Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

Sviluppo dell'app GreenPalate con il supporto di ChatGPT: valutazione dell'esperienza

Relatore: *Proff.ssa Daniela Micucci*

Co-relatore: *Dott.ssa Maria Teresa Rossi*

Relazione della prova finale di:

Fabio Nonnis

Matricola 857170

Anno Accademico 2023-2024

Indice

Capitolo 1

Introduzione

La seguente relazione intende analizzare l'utilizzo dei LLM (Large Language Model), più in particolare ChatGPT, come strumento di supporto per lo sviluppo di una applicazione mobile. L'applicazione che verrà sviluppata consente di eseguire ricerche di alimenti utilizzando il loro nome o il codice a barre del prodotto e restituisce determinate informazioni, tra cui dati più classici come i valori nutrizionali l'applicazione, con l'intento di facilitare scelte più sostenibili negli alimenti che si vanno a consumare.

1.1 Obiettivo dell'applicazione

GreenPalate nasce con l'idea di creare un'applicazione a tema sostenibilità. Il suo punto di forza è infatti di permettere in maniera semplice di vedere il punteggio EcoScore assegnato a ogni prodotto che si va a cercare. L'Ecoscore [20] è un punteggio calcolato in maniera standard consistente di un voto dalla A alla E, dove A è il punteggio assegnato ad un alimento con minimo impatto ambientale ed E all'alimento con massimo impatto ambientale. L'applicazione permette anche di vedere molti altri valori che oltre alla sostenibilità dal punto di vista ambien-

tale permettono all'utente di valutarne anche la qualità da un punto di vista nutrizionale.

1.2 Large Language Model e ChatGPT

Negli ultimi anni la nascita di nuovi LLM come ChatGPT è sempre più frequente e la loro diffusione sia come applicazioni a sè stanti che come integrazione in altre applicazioni come supporto. Per esempio nelle ultime versioni di Android Studio, IDE per lo sviluppo di applicazioni android, è integrato Gemini, il LLM sviluppato da Google.

È stato deciso di utilizzare ChatGPT, uno dei più diffusi al momento, nella sua versione 3.5 come strumento di supporto durante ogni fase di progettazione, implementazione e testing dell'applicazione android così da vederne l'efficacia e la potenza.

ChatGPT è il modello di linguaggio generativo sviluppato da OpenAI, utilizza tecniche di Deep Learning per comprendere il linguaggio naturale. ChatGPT è ottimizzato per dare risposte, sempre in linguaggio naturale, rendendosi un potente strumento per diverse applicazioni, tra le quali il supporto alla creazione di software. Proprio tale funzionalità verrà analizzata in questo elaborato.

ChatGPT si è rivelato un ottimo strumento di supporto in maniera particolare per la fase di implementazione dove è riuscito a fornire risposte utili allo sviluppo, codice coerente alle richieste effettuate e in generale aiuto all'implementazione delle varie funzioni dell'applicazione.

Capitolo 2

Analisi dei Requisiti

L'analisi dei requisiti rappresenta una fase fondamentale nello sviluppo di un'applicazione Android, ma più in generale nello sviluppo software, poiché consente di definire chiaramente le necessità degli utenti finali così come le funzionalità che devono essere implementate. Questo capitolo descrive i requisiti funzionali e non funzionali dell'applicazione, basati su una combinazione di ricerche, interviste con i potenziali utenti e l'analisi di applicazioni simili già esistenti sul mercato.

2.1 Requisiti funzionali

I requisiti funzionali descrivono le specifiche funzionalità e i servizi offerti dall'applicazione, aiutano quindi a dare una guida su cosa va implementato. I requisiti funzionali definiscono cosa l'applicazione deve fare. Verrà assegnato un punteggio di priorità da 1 a 5 ai seguenti requisiti, verrà anche indicato quali di questi sono stati implementati

Ricerca per nome : *[Priorità 5, implementato completamente]*

La ricerca per nome deve fornire una lista di risultati di cibi che contengono le parole cercate o simili mediante le API Open-

FoodFactsAPI [24]. I risultati della ricerca devono essere mostrati in formato compatto, ma che permetta all'utente di trovare facilmente e riconoscere il cibo a cui è interessato o che sta cercando.

La visualizzazione compatta fornirà le seguenti informazioni: immagine dell'alimento, nome, marchio e EcoScore.

L'utente deve poter aprire il risultato a cui è interessato per vedere informazioni aggiuntive e più dettagliate.

Ricerca per codice a barre : *[Priorità 3, implementato completamente]*

La ricerca per codice a barre deve fornire un unico risultato tramite chiamata API OpenFoodFactsAPI. In particolare verrà aperta una schermata che mostra nel dettaglio tutte le informazioni dell'alimento ricercato, la stessa schermata che appare una volta selezionato l'alimento dalla ricerca per nome.

Nella schermata dettagliata verranno mostrati oltre alle informazioni della "Ricerca per nome" anche valori nutrizionali, NutriScore [25], materiali della confezione e allergeni.

Cronologia alimenti visualizzati : *[Priorità 1, implementato completamente]*

L'applicazione deve fornire una breve cronologia degli ultimi alimenti che sono stati visualizzati nel dettaglio così che l'utente possa riaprirli rapidamente nel caso.

