

UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea triennale in Ingegneria Elettronica



TESI DI LAUREA

**Design e sviluppo di una applicazione multiplatforma per la
condivisione e il riconoscimento automatico di immagini attraverso
intelligenza artificiale**

**Design and development of a cross-platform application for sharing and
automatic recognition of images through artificial intelligence**

Relatore:

Dott. Francesco Cauteruccio

Correlatore:

Dott. Enrico Corradini

Candidato:

Marco Di Vita

Indice

Introduzione	1
Capitolo 1: Background tecnologico	3
1.1 Framework	3
1.1.1 Flutter	4
1.2 Linguaggio di programmazione	5
1.2.1 Dart	6
1.3 IDE	7
1.3.1 Android Studio	7
1.4 AppGallery Connect	8
1.4.1 Auth Service	9
1.4.2 Cloud DB	10
1.4.3 Cloud Storage	10
Capitolo 2: Progettazione dell'applicazione	12
2.1 Overview generale dell'applicazione	13
2.2 Fase di autenticazione	14
2.2.1 Registerloginpage	15
2.2.2 Sign-Up	16
2.2.3 Log-In	18
2.3 Gestione interfaccia principale	20
2.3.1 HomePage	21
2.4 Classi Utili	24
2.4.1 Cestino	24
2.4.2 Impostazioni	26
2.4.3 Guida e tutorial	27
Capitolo 3: Implementazione dell'applicazione	28
3.1 Logica dell'applicazione	28
3.1.1 Main Activity	29
3.1.2 DBManager	31
3.2 Feature principali	33
3.2.1 Auth service	33
3.2.2 Classificazione immagine e Object detection	35

3.2.3 Cloud Storage	38
Capitolo 4: Discussione dei risultati ottenuti	40
4.1 Impatto dell'applicazione	40
4.2 Utilità dell'applicazione	41
4.3 Lesson Learned	42
Conclusioni	43
Bibliografia	44

Introduzione

Nell'era digitale in cui viviamo, la possibilità di usare le immagini ha acquisito un ruolo di primaria importanza nella comunicazione e nell'espressione personale. Con lo sviluppo di dispositivi mobili avanzati e la diffusione di piattaforme di social media, la condivisione di immagini è diventata una pratica comune per milioni di persone in tutto il mondo. Tuttavia, con il crescente volume di immagini condivise quotidianamente, processi come la gestione e l'organizzazione di questo enorme flusso di dati visivi costituiscono una sfida complessa.

In risposta a questa sfida, il presente lavoro di tesi si concentra sulla progettazione e lo sviluppo di un'applicazione multiplatforma per la condivisione e il riconoscimento automatico di immagini attraverso l'utilizzo dell'intelligenza artificiale (IA). L'obiettivo principale è fornire agli utenti uno strumento potente ed efficiente per gestire le proprie collezioni di immagini, semplificando la ricerca, l'organizzazione e la condivisione di contenuti visivi.

L'applicazione proposta sfrutta le recenti evoluzioni nell'ambito dell'apprendimento automatico e dell'IA per implementare algoritmi di riconoscimento e classificazione di immagini. Grazie a questi algoritmi, l'applicazione è in grado di analizzare automaticamente le immagini caricate dagli utenti, identificando oggetti, persone, luoghi e altri elementi presenti nelle foto. Ciò consentirà agli utenti di cercare e organizzare le proprie immagini in base a criteri specifici.

Inoltre, l'applicazione è dotata di funzionalità di condivisione, consentendo agli utenti di condividere le proprie immagini con amici e familiari attraverso link (dedicato alle singole immagini o alle cartelle). In questo modo, si promuove la condivisione di contenuti tra utenti con interessi simili.

Lo sviluppo di un'applicazione di questo tipo richiede una combinazione di topic quali user experience, progettazione e implementazione di algoritmi di intelligenza artificiale, nonché lo sviluppo di applicazioni multiplatforma. Questa tesi si propone di affrontare questi aspetti, fornendo una solida base di progettazione e mettendo in pratica le competenze acquisite per realizzare un prototipo funzionante dell'applicazione proposta.

Lo scopo principale di questa tesi, e dell'app in essa sviluppata, è individuare

nell'utilizzo dell'intelligenza artificiale nell'ambito della gestione delle immagini una significativa ottimizzazione delle attività di organizzazione, ricerca e condivisione delle immagini.

Questa tesi si articola in diverse sezioni:

Inizialmente, si descrivono le tecnologie utilizzate. Successivamente, si descrive il design dell'applicazione proposta, discutendo le scelte di progettazione e l'architettura del sistema. Infine, vengono presentati i risultati dell'implementazione del prototipo e saranno valutate le sue prestazioni e il suo impatto nella società moderna.

Capitolo 1: Background tecnologico

Con il termine “background tecnologico” si intende la conoscenza e la comprensione delle tecnologie, degli strumenti, dei linguaggi di programmazione e dei concetti fondamentali necessari per sviluppare e implementare un'applicazione o un progetto tecnologico.

