e grazie al linguaggio naturale è parso sempre più crearsi uno scambio tipico del dialogo tra umani.
Una grossa componente che ha contribuito al fenomeno è stata lo scatenarsi di emozioni: stress e frustrazione in fase di *fixing* ed in prossimità della carenza di risposte del ChatBot, oppure soddisfazione ed orgoglio per il completamento di parti cruciali. Indubbiamente queste sono state "assorbite" dal prompt, rendendo ChatGPT partecipe anche di questo aspetto umano, e grazie all'abilità di ChatGPT di adottare diversi stili di comunicazione si può venire a creare una sorta di empatia in risposta a questo tipo di percezione.
Analizzare l'Ethopoeia in relazione a ChatGPT offre anche spunti su come gli utenti accettano le risposte del ChatBot. Quando ChatGPT risponde in modo convincente e adeguato al contesto, gli utenti possono trovarsi a interagire con esso come farebbero con un essere umano. Questo aspetto ha implicazioni significative per l'usabilità, la fiducia e l'accettazione dello strumento stesso.

6.4 Commento finale

Nel complesso, l'impiego di ChatGPT durante le diverse fasi del progetto UniFolder ha dimostrato di essere alquanto vantaggioso. Nonostante il numero relativamente ridotto di interazioni registrate in alcune fasi, il feedback generale fornito da ChatGPT è stato positivo, dimostrando alta qualità e discreta pertinenza delle risposte. In contesti di utilizzo più intenso, come durante la scrittura di codice e la risoluzione di problemi tecnici, ChatGPT ha offerto un supporto significativo, seppur non sempre con una pari qualità delle risposte, soddisfacendo le esigenze della maggior parte delle interazioni.
L'uso massiccio di ChatGPT in fasi particolarmente complesse e critiche del progetto ha messo in luce l'importanza del suo contributo. Nonostante alcune difficoltà iniziali, il ChatBot ha fornito indicazioni utili e soluzioni che hanno contribuito a migliorare la qualità complessiva e la robustezza del progetto. Questo dimostra l'efficacia di ChatGPT come strumento di supporto versatile e adattabile in diversi contesti di sviluppo.
L'integrazione di ChatGPT nel progetto UniFolder ha non solo accelerato il processo di sviluppo, ma ha anche elevato il livello qualitativo dell'applicazione. Le sue capacità di emulare il linguaggio umano e adattarsi a vari contesti sono state cruciali per la riuscita del progetto.
Questa esperienza ha dimostrato che strumenti basati su intelligenza artificiale, come ChatGPT, possono giocare un ruolo fondamentale nello sviluppo software, portando innovazione e efficienza nei processi di creazione di applicazioni complesse.

Bibliografia

[1] Marco Cascella et al. “Evaluating the Feasibility of ChatGPT in Healthcare: An Analysis of Multiple Clinical and Research Scenarios.” In: *Journal of medical systems* 47.1 (4 mar. 2023). https://link.springer.com/content/pdf/10.1007/s10916-023-01925-4.pdf ; https://doi.org/10.1007/s10916-023-01925-4, pp. 33–. DOI: 10.1007/s10916-023-01925-4. URL: https://lens.org/165-762-042-333-335.

[2] Hussein Mozannar et al. *The RealHumanEval: Evaluating Large Language Models' Abilities to Support Programmers*. https://arxiv.org/abs/2404.02806. 3 Apr. 2024. DOI: 10.48550/arxiv.2404.02806. URL: https://lens.org/082-679-288-724-057.

[3] Zhonghua Zheng et al. *Building Emotional Support Chatbots in the Era of LLMs*. https://arxiv.org/abs/2308.11584. 1 Gen. 2023. DOI: 10.48550/arxiv.2308.11584. URL: https://lens.org/168-883-277-542-668.

[4] Yiheng Liu et al. “Summary of ChatGPT-Related research and perspective towards the future of large language models”. In: *Meta-Radiology* 1.2 (2023). https://arxiv.org/pdf/2304.01852 ; https://arxiv.org/abs/2304.01852, pp. 100017–100017. DOI: 10.1016/j.metrad.2023.100017. URL: https://lens.org/017-871-633-579-958.

[5] Matthew S Jaffe et al. “Software requirements analysis for real-time process-control systems”. In: (1990).