Gli elementi della cronologia verranno mostrati in maniera compatta proprio come nella "Ricerca per nome" e una volta aperti verranno visualizzarti dettagliatamente come nella "Ricerca per

Codice a barre".

2.2 Requisiti non funzionali

I requisiti non funzionali sono le caratteristiche dell'applicazione non richieste dai requisiti funzionali, ma che hanno comunque un impatto sulla sua implementazione. Dato che non descrivono funzionalità, ma la modalità di esecuzione di particolari compiti del sistema.

Performance :

- Il sistema deve mostrare i risultatati al massimo 1 secondo dopo aver ricevuto la risposta dalle API, sia per la ricerca con nome che per la ricerca con codice a barre.
- La lettura del codice a barre deve essere letto in non più di 2 secondi una volta inquadrato.

Usabilità :

- Il sistema deve avere un'interfaccia grafica intuitiva e quindi facile da utilizzare, con un design chiaro che rispetti le linee guida Material Design [21].
- Il sistema deve fornire sia una modalità chiara con colori chiari che una modalità "notte" con colori più scuri per facilitare l'utilizzo nelle ore notturne evitando l'affaticamento degli occhi.

Affidabilità :

- Il sistema deve funzionare in presenza di connessione alla rete, a meno di malfunzionamenti del server verso il quale vengono effettuate le chiamate API.
- Il sistema deve gestire i principali errori che possono occorrere, riscontrati durante la fase di test dell'applicazione.
- La lettura del codice a barre deve restituire il codice corretto almeno 8/10 volte.

Manutenibilità :

- Il codice deve essere scritto in maniera pulita e comprensibile così da poter essere manutenuto e aggiornato con semplicità.

Compatibilità :

- L'applicazione deve essere compatibile con la maggior parte dei dispositivi mobili utilizzanti Android, nel dettaglio quelli aggiornati alla versione Android Nougat 7.0 o superiore (versione API ≥ 24).
- L'applicazione deve essere compatibile con le varie risoluzioni e formati di schermi dei dispositivi che potrebbero utilizzarla.

Localizzazione :

- L'applicazione deve supportare le lingue inglese e italiana

2.3 Analisi delle parti interessate

In questa sezione verranno identificate le parti interessate coinvolte nello sviluppo e nell'utilizzo dell'applicazione e ne verranno descritti i ruoli.

Utente finale :

L'utente finale è una persona interessata al suo impatto ambientale che quindi decide di scaricare l'applicazione così da avere una maggiore conoscenza degli impatti ambientali del cibo che consuma, avendo anche l'opzione di cercare un'alternativa migliore da quel punto di vista. Più in generale sono anche quelle persone interessate ad uno stile di vita salutare o che semplicemente vogliono conoscere i valori nutrizionali degli alimenti che acquistano.

Esigenze specifiche :

- **Facilità di utilizzo:** Interfaccia intuitiva e user-friendly.
- **Informazioni accurate:** Accesso a dati aggiornati e precisi.
- **Ricerca rapida:** Risultati restituiti velocemente nella ricerca.
- **Accessibilità** Disponibilità di diverse lingue.

Sviluppatori :

Gli sviluppatori sono i professionisti tecnici che progettano, sviluppano, testano e mantengono l'applicazione.

Requisiti tecnici :

- **Tecnologie e strumenti adeguati:** Accesso agli strumenti di sviluppo necessari, come ambienti di sviluppo integrati (Android Studio [1]), sistemi di controllo della versione (Git) e piattaforme di testing.
- **Standard di Codifica:** Linee guida e best practice per garantire un codice pulito, manutenibile e scalabile.

2.4 Casi d'uso

I casi d'uso descrivono le interazioni degli utenti con il sistema, illustrando come gli utenti utilizzeranno l'applicazione per raggiungere specifici obiettivi.

1. **Ricerca alimento per nome**

   Attori: Utente finale.

   Descrizione: L'utente ricerca un alimento specifico utilizzando la funzione di ricerca per nome dell'applicazione.

   Precondizioni :

   - L'utente ha a disposizione una connessione a internet.
   - Il sito fornitore delle API è in funzione.

   Flusso principale :

   (a) L'utente avvia l'applicazione.

(b) L'utente seleziona la barra di ricerca nella schermata principale e inserisce il nome dell'alimento desiderato.

(c) L'utente seleziona l'alimento desiderato tra le opzioni che vengono fornite dalla ricerca e ne visualizza le informazioni.

Post condizioni :

- L'utente visualizza le informazioni dettagliate sull'alimento.
- La scheda dell'alimento viene salvata nella cronologia e quindi mostrata nella schermata principale dell'applicazione.

Estensioni :

(c.2) La ricerca non produce risultati e viene mostrato un segnale di errore.

2. Ricerca alimento per codice a barre

Attori: Utente finale

Descrizione: L'utente ricerca un alimento specifico utilizzando la funzione di ricerca per codice a barre inquadrando il codice a barre con la fotocamera.

Precondizioni :

- L'utente ha a disposizione una connessione a Internet.