Nel contesto dello sviluppo software, il background tecnologico comprende familiarità con linguaggi di programmazione come Java, C++, Python, JavaScript, Swift, Kotlin, Ruby, tra gli altri.

Il background tecnologico può anche comprendere la comprensione dei concetti di base dell'architettura software, come la programmazione orientata agli oggetti, il design pattern, l'architettura Client-Server, l'architettura a microservizi e l'architettura senza server (Serverless).

Un solido background tecnologico è essenziale per comprendere e utilizzare le tecnologie a disposizione in modo efficace e per adattarsi ai continui cambiamenti nel campo dello sviluppo software. Essa fornisce le basi per affrontare le sfide nello sviluppo di applicazioni e per adottare nuove tecnologie emergenti.

1.1 Framework

Un framework è un'infrastruttura di supporto che fornisce una serie di librerie, strumenti e convenzioni per lo sviluppo di software. In altre parole, è un'architettura di base che consente agli sviluppatori di creare applicazioni più facilmente e in modo più efficiente, fornendo una struttura predefinita per organizzare, creare e gestire il codice. Sono progettati per risolvere problemi comuni nello sviluppo di software, fornendo un insieme di funzionalità pronte all'uso. Queste funzionalità possono includere librerie per gestire operazioni di base come l'input/output, la gestione della memoria e la connettività di rete, nonché moduli per la gestione dell'interfaccia utente, la sicurezza, la gestione dei dati e molto altro. Il loro utilizzo può semplificare lo sviluppo del software in quanto offre un'architettura stabilita, una serie di best practice e un'astrazione dei dettagli tecnici complessi. I framework possono anche fornire una maggiore produttività, consentendo agli sviluppatori di concentrarsi maggiormente sulla logica dell'applicazione piuttosto che sulle questioni tecniche di basso livello.

Ci sono molti tipi di framework, ad esempio:

- Framework di sviluppo web: questi framework forniscono un'infrastruttura per lo sviluppo di applicazioni web, semplificando l'implementazione di funzionalità come la gestione delle richieste e delle risposte HTTP, la gestione delle sessioni, l'accesso al database e la generazione di pagine web dinamiche. Esempi comuni includono Django per Python, Ruby on Rails per Ruby e Express.js per JavaScript.
- Framework di sviluppo mobile: questi framework sono progettati per lo sviluppo di applicazioni per dispositivi mobili, consentendo agli sviluppatori di creare app che possono essere eseguite su più piattaforme come iOS e Android. Esempi noti includono React Native, Xamarin e Flutter.
- Framework di sviluppo di giochi: questi framework offrono strumenti e risorse per lo sviluppo di giochi, come la gestione dei grafici, la fisica dei giochi, l'audio e l'interazione con l'utente. Alcuni esempi sono Unity per lo sviluppo di giochi 2D e 3D e Cocos2d per lo sviluppo di giochi per dispositivi mobili.

1.1.1 Flutter

Flutter è un framework open-source sviluppato da Google per la creazione di applicazioni mobili native per diverse piattaforme, come iOS, Android, web e desktop.



Figura 1 Logo Flutter [1]

È stato introdotto per la prima volta nel 2017 e si basa sul linguaggio di programmazione Dart. Una caratteristica distintiva è il concetto di "single codebase", che significa che si può scrivere il codice dell'applicazione una volta sola e poi eseguirlo su diverse piattaforme senza doverlo adattare o riscrivere per ognuna di esse. Ciò consente agli sviluppatori di risparmiare tempo e sforzi nello sviluppo delle app multipiattaforma.

Ecco alcune caratteristiche chiave di Flutter:

- Interfaccia utente reattiva e personalizzabile: Offre un'ampia gamma di widget predefiniti e personalizzabili che consentono di creare interfacce utente accattivanti e fluide. Puoi creare facilmente widget personalizzati per adattare

l'aspetto e il comportamento dell'applicazione alle tue esigenze specifiche.

- **Rendering veloce:** Utilizza il suo motore di rendering per disegnare direttamente l'interfaccia utente sui dispositivi, garantendo prestazioni fluide e rapide. Questo approccio elimina la necessità di utilizzare i componenti di interfaccia nativi delle piattaforme sottostanti.
- **Hot Reload:** Una delle funzionalità più apprezzate. Permette agli sviluppatori di visualizzare immediatamente le modifiche apportate al codice durante lo sviluppo, senza dover ricompilare l'intera app. Questo consente un iterativo processo di sviluppo e testing rapido.
- **Architettura basata su widget:** Segue un'architettura basata su widget, in cui ogni elemento dell'interfaccia utente è considerato un widget. Questo approccio consente una facile composizione dei widget per creare gerarchie complesse e facilita la gestione dello stato dell'applicazione.
- **Ampia libreria di pacchetti:** Offre una vasta gamma di pacchetti e librerie open-source che possono essere utilizzati per aggiungere funzionalità extra all'applicazione, come l'integrazione con API esterne, l'accesso al database, la gestione dello stato globale e molto altro.