[6] Tore Dyba e Torgeir Dingsoyr. “What do we know about agile software development?” In: *IEEE software* 26.5 (2009), pp. 6–9.

[7] Amy J. Ko et al. “The state of the art in end-user software engineering”. In: *ACM Computing Surveys* 43.3 (29 apr. 2011). https://dl.acm.org/doi/10.1145/1922649.1922658, pp. 21–44. DOI: 10.1145/1922649.1922658. URL: https://lens.org/134-124-959-869-763.

[8] Bashar Nuseibeh e Steve Easterbrook. “ICSE - Future of SE Track - Requirements engineering: a roadmap”. In: (2000). http://www.cse.chalmers.se/~feldt/courses/reqeng/papers/nuseibeh_2000_re_a_roadmap.pdf, pp. 35–46. DOI: 10.1145/336512.336523. URL: https://lens.org/064-722-439-234-281.

[9] Manishaben Jaiswal. “Android the mobile operating system and architecture”. In: *Manishaben Jaiswal," ANDROID THE MOBILE OPERATING SYSTEM AND ARCHITECTURE", International Journal of Creative Research Thoughts (IJCRT), ISSN* (2018), pp. 2320–2882.

[10] Pritee Uttarwar et al. “A Literature Review on Android-A Mobile Operating system”. In: *International Research Journal of Engineering and Technology* 8.1 (2021), pp. 1–6.

[11] Android Developers. *Android 7.0, (N) Compatibility Definition*. Accessed: 2024-06-19. 2024. URL: https://source.android.com/docs/compatibility/7.0/android-7.0-cdd.

[12] Android Developers. *Firebase*. Accessed: 2024-06-18. 2024. URL: https://developer.android.com/studio/write/firebase?hl=it.

[13] Android Developers. *Firebase Authentication*. Accessed: 2024-06-18. 2024. URL: https://firebase.google.com/docs/auth?hl=it.

[14] Android Developers. *Firebase Realtime Database*. Accessed: 2024-06-18. 2024. URL: https://firebase.google.com/docs/database?hl=it.

[15] Android Developers. *Firebase Cloud Firestore*. Accessed: 2024-06-18. 2024. URL: https://firebase.google.com/docs/firestore?hl=it.

[16] Android Developers. *Firebase Cloud Storage*. Accessed: 2024-06-18. 2024. URL: https://firebase.google.com/docs/storage?hl=it.

[17] Android Developers. *Guida all'architettura delle app*. Accessed: 2024-06-25. 2024. URL: https://developer.android.com/topic/architecture?hl=it#recommended-app-arch.

[18] Craig Larman e Luca Cabibbo. *Applicare UML e i pattern : analisi e progettazione orientata agli oggetti*. Italian. 5. ed. Milano: Pearson Italia, 2020. ISBN: 9788891904591; 8891904597.

[19] Jiantao Pan. “Software testing”. In: *Dependable Embedded Systems* 5.2006 (1999), p. 1.

[20] Paul Blundell e Diego Torres Milano. *Learning Android application testing : improve your Android applications through intensive testing and debugging*. English. Second edition. Birmingham, UK: Packt Publishing, 2015. ISBN: 9781784397999; 1784397997. URL: https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=971802.

[21] Android Developers. *Cosa testare in Android*. Accessed: 2024-06-28. 2024. URL: https://developer.android.com/training/testing/fundamentals/what-to-test?hl=it.

[22] Android Developers. *UI Tests*. Accessed: 2024-07-01. 2024. URL: https://developer.android.com/training/testing/instrumented-tests/ui-tests.

[23] J.D. Zamfirescu-Pereira et al. *Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts*. https://dl.acm.org/doi/pdf/10.1145/3544548.3581388 ; https://doi.org/10.1145/3544548.3581388. 19 Apr. 2023. DOI: 10.1145/3544548.3581388. URL: https://lens.org/129-892-094-684-444.

[24] Shuyin Ouyang et al. “LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation”. In: *arXiv preprint arXiv:2308.02828* (2023).

[25] Astrid M Von der Pütten et al. ""It doesn't matter what you are!" Explaining social effects of agents and avatars". In: *Computers in Human Behavior* 26.6 (2010), pp. 1641–1650.