- La fotocamera esterna dello smartphone funziona correttamente.
- Il sito fornitore delle API è in funzione.

Flusso principale :

(a) L'utente avvia l'applicazione.

(b) L'utente preme il bottone della fotocamera per scansionare il codice a barre e lo inquadra.

(c) Viene mostrata la pagina del prodotto con tutti i dettagli.

Post condizioni :

- L'utente visualizza le informazioni dettagliate dell'alimento.
- La scheda dell'alimento viene salvata nella cronologia e quindi mostrata nella schermata principale dell'applicazione.

Estensioni :

(c.2) La ricerca non produce risultati e viene mostrato un segnale di errore.

3. Apertura scheda della cronologia

Attori: Utente finale.

Descrizione: L'utente apre una scheda della cronologia per visualizzare i dettagli del prodotto. I prodotti presenti nella cronologia sono gli ultimi sette che l'utente aveva già cercato e visualizzato.

Precondizioni :

- L'utente ha a disposizione una connessione a Internet.
- L'utente deve aver effettuato almeno una ricerca così da popolare la cronologia.

Flusso principale :

(a) L'utente avvia l'applicazione.

(b) L'utente seleziona dalla cronologia, presente nella schermata principale, il prodotto di cui desidera conoscere le informazioni dettagliate.

(c) Viene mostrata la pagina del prodotto con tutti i dettagli.

Post condizioni :

- L'utente visualizza le informazioni dettagliate del prodotto.
- La cronologia viene aggiornata e l'alimento visualizzato apparirà ora in cima.

Capitolo 3

Implementazione

In questo capito verrà descritta l'implementazione dell'applicazione, verranno approfondite le scelte architetturali, la struttura dell'applicazione, Gli strumenti utilizzati per lo sviluppo e le scelte nella gestione della funzionalità, sempre rispettando le linee guida fornite da Google per la corretta implementazione di una applicazione Android.

3.1 Architettura dell'applicazione

L'architettura di un'applicazione è l'organizzazione del software, la divisione e le funzioni delle varie componenti che la compongono.

3.1.1 Modello architetturale

Il modello architetturale utilizzato per lo sviluppo dell'applicazione è il Model-View-ViewModel [27] (MVVM). Il modello Model-View-ViewModel è un'architettura di progettazione del software utilizzata per separare la logica di presentazione dalla logica di business in modo da rendere il codice più modulare, e manutenibile rispettando i pattern "low coupling" [28] e "high cohesion" [29]. È particolarmente utile nel-

lo sviluppo di applicazioni Android. Di seguito ciascun componente in dettaglio:

Model: Il "Model" contiene la logica di business e i dati dell'applicazione. Include le definizioni delle entità, l'accesso ai dati, per esempio mediante le repository e le API.

Ha il ruolo di ottenere e gestire/elaborare i dati utilizzati dall'applicazione

View: La "View" include tutta l'interfaccia utente dell'applicazione, quindi *Activity* [2], *Fragment* [3] e altri componenti UI che mostrano i dati. Contiene solo la logica di presentazione, quindi tutta la logica che ha come scopo di presentare i dati e modificare l'interfaccia oltre che accettare gli input degli utenti. Si collega al ViewModel per ricevere aggiornamenti e mostrare i dati all'utente.

Viewmodel: Il "ViewModel" agisce come intermediario tra il Model e la View. Contiene la logica di presentazione e gestisce lo stato della View.

Raccoglie i dati dal Model e li prepara per la visualizzazione nella View. Utilizza LiveData [4] o altre forme di osservabili per notificare automaticamente la View quando i dati cambiano, rendendo l'applicazione più reattiva.

3.1.2 Componenti principali

L'applicazione è composta da diversi componenti principali che dividono tra loro i vari compiti.

Activity e Fragment (*View*): sono le componenti che di un'interfaccia utente, hanno il compito di gestire l'interfaccia e quindi l'interazione dell'utente con l'applicazione. Non contengono la logica dell'applicazione, anche per motivi di low coupling e di sicurezza, gestiscono solo la visualizzazione delle informazioni e ricevono gli input dell'utente che vengono poi eventualmente processati da altre componenti architetturali.

Viewmodel: Riceve i dati dalla UI (activity e fragment), immagazzinati in componenti chiamate LiveData, e si interfaccia con le Repository che contengono la logica.

3.2 Strumenti e tecnologie

In questa sezione sono presentati tutti gli strumenti e le tecnologie utilizzate per lo sviluppo dell'applicazione.

3.2.1 Ambiente di sviluppo

Per lo sviluppo di questa applicazione, come per la maggior parte delle applicazioni Android, è stato utilizzato l'IDE (integrated development environment) Android Studio che è lo strumento di sviluppo per Android ufficiale creato e mantenuto da Google.

Per favorire un'utilizzabilità più ampia possibile è stato scelto di rendere l'app compatibile con telefoni che supportano Android Nougat 7.0 versione API 24 o superiori, rendendo così l'app compatibile con il 97,4% [5] dei dispositivi mobili Android.