Flutter è diventato sempre più popolare tra gli sviluppatori per lo sviluppo di app cross-platform, consentendo loro di creare app dall'aspetto nativo e di elevata qualità per diverse piattaforme con un singolo codice base. [1]

1.2 Linguaggio di programmazione

Un linguaggio di programmazione è un insieme di regole e sintassi che definiscono come scrivere istruzioni che un computer può comprendere ed eseguire. Consentono agli sviluppatori di scrivere algoritmi e codice sorgente che vengono successivamente tradotti in istruzioni comprensibili dalla macchina, consentendo così di creare programmi e software.

Ogni linguaggio di programmazione ha le sue caratteristiche, la sua sintassi e le sue librerie specifiche. Alcuni sono più adatti per determinati scopi, come lo sviluppo di applicazioni web o l'analisi dei dati, mentre altri sono più generici e possono essere utilizzati in diversi contesti.

Scegliere il linguaggio di programmazione più adatto dipende dalle esigenze del

progetto, dalla familiarità degli sviluppatori e da altri fattori come la comunità di supporto, la disponibilità di librerie e strumenti, e le prestazioni richieste.

1.2.1 Dart

Dart è un linguaggio di programmazione open source sviluppato da Google. È stato creato come linguaggio principale per lo sviluppo delle applicazioni Flutter, ma può essere utilizzato anche per altri scopi come lo sviluppo di applicazioni web o server-side.



Figura 2 Logo Dart [2]

Le caratteristiche principali di Dart includono:

- **Sintassi moderna:** Dart utilizza una sintassi familiare e intuitiva, che facilita l'apprendimento e la scrittura del codice. Supporta elementi come classi, ereditarietà, tipi generici, funzioni di ordine superiore e altro ancora.
- **Tipizzazione statica opzionale:** Dart offre un sistema di tipizzazione statica opzionale. Ciò significa che gli sviluppatori possono scegliere se specificare o meno i tipi delle variabili. La tipizzazione statica può aiutare a rilevare errori durante la compilazione e migliorare le prestazioni dell'esecuzione del codice.
- **Gestione dello stato:** Dart offre meccanismi avanzati per la gestione dello stato, come le parole chiave `final` e `const` per le variabili immutabili, il supporto per i getter e setter, e la possibilità di utilizzare il pattern di design Observer per reagire ai cambiamenti di stato.
- **Librerie e pacchetti:** Dart dispone di un'ampia gamma di librerie e pacchetti che coprono una vasta gamma di casi d'uso, come l'elaborazione delle immagini, la gestione delle richieste HTTP, la serializzazione dei dati, la manipolazione di stringhe, la creazione di interfacce utente e molto altro ancora.
- **Performance:** Dart è stato progettato per offrire prestazioni elevate. Utilizza un compilatore just-in-time (JIT) durante lo sviluppo e un compilatore ahead-of-time (AOT) per produrre codice altamente ottimizzato per la distribuzione. [2]

1.3 IDE

Un IDE (Integrated Development Environment) è un software che fornisce un ambiente completo per lo sviluppo di software. È uno strumento che aiuta gli sviluppatori a scrivere, modificare, testare e debuggare il codice in modo più efficiente. Combina diversi componenti e strumenti che rendono il processo di sviluppo più integrato e agevole.

L'obiettivo principale di un IDE è semplificare il processo di sviluppo del software, fornendo strumenti e funzionalità integrate in un unico ambiente. Ciò aiuta gli sviluppatori a risparmiare tempo, ridurre gli errori e migliorare la qualità complessiva del codice.

1.3.1 Android Studio

Android Studio è l'ambiente di sviluppo integrato (IDE) ufficiale per lo sviluppo di applicazioni Android. È stato sviluppato da Google e offre un insieme completo di strumenti e funzionalità per la scrittura, il debug, il test e il rilascio di applicazioni Android.

Ecco alcune caratteristiche principali di Android Studio:

- **Editor di codice avanzato:** Android Studio fornisce un editor di codice ricco di funzionalità, che supporta il completamento automatico, la formattazione del codice, la navigazione rapida tra le classi e i file, il refactoring del codice e molto altro. L'editor di codice è specificamente progettato per lo sviluppo di applicazioni Android e offre suggerimenti e strumenti specifici per il framework Android.
- **Emulatore e dispositivo virtuale:** Android Studio include un emulatore Android che consente di eseguire e testare le applicazioni su diversi dispositivi Android virtuali. Puoi simulare varie configurazioni hardware, risoluzioni dello schermo e versioni del sistema operativo Android per testare l'applicazione su una vasta gamma di dispositivi virtuali.
- **Strumenti di debug avanzati:** Android Studio offre potenti strumenti di debug che consentono di individuare e risolvere i bug nelle applicazioni Android. Puoi

eseguire il debug del codice passo dopo passo, monitorare le variabili, esaminare la traccia dello stack e utilizzare il debugger visuale per individuare errori di rendering nell'interfaccia utente.

- **Integrazione con Gradle:** Android Studio utilizza Gradle come sistema di build per le applicazioni Android. Gradle semplifica la gestione delle dipendenze, la compilazione del codice sorgente, la generazione di file APK e altre attività di build. Android Studio offre strumenti per configurare e personalizzare il processo di build dell'applicazione tramite file di configurazione Gradle.
- **Integrazione con altre librerie e strumenti:** Android Studio supporta l'integrazione con una vasta gamma di librerie e strumenti di terze parti. Puoi utilizzare librerie esterne per aggiungere funzionalità specifiche all'applicazione e strumenti come Firebase per l'integrazione dei servizi cloud di Google, strumenti di test e altro ancora.



Figura 3 Logo Android Studio [3]

Android Studio è uno strumento potente e completo per lo sviluppo di applicazioni Android. Fornisce un ambiente integrato che semplifica il processo di sviluppo, testing e distribuzione delle applicazioni, offrendo strumenti e funzionalità specificamente progettati per l'ecosistema Android. [3]

1.4 AppGallery Connect

AppGallery Connect è una piattaforma di sviluppo e distribuzione delle applicazioni fornita da Huawei. È progettata per aiutare gli sviluppatori a creare, pubblicare e gestire le proprie app per i dispositivi Huawei, inclusi smartphone, tablet, smartwatch e altri dispositivi basati su HarmonyOS.

AppGallery Connect offre una serie di funzionalità per semplificare il processo di sviluppo e distribuzione delle app.



HUAWEI

Figura 4 Logo di Huawei [4]

Alcune delle principali caratteristiche includono:

- **Creazione e distribuzione delle app:** Gli sviluppatori possono registrarsi su AppGallery Connect per creare un account sviluppatore e iniziare a caricare le proprie app. La piattaforma fornisce strumenti per la gestione delle versioni, la firma delle app e la pubblicazione su AppGallery, il negozio di app di Huawei.
- **Analytics e monitoraggio delle prestazioni:** AppGallery Connect offre funzionalità di analisi e monitoraggio delle prestazioni delle app. Gli sviluppatori possono accedere a dati dettagliati sulle installazioni, gli utenti attivi, i crash e altre metriche importanti per valutare le prestazioni e migliorare le app.
- **Servizi cloud:** La piattaforma fornisce servizi cloud per consentire agli sviluppatori di integrare funzionalità come la gestione degli utenti, i servizi di autenticazione, i servizi di pagamento e altro ancora nelle proprie app.
- **Gestione delle risorse:** Gli sviluppatori possono utilizzare AppGallery Connect per gestire le risorse delle loro app, come immagini, video, documentazione e risorse multimediali.
- **Strumenti di promozione e monetizzazione:** La piattaforma offre strumenti per la promozione delle app, come campagne pubblicitarie, coupon e sconti. Inoltre, supporta modelli di monetizzazione come gli acquisti in-app e la pubblicità per consentire agli sviluppatori di guadagnare dalle proprie app. [4]

1.4.1 Auth Service

Il servizio di autenticazione (Auth Service) aiuta a costruire rapidamente un sistema di autenticazione utente sicuro e affidabile per la tua app, integrando direttamente le funzionalità del servizio di autenticazione basato su cloud nella tua app.

Il servizio di autenticazione fornisce un SDK e servizi di backend, supporta diverse

modalità di autenticazione e offre una potente console di gestione, che ti permette di sviluppare e gestire facilmente l'autenticazione degli utenti.

L'SDK del servizio di autenticazione supporta diverse piattaforme e linguaggi di programmazione e consente agli utenti di godere di un'esperienza di accesso coerente utilizzando la stessa identità utente su vari tipi di dispositivi.

Come serverless, il servizio di autenticazione può adattarsi automaticamente ad altri servizi senza server. Puoi proteggere la sicurezza dei dati degli utenti nei servizi senza server definendo semplici regole di sicurezza. [5]

1.4.2 Cloud DB

Cloud DB è un servizio di database che sfrutta la sinergia tra dispositivo e cloud, fornendo la gestione dei dati tra dispositivo e cloud, modelli di dati unificati e varie API per la gestione dei dati. Oltre a garantire la disponibilità, l'affidabilità, la coerenza e la sicurezza dei dati, Cloud DB consente la sincronizzazione dei dati in modo trasparente tra dispositivo e cloud e fornisce il supporto offline per le app, aiutandoti a sviluppare rapidamente app per la sinergia tra dispositivo e cloud e per l'uso su più dispositivi. Come servizio di AppGallery Connect, Cloud DB costruisce la capacità di Mobile Backend as a Service (MBaaS) per AppGallery Connect. In questo modo, è possibile concentrarsi sullo sviluppo dell'app, migliorando notevolmente l'efficienza produttiva. [6]

1.4.3 Cloud Storage

Cloud Storage ti consente di archiviare grandi quantità di dati, come immagini, audio, video e altri contenuti generati dagli utenti, in modo sicuro ed economico.