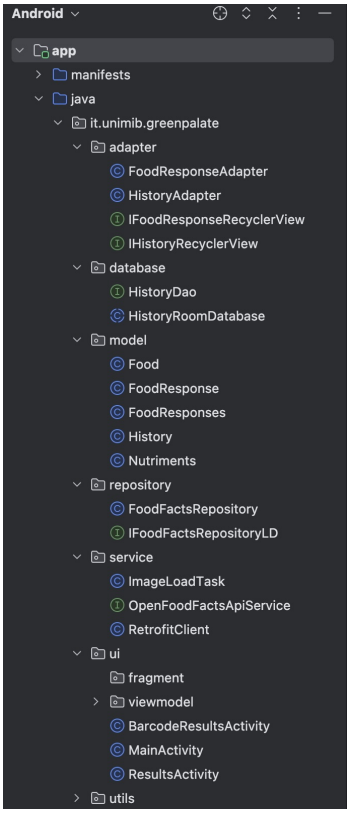

Figura 3.1: Struttura dei file del progetto

3.2.2 Linguaggio di programmazione

Come linguaggio di programmazione è stato scelto Java [30], linguaggio di programmazione Object-oriented, scelto per la mia personale conoscenza del linguaggio e per la grande community di supporto rispetto a Kotlin [23], seppur sempre più comunemente adottato per lo sviluppo di applicazioni Android

3.2.3 Librerie e Frameworks

UI: Material Design, linee guida di Google, per creare interfacce utente intuitive e unificate rispetto ad altre applicazioni.

Accesso a risorse remote: Per l'accesso a risorse remote, nel caso specifico per l'accesso a API è stata utilizzata la libreria Retrofit [26], una libreria apposita per effettuare chiamate api in maniera semplice.

Database: Come database locale è stata utilizzata Room [6], libreria che permette di implementare un database interno all'applicazione gestendolo con query SQL.

3.3 Sviluppo delle funzionalità principali

In questa sezione verranno analizzate e descritte le implementazione delle principali funzionalità fornite dall'applicazione. Le funzionalità sono le stesse descritte nel capitolo 2.

Ricerca degli alimenti per nome: Questa è la funzionalità principale dell'applicazione. Tramite questa funzione si effettua una ricerca mediante stringa che restituisce come risultato una collezione di alimenti con le loro informazioni.

- **Interfaccia:** Come componenti di interfaccia utente per questa funzionalità sono utilizzate una `SearchView` [7] e una `RecyclerView` [8];

  La `SearchView` nella schermata principale ha come scopo dare la possibilità all'utente di scrivere l'alimento che vuole cercare. La stringa verrà utilizzare per completare la query di richiesta alle API.

Figura 3.2: Schermata principale dell'applicazione

La `RecyclerView` è utilizzata per mostrare i risultati della ricerca, gli elementi sono mostrati singolarmente in `Items` che la popolano.

- **Gestione della query di ricerca:** La query viene gestita con `OnQueryTextListener` [9] che "ascolta" la `SearchView` e ne osserva lo stato durante il cambiamento. Cioè mentre l'utente scrive la query di ricerca, in seguito tramite il metodo `onQueryTextSubmit` una volta che l'utente preme il tasto di invio fa si che venga salvata la stringa e inviata alla classe che si occupa di svolgere le attività per la chiamata API.
- **Visualizzazione dei risultati:** La visualizzazione dei risultati avviene in una nuova activity, figura 3.3, chiamata `ResultsActivity` dove vengono mostrati mediante `RecyclerView`,

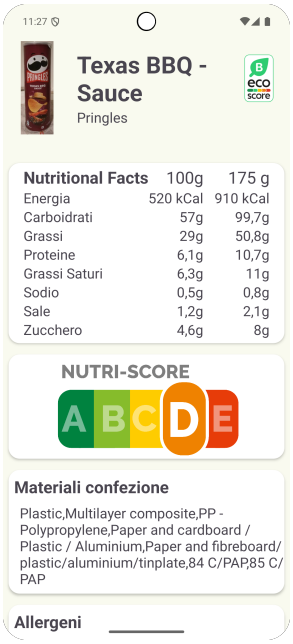

Figura 3.3: Schermata dettagliata

popolata da `items` che a loro volta sono creati con la classe `Adapter` [10]. Tutti i dati sono invece recuperati tramite una chiamata API eseguita con Retrofit, la chiamata viene fatta a *OpenFoodFacts* e recupera tutti i dati degli alimenti che contengono la query nel loro nome.

Visualizzazione dettagliata alimento: La visualizzazione non è una funzione a sé stante, ma una funzionalità essenziale che viene richiamata in quasi tutte le modalità di utilizzo dell'applicazione.

- **Interfaccia:** L'interfaccia è composta da molti elementi essendo una pagina predisposta alla visualizzazione di diversi componenti visive, nel dettaglio vengono utilizzate
  - `ImageView` [11] per la visualizzazione dell'immagine di copertina dell'alimento, per l'EcoScore e NutriScore.

  - `TextView` [12] per la visualizzazione del nome prodotto, nome del marchio e, quando presenti, materiali della confezione e allergeni.
  - `TableView` [13] per la visualizzazione dei valori nutrizionali dell'alimento separati tra valori per 100 grami di prodotto e per porzione.