Il SDK di Cloud Storage fornito per vari client presenta i seguenti vantaggi per il caricamento e il download dei file:

- **Sicurezza robusta:** I dati vengono trasmessi tramite HTTPS e i file vengono crittografati e archiviati nel cloud utilizzando protocolli di crittografia sicuri.
- **Trasferimento riprendibile:** È possibile riprendere i caricamenti o i download dal punto in cui si è verificato un'interruzione di rete o un'errata operazione durante il caricamento o il download.
- **Scalabilità elevata:** Cloud Storage è progettato per la gestione di exabyte di dati

quando l'app deve archiviare una grande quantità di dati.

- Facile gestione e manutenzione: È possibile individuare facilmente la causa di un errore in base al codice di risultato. [7]

Capitolo 2: Progettazione dell'applicazione

La progettazione di un'applicazione è il processo di definizione e pianificazione dettagliata di come l'applicazione sarà strutturata, organizzata e funzionerà. Si tratta di tradurre i requisiti e le specifiche dell'applicazione in un design tecnico che guidi l'implementazione del software.

Due aspetti fondamentali di cui bisogna tener conto prima della progettazione di una applicazione:

- **Scopo e obiettivi dell'applicazione:** Prima di iniziare la progettazione, è fondamentale avere una chiara comprensione dello scopo dell'applicazione e dei suoi obiettivi. Definire quali problemi o necessità dell'utente l'applicazione mira a risolvere e quali funzionalità o valore aggiunto fornirà.
- **Ricerca dell'utente:** È essenziale condurre una ricerca sull'utente per comprendere il pubblico target dell'applicazione. La comprensione degli utenti aiuta a guidare le decisioni di progettazione per creare un'esperienza utente efficace e soddisfacente.

La progettazione di un'applicazione richiede una combinazione di competenze creative, di comprensione degli utenti e di padronanza delle best practice di progettazione.

La progettazione dell'applicazione coinvolge diversi aspetti, tra cui:

- **Architettura del software:** La progettazione dell'architettura definisce la struttura generale dell'applicazione, inclusi i componenti principali, i moduli, le interazioni e i flussi di dati. Vengono identificati i pattern architetturali appropriati, come l'architettura a strati, l'architettura a microservizi o l'architettura MVC (Model-View-Controller).
- **Progettazione dell'interfaccia utente (UI):** Questa fase coinvolge la creazione di un'interfaccia utente intuitiva e user-friendly. Vengono definite le schermate, i layout, i controlli e le interazioni che consentono agli utenti di interagire con l'applicazione in modo efficace.
- **Progettazione delle funzionalità:** Si tratta di definire le diverse funzionalità e i casi d'uso dell'applicazione. Vengono identificati i requisiti specifici di ogni funzionalità e si decide come implementarle in modo efficiente.

2.1 Overview generale dell'applicazione

L'applicazione offre agli utenti la possibilità di caricare immagini dal proprio dispositivo, consentendo loro di selezionare le foto desiderate e importarle nell'app. Queste immagini possono essere organizzate in cartelle per una migliore gestione e accesso e attraverso l'utilizzo di un menu a comparsa, gli utenti possono scegliere tra varie funzionalità.

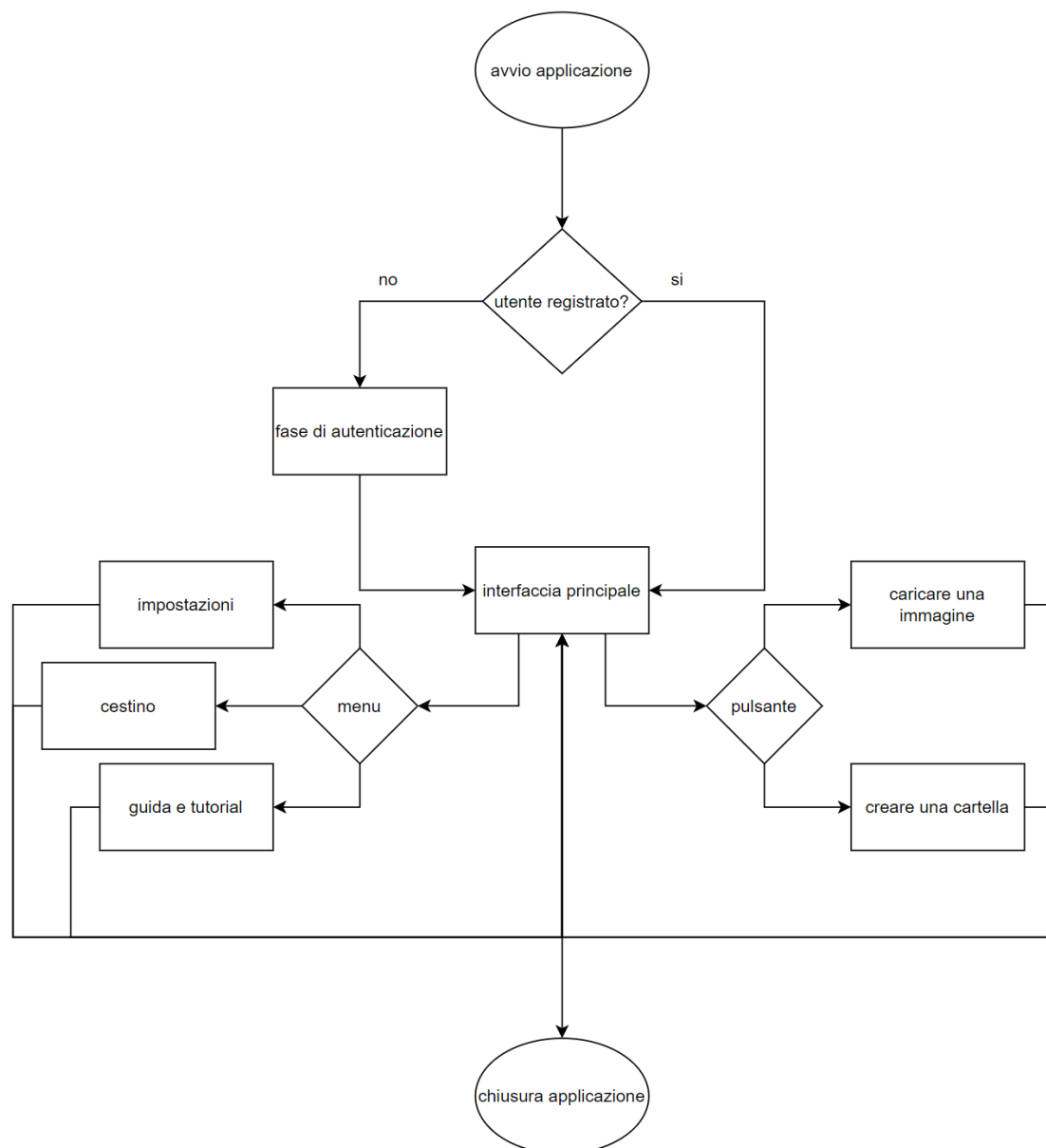


Figura 5 Workflow applicazione

Di seguito viene descritta la progettazione delle parti principali dell'applicazione, composte da: l'autenticazione, la schermata principale e le tre classi utili.

2.2 Fase di autenticazione

La prima schermata che viene mostrata alla prima installazione è una pagina per la gestione dell'accesso. Questa permette all'utente di decidere se registrarsi o accedere al proprio account. Cliccando uno dei due bottoni, infatti, l'utente verrà direttamente indirizzato alle pagine che gestiscono queste due operazioni. È stato scelto un design semplice ed intuitivo per far valere il concetto di user friendly (tutto quello che è di facile utilizzo anche per chi non è esperto).

All'interno di questa prima fase vengono utilizzati, per la progettazione, strumenti analoghi. Di seguito alcune delle principali:

- `TextEditingController`: È una classe fornita da Flutter che consente di gestire l'input di testo in un campo di testo (`TextField`) e di accedere al testo inserito dall'utente. Viene utilizzato per collegare un campo di testo a un controller, consentendo di ottenere o impostare il testo presente nel campo di testo, nonché di eseguire altre operazioni sul testo, come salvarlo e confrontarlo. [8]
- `Container`: È un widget che consente di creare un contenitore rettangolare con la possibilità di personalizzarne l'aspetto e il comportamento. Può essere utilizzato per raggruppare altri widget all'interno di un layout, fornendo loro proprietà di stile, come ad esempio il colore di sfondo, il bordo, il padding e il margine. Inoltre, può essere utilizzato per definire le dimensioni del widget interno o per imporre vincoli sulle dimensioni massime o minime. [9]
- `MaterialButton`: È un widget che rappresenta un pulsante materiale seguendo le linee guida del Material Design, il design system sviluppato da Google. Un `MaterialButton` è un pulsante reattivo che risponde agli input dell'utente e offre diversi stili predefiniti. Offre anche altre proprietà per personalizzare l'aspetto e il comportamento del pulsante, come "elevation" per aggiungere un'ombra, "disabledColor" per impostare un colore quando il pulsante è disabilitato e "padding" per regolare il "padding" interno del pulsante. [10]
- `Scaffold`: È un widget fondamentale per la creazione di schermate e interfacce utente. Rappresenta la struttura base di una schermata. [11]
- `Navigator.push`: È un metodo che consente di navigare da una schermata

all'altra. È molto utilizzato per gestire la navigazione tra le schermate. [12]

2.2.1 Registerloginpage

La prima delle tre classi (insieme di metodi e attributi per la gestione delle informazioni) che vengono analizzate in questo capitolo è `registerloginpage`.