  Tutti questi elementi per creare separazione sono inseriti a loro volta in delle `CardView` [14].

- **Gestione dei dati:** I dati arrivano in risposta alla chiamata API (utilizzando il codice numerico collegato all'alimento) in formato JSON [22], quindi vengono salvati in un oggetto `FoodResponse` che vengono creati automaticamente utilizzando l'annotazione `@SerializedName` per inizializzare ogni variabile.

Ricerca alimenti per codice a barre: Questa è la funzionalità secondaria dell'applicazione. Creata per controllare rapidamente i valori nutrizionali e tutti i dettagli di un prodotto che si ha tra le mani dato che è necessario inquadrarne il codice a barre. Così come nella ricerca per nome si effettua una chiamata API, ma utilizzando il codice numerico collegato al codice a barre.

- **Interfaccia:** L'interfaccia è molto semplice in quanto composta solo da un `ImageButton` [15] che una volta cliccato apre la fotocamera che automaticamente legge il codice inquadrato e porta alla schermata successiva.
- **Lettore codice a barre:** La funzionalità di lettura del codice a barre è fornita da lo scanner di codici Google (API Google code scanner [19]), funzione che non richiede neanche l'accesso alla fotocamera da parte dell'applicazione poiché

la funzione è delegata ai Google Play Services che restituiscono semplicemente il valore del codice all'applicazione. Il lettore interno all'applicazione per motivi di ottimizzazione è costruito in modo che possa leggere solo codici a barre e effettui uno zoom automatico quando aprendo la funzione di lettura identifica il codice.

- **Visualizzazione dei dati:** Per la visualizzazione dei dati viene mostrata la schermata descritta precedentemente in "*visualizzazione dettagliata alimento*".
- **Recupero dati:** Il recupero dei dati viene fatto tramite chiamata API utilizzando il codice letto con il Google code scanner.

Salvataggio cronologia: La funzionalità di cronologia offre una breve lista composta dagli ultimi sette prodotti che sono stati aperti nella *visualizzazione dettagliata*. Gli oggetti di tipo `FoodResponse` vengono salvati in un DataBase locale e poi mostrati nella schermata principale.

- **Interfaccia:** Gli elementi sono visualizzati in una `RecyclerView` all'interno della activity principale dell'applicazione. Rimane visibile anche quando il database `Room` non è popolato.
- **Database:** Per immagazzinare gli elementi viene utilizzata la libreria `Room`. La libreria `Room` fornisce un accesso al database sfruttando la potenza di SQLite. Gli oggetti vengono salvati come oggetti di tipo `History`, classe creata per poter salvare solo gli elementi essenziali che vengono mostrati, così da non gravare inutilmente sulla memoria. nel dettaglio vengono salvati codice a barre, nome, nome del marchio, immagine ed Ecoscore. Viene anche creato un codice univoco auto-generato così da tenere traccia dell'ordine

di inserimento degli elementi nel database e mostrarli nella `RecyclerView` da quello aggiunto più recentemente. Anche il codice a barre pur non essendo la chiave primaria viene trattato come un valore unico.

3.4 Gestione API

Molte delle funzioni dell'applicazioni fanno affidamento all'utilizzo delle API. L'API utilizzata per recuperare tutte le informazioni collegate ai prodotti è OpenFoodFactsAPI, usata per accedere a una banca dati open-source che contiene tutti i dati collegati agli alimenti, OpenFoodFacts è un database popolato pubblicamente quindi ha come difetto che non sempre tutti i dati sono completi.

Per le chiamate API viene utilizzata la libreria `Retrofit`, una libreria per eseguire chiamate HTTP facile da definire ed utilizzare in Android. Nella applicazione è implementato con il pattern *Singleton* quindi viene creata un'istanza del client Retrofit che permette le chiamate e poi viene richiamata così da non creare il client ad ogni utilizzo sprecando risorse. Retrofit permette di selezionare il formato con cui restituisce i dati, in questo caso si è optato per il formato di dati JSON così da poter gestire più semplicemente i dati grazie alla libreria GSON.

Vengono infine create delle query per le chiamate attaccando a l'URL gli endpoint richiesti che possono essere il codice univoco dell'alimento codificato nel codice a barre oppure una stringa. I dati in formato JSON vengono letti e inizializzati in oggetti Java e poi mostrati a display.

Viene anche usata l'API `Google code scanner` utilizzata unicamente per la lettura dei codici a barre. Per l'utilizzo viene creato uno scanner personalizzato per ottimizzare la lettura dei codici a barre.

Infine viene utilizzata l'API Android Espresso [16] per il testing dell'interfaccia utente.

Capitolo 4

Progetto di testing

In questo capitolo vengono descritte tecniche e strumenti utilizzati per costruire il progetto di testing dell'applicazione. Il test ha focus principale sull'interfaccia utente dal momento che l'applicazione non ha una vera e propria parte di logica testabile. I test vengono effettuati tramite `Espresso`, il tool integrato che mette a disposizione Android, il quale tramite API mette a disposizione strumenti semplici per testare l'interfaccia.

Il testing dell'interfaccia utente è essenziale per garantire il funzionamento dell'applicazione così come è stato pensato. In questo modo si può garantire un'esperienza fluida e priva di errori.

4.1 Test UI

Il testing della UI è una fase dello sviluppo dell'applicazione che mira a verificare le varie funzionalità offerte così da scovare gli errori che si palesano durante l'utilizzo dell'applicazione e risolverli.

4.1.1 Espresso

Espresso è il framework che è stato utilizzato per sviluppare tutti i test dell'interfaccia per l'applicazione. Ha un API fornita da Google integrabile nel progetto dell'applicazione, facile da utilizzare e affidabile.

Permette di creare test sulla Interfaccia dell'applicazione che ne simulano l'utilizzo interagendo con gli elementi di UI seguendo le istruzioni Java.

Caratteristiche principali :

- **API semplice e intuitiva:** Espresso fornisce una API facile da integrare e utilizzare, è possibile interagire con gli elementi di UI semplicemente utilizzando i loro ID creati in fase di sviluppo.
- **Sincronizzazione automatica:** Espresso gestisce automaticamente la sincronizzazione con il thread dell'interfaccia utente. Attende quindi che la UI sia in uno stato di inattività prima di eseguire le interazioni, riducendo così la necessità di aggiungere ritardi artificiali nei test.
- **Matcher potenti:** Espresso utilizza dei matcher per identificare elementi di UI tramite i loro codici id, consentendo di selezionare con precisione ogni elemento dell'interfaccia e testarlo in maniera specifica.
- **Integrazione AndroidJUnitRunner:** L'integrazione con `AndroidJUnitRunner` consente a Espresso di eseguire test sia su dispositivi fisici che su dispositivi emulati.

4.1.2 Test dettagliati

In questa sezione sono descritti in maniera dettagliata e schematica i test che verificano le funzionalità presenti nell'applicazione.

1. **Test ricerca con stringa scorretta:**

   Descrizione breve: Il test verifica che inserendo una stringa scorretta, che non possa è un alimento, il sistema non restituisca risultati ma apra un dialogo di errore.

   Step :

   (a) L'applicazione viene avviata.
   (b) Viene inserita un stringa numerica nella barra di ricerca che non è presente nel database come alimento.
   (c) Viene avviata la ricerca
   (d) Il sistema dopo l'arrivo della risposta API mostra a display un messaggio di errore.

   Condizioni di successo del test: Il test viene passato se una volta la risposta del sistema alla ricerca è l'apertura del dialogo di errore per risultato inesistente.

2. **Test ricerca con stringa corretta:**

   Descrizione breve: Il test verifica la funzionalità di ricerca, quindi verifica che inserendo una stringa di cui si conosce il

risultato nel caso di ricerca il sistema restituisca effettivamente la schermata con i risultati.

Step :

(a) L'applicazione viene avviata.
(b) Viene inserita una stringa corretta , in questo caso "banana", che quindi è un nome di un alimento presente nel database.
(c) Viene avviata la ricerca
(d) Il sistema apre la schermata di risultati con la lista di risultati per la ricerca "banana".

Condizioni di successo del test: Il test viene considerato un successo se il sistema apre la schermata dei risultati ed i risultati sono presenti.

3. Test di apertura lettore per codici a barre:

Descrizione breve: Il test verifica semplicemente che con la pressione del bottone designato venga aperta la fotocamera per leggere il codice a barre.

Step :

(a) Viene aperta l'applicazione.
(b) Viene premuto l' `ImageButton` adibito all'apertura del lettore di codice a barre.

(c) Si apre una schermata della fotocamera così da leggere il codice a barre.

Condizioni di successo del test: Il test viene superato se si verifica l'apertura della fotocamera.

Note: purtroppo non è possibile testare la lettura dei codici a barre perché andrebbe fatto con un codice a barre fisico e non è possibile farlo con l'emulatore.

4. Test apertura dettagli alimento:

Descrizione breve: Il test verifica l'apertura della pagina dei dettagli alimento dopo una ricerca.

Step :

(a) Viene aperta l'applicazione.
(b) Viene inserita una stringa corretta nella barra di ricerca.
(c) Viene eseguita la ricerca.
(d) Si apre la schermata con la lista dei risultati.
(e) viene selezionato il primo elemento della lista.
(f) Si apre la schermata con tutti i dettagli legati all'alimento selezionato.

Condizioni di successo del test: Il test ha successo quando si apre correttamente la schermata dei dettagli.

5. **Test inserimento cronologia:**

Descrizione breve: Il test verifica che dopo aver aperto la schermata dettagliata di un alimento questo venga salvato nella cronologia delle ricerche recenti.

Step :

(a) Viene aperta l'applicazione.
(b) Viene inserita una stringa corretta nella barra di ricerca.
(c) Viene eseguita la ricerca.
(d) Si apre la schermata con la lista dei risultati.
(e) Viene selezionato il primo elemento della lista.
(f) Si torna alla schermata principale(dove è presente la cronologia).

Condizioni di successo del test: Il test ha successo se l'elemento che viene aperto è lo stesso che appare come primo nella cronologia questo viene verificato controllando che il codice identificativo unico del codice a barre dell'alimento sia lo stesso.