Questa classe, una `StatelessWidget`, (è statica, non ha uno stato interno che può essere modificato) contiene la schermata principale e il suo compito principale è di reindirizzare l'utente alla scelta da lui effettuata tramite il comando "Navigator.push" che permette di spostarsi tra "rotte" all'interno di Flutter.

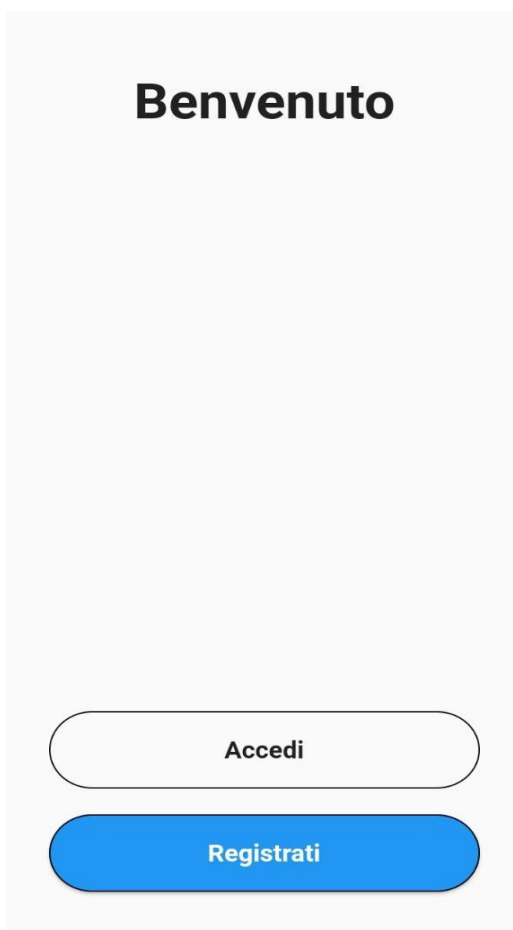


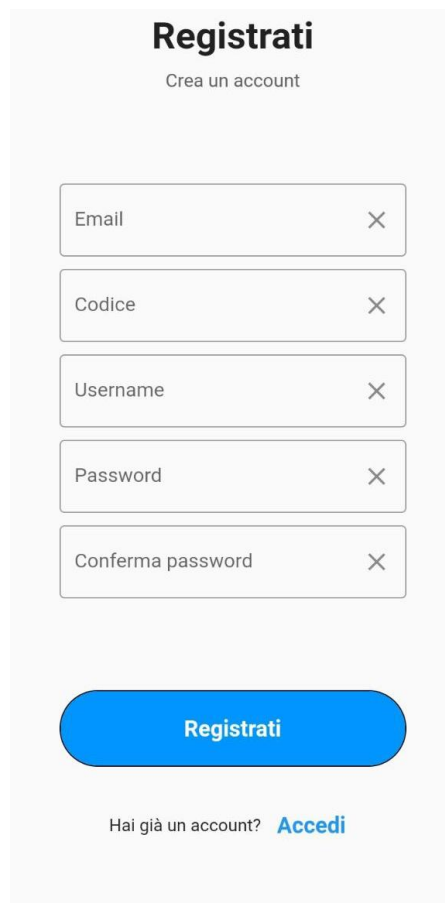
Figura 6 Schermata al primo avvio dell'applicazione

Le tre classi analizzate vengono implementate tutte secondo la stessa logica: Vengono usati uno Scaffold e un Container per contenere tutte le informazioni e per impostare parametri importanti, come la spaziatura, il margine e il colore dello sfondo. Viene utilizzato, per ogni parametro richiesto dall'utente, il `TextEditingController` che

permette di salvare il valore che l'utente inserisce all'interno del TextField. Infine, per la conferma e il passaggio alle operazioni successive viene utilizzato un MaterialButton.

2.2.2 Sign-Up

La prima pagina che viene visualizzata contiene un campo di testo per l'inserimento dell'e-mail da parte dell'utente. Per la gestione dell'accesso tramite servizi Huawei, infatti, la registrazione che avviene tramite e-mail, deve essere accompagnata da un codice di verifica. Se il campo è vuoto, viene visualizzata una finestra di dialogo che avvisa l'utente di non aver inserito alcuna e-mail.