Capitolo 5

Utilizzo di ChatGPT

Nel seguente capitolo verrà analizzato l'utilizzo fatto di ChatGPT, in quanto strumento di supporto per lo sviluppo dell'applicazione descritta nei precedenti capitoli. Nel dettaglio verranno analizzate le fasi di analisi dei requisiti, implementazione e testing. Verrà inoltre anche valutata l'utilità delle risposte di ChatGPT così da valutare quanto sia uno strumento efficace come supporto a uno sviluppo di applicazioni mobili.

5.1 Analisi dei requisiti

[Percentuale utilizzo: 40%, percentuale risposte utili 50%]

Per questa fase di progettazione ChatGPT è stato usato come supporto per ottenere consigli sui requisiti funzionali e non funzionali utili per il tipo di applicazione che si andava a sviluppare. È stato utilizzato per chiedere dei consigli su i requisiti che potrebbe aver avuto una applicazione di questo tipo e poi valutare su quali fossero le idee migliori.

In questa fase ho trovato che ChatGPT possa essere utile come suggeritore o appoggio, ma sicuramente non essenziale. Tuttavia è

comunque considerabile uno strumento utile soprattutto valutandone la facilità di utilizzo.

5.2 Implementazione

[Percentuale utilizzo: 50%, percentuale risposte utili 65%]

In questa fase ChatGPT è stato utilizzato come supporto alla creazione di codice o anche come supporto al debugging.

5.2.1 Fasi di implementazione

Creazione Viewmodel: ChatGPT è stato usato in questa fase per controllare la struttura dei `ViewModel` [17] e dei `ViewModelFactory` [18] ad esso collegato. Quindi per scrivere le classi di `ViewModel` in maniera corretta. In particolare la classe che ho creato è `FoodFactsViewmodel`, la quale ha come compito fare la richiesta di get alle API per ottenere i dati collegati agli alimenti, che siano i dati completi quando viene effettuata la richiesta tramite codice a barre oppure richiesta per la lista di alimenti quando viene effettuata la richiesta tramite nome.

Gestione JSON: In questa fase è stato utilizzato per capire come gestire al meglio il file JSON restituito dalle API con la chiamata GET. Grazie a ChatGPT ho utilizzato l'annotazione `SerializedName` di Google per salvare automaticamente i campi del file JSON in oggetti o parametri di oggetti Java. Per questa fase è stato molto utile il supporto di ChatGPT in quanto ho testato diversi metodi per riuscire ad utilizzare le risposte alle chiamate API che venivano effettuate dal momento che sono file JSON molto lunghi soprattutto quando viene effettuata la chiamata GET per nome che restituisce la lista.

Gestione file XML: ChatGPT è stato utilizzato anche per la gestione di alcuni file XML che inizialmente venivano mostrati in maniera scorretta internamente all'applicazione.

Room DataBase: Per il database è stato utilizzato per comprendere meglio il funzionamento delle chiavi così da poter avere per ogni elemento una chiave primaria auto-generata in maniera incrementale e utilizzare comunque come chiave unica il codice a barre preesistente dell'alimento. Questo consente di aggiornare il database e mostrare a schermo la cronologia in ordine di ultima ricerca effettuata.

5.2.2 Conclusioni

Durante la fase di sviluppo, ChatGPT si è rivelato molto utile, talvolta anche in grado di suggerire frammenti di codice funzionanti o addirittura capace di effettuare debug del codice che si inserisce come query. Comunque la sua utilità resta quella di un tool di supporto, non è ancora in grado di generare sempre codice corretto, completo e utilizzabile.

5.3 Testing

[Percentuale utilizzo: 20%, percentuale risposte utili 30%]

Durante la fase di testing ChatGPT è stato utilizzato in maniera analoga a come è stato utilizzato nella fase di implementazione. Per supporto alla creazione ed al debugging di codice di testing, non è servito come linea guida su quali test andare a creare dal momento che le funzionalità principali dell'applicazione sono ben definite e non è presente una vera e propria logica in senso stretto da testare con JUnit.

Gestione del DataBase: Durante la fase di testing `Espresso` si sono notati dei problemi riguardanti il funzionamento del database Room, nel dettaglio i dati venivano cancellati a ogni esecuzione della batteria di test. ChatGPT è stato utilizzato per analizzare il problema e provare a trovarne una soluzione.

Test RecyclerView: ChatGPT è stato utilizzato per capire come interagire con la `RecyclerView` contenente la lista degli alimenti restituita dalla ricerca degli alimenti per nome.

5.3.1 Conclusioni

ChatGPT utilizzato nella fase di testing anche se molto simile a come può essere usato nella fase di Implementazione è risulto meno efficace, dando risposte meno utili e talvolta restituendo codice che non funziona sempre correttamente.

Capitolo 6

Conclusione

Per la stesura di questo elaborato è stata creata una applicazione android utilizzando il supporto di ChatGPT così da valutarne e verificarne la sua utilità in questo ambito. L'applicazione è una applicazione che ha come obiettivo la promozione di scelte sostenibili nell'alimentazione

Durante tutta la fase di progettazione e sviluppo dell'applicazione è stato utilizzato come strumento di supporto principale ChatGPT così da poterne valutare il funzionamento nell'ambito dello sviluppo di applicazioni mobili. Il progetto della relazione era inserito in una ricerca più grande sull'utilizzo di ChatGPT e questo doveva essere utilizzato quando ritenuto necessario quindi senza utilizzarlo più spesso per raccogliere dati.

Come già analizzato nel capitolo 5 ChatGPT si rivela un ottimo e potente tool di supporto per la sua facilità di utilizzo, anche se non sempre efficace. Nel particolare si è riscontrato che la sua maggior efficacia durante la fase di implementazione, in quanto i requisiti, particolarmente quelli funzionali, possono essere molto soggettivi in base a come si vuole sviluppare l'applicazione e al pubblico al quali si intende rivolgersi. Inoltre è già più efficace nella fase di analisi dei requisiti non funzionali dove può fornire delle linee guida generali molto utili anche

se sempre da verificare.

Nella fase di implementazione nello specifico si è riscontrato il funzionamento migliore soprattutto utilizzando una sola conversazione e non aprendone una nuova ogni volta. Molto spesso le risposte sono corrette e anche il codice fornito spesso è funzionante e ben scritto, come già detto soprattutto dopo un po' di messaggi nella conversazione.

Durante il Testing invece è dove si sono riscontrati il maggior numero di errori ricevendo anche più frequentemente codice, anche se sintatticamente corretto, non semanticamente e che non produceva il risultato desiderato.

In conclusione ChatGPT non si considera ancora in grado di svolgere completamente il lavoro di sviluppo nel caso almeno di sistemi complessi come lo è appunto una applicazione, tuttavia rappresenta un ottimo strumento di supporto per sviluppatori che hanno solide basi, essendo in grado di fornire alcune porzioni di codice e quindi velocizzando lavori che possono risultare ripetitivi e tediosi o anche fornendo consigli utili.

Bibliografia e Sitografia

[1] developer.android.com, Android Studio, https://developer.android.com/studio/intro?hl=it.

[2] developer.android.com, Overview Activity Android, https://developer.android.com/reference/android/app/Activity.

[3] developer.android.com, Overview Fragment Android, https://developer.android.com/guide/fragments?hl=it.

[4] developer.android.com, Overview LiveData Android, https://developer.android.com/topic/libraries/architecture/livedata.

[5] developer.android.com, Dashboard di distribuzione delle versioni di Android, https://developer.android.com/about/dashboards?hl=it.

[6] developer.android.com, Room—Jetpack, https://developer.android.com/jetpack/androidx/releases/room?hl=it#groovy.

[7] developer.android.com, Searchview, https://developer.android.com/reference/android/widget/SearchView.

[8] developer.android.com, RecyclerView, https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.

[9] developer.android.com, SearchView.OnQueryTextListener, https://developer.android.com/reference/kotlin/androidx/appcompat/widget/SearchView.OnQueryTextListener.

[10] developer.android.com, Adapter, https://developer.android.com/reference/android/widget/Adapter.

[11] developer.android.com, ImageView, https://developer.android.com/reference/android/widget/ImageView.

[12] developer.android.com, TextView, https://developer.android.com/reference/android/widget/TextView.

[13] developer.android.com, TableLayout, https://developer.android.com/guide/topics/ui/layout/grid?hl=it.

[14] developer.android.com, CardView, https://developer.android.com/reference/androidx/cardview/widget/CardView.

[15] developer.android.com, ImageButton, hhttps://developer.android.com/reference/android/widget/ImageButton.

[16] developer.android.com, Espresso , https://developer.android.com/training/testing/espresso.

[17] developer.android.com, ViewModel, https://developer.android.com/reference/androidx/lifecycle/ViewModel.

[18] developer.android.com, ViewModelProvider.Factory, https://developer.android.com/reference/androidx/lifecycle/ViewModelProvider.Factory.

[19] developers.google.com, Google code scanner (Andorid only), https://developers.google.com/ml-kit/vision/barcode-scanning/code-scanner.

[20] Eco-score Consortium, Documentazione Eco-score, https://docs.score-environnemental.com/v/en

[21] Google, Linee guida del Material Design, https://m3.material.io.

[22] json.org, Introduzione a JSON, https://www.json.org/json-it.html.

[23] kotlinlang.com, Kotlin for Android, https://kotlinlang.org/docs/android-overview.html.

[24] openfoodfacts.org, Documentazione OpenFoodFactsAPI, https://world.openfoodfacts.org/files/api-documentation.html.

[25] Santé publique France, Nutri-score, https://www.santepubliquefrance.fr/en/nutri-score.

[26] square.github.com, Retrofit a type-safe HTTP client for Android and Java, https://square.github.io/retrofit/.

[27] wikipedia.com, Architettura Model-View-ViewModel, https://en.wikipedia.org/wiki/Modelviewviewmodel.

[28] wikipedia.com, Loose coupling, https://en.wikipedia.org/wiki/Loose_coupling#cite_ref-1.

[29] wikipedia.com, Cohesion in computer science, https://en.wikipedia.org/wiki/Cohesion_(computer_science).

[30] wikipedia.com, Java(programming language), https://en.wikipedia.org/wiki/Java_(programming_language).