The screenshot shows a registration form with the following elements:

- Title: **Registrati**
- Subtitle: Crea un account
- Input fields: Email, Codice, Username, Password, Conferma password (each with a clear button 'X')
- Registration button: Registrati (blue)
- Footer: Hai già un account? [Accedi](#)

Figura 7 Schermata di registrazione

Altrimenti, viene chiamato il metodo `sendEmailCode` del `DBManager` (in poche parole, il gestore delle funzioni di logica, viene spiegato nel dettaglio nel capitolo successivo) per inviare il codice di verifica all'email scritta dall'utente che viene reindirizzato alla pagina `Sign-Up`. Questa pagina contiene campi di testo per

l'inserimento dell'e-mail, del codice di verifica, dello username, della password e la conferma della password. Prima di poter effettuare la registrazione, vengono effettuate alcune verifiche sui campi inseriti: se uno qualsiasi dei campi (e-mail, codice, username o password) è vuoto, viene visualizzata una finestra di dialogo che avvisa l'utente di non aver inserito correttamente i parametri richiesti. Inoltre, se la password ha meno di 8 caratteri, viene visualizzata una finestra di dialogo che avvisa l'utente che deve essere di almeno 8 caratteri e successivamente viene verificato se le due password inserite corrispondano. Dopo aver superato tutte le verifiche, viene chiamato il metodo `registerEmail` del `DBManager` per registrare l'utente utilizzando l'e-mail, il codice di verifica e la password inseriti. Una volta terminata la registrazione l'utente viene reindirizzato alla pagina principale, pronto per usufruire delle possibilità proposte.

2.2.3 Log-In

La classe `LoginPage` è una `StatelessWidget` che rappresenta la schermata di accesso dell'applicazione. All'interno del suo metodo, è presente, come per gli altri, uno `Scaffold` che contiene tutte le informazioni necessarie per eseguire l'accesso al proprio account. Per l'accesso l'utente deve fornire nel `TextField` l'e-mail e la password utilizzate al momento della registrazione. Queste vengono salvate e, quando viene premuto il pulsante di accesso, vengono eseguite alcune verifiche sui campi di testo e viene chiamato il metodo `loginPassword` della classe `DBManager` per gestire la logica di accesso. Se il login ha successo, l'app viene indirizzata alla schermata `HomePage2`, la schermata vera e propria per l'utilizzo dell'applicazione, altrimenti viene visualizzato un messaggio di errore.

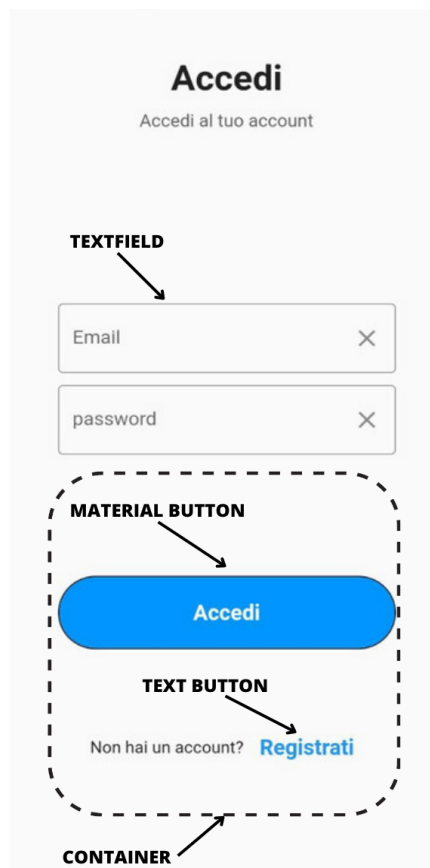


Figura 8 Schermata di accesso

In fondo alla schermata è presente una riga che contiene un `TextButton` (un testo cliccabile che permette di seguire azioni). Questo bottone serve per l'utente in caso di password dimenticata, va infatti a richiamare una funzione, `code`, che restituisce uno

Scaffold per la schermata di richiesta del codice di reset della password. È simile alla fase di richiesta del codice per la registrazione, ma viene chiamata una funzione di logica diversa, che fornisce all'utente un codice specifico per il recupero della password. Successivamente viene utilizzato il widget `resetPassword`, una funzione che restituisce una schermata di reset contenente campi di testo per l'e-mail, il codice, la nuova password e la conferma di quest'ultima. Infine, tramite il pulsante, l'utente può convalidare la sua richiesta di reset, potendo così accedere ai suoi dati, che verranno usufruiti per la gestione dell'applicazione.

2.3 Gestione interfaccia principale

Dopo la fase di autenticazione, l'utente viene rimandato alla pagina principale dell'applicazione. Questa pagina, e tutte le possibilità che offre, sono racchiuse nella classe `Homepage2`. Questa interfaccia è stata sviluppata in modo da essere il più intuitiva ed efficiente possibile. È infatti ben visibile un bottone per caricare immagini o creare cartelle.

In cima all'interfaccia è presente una barra del menu, contenente diverse opzioni per la navigazione e la gestione delle funzioni chiave dell'applicazione: Impostazioni, Guida e Tutorial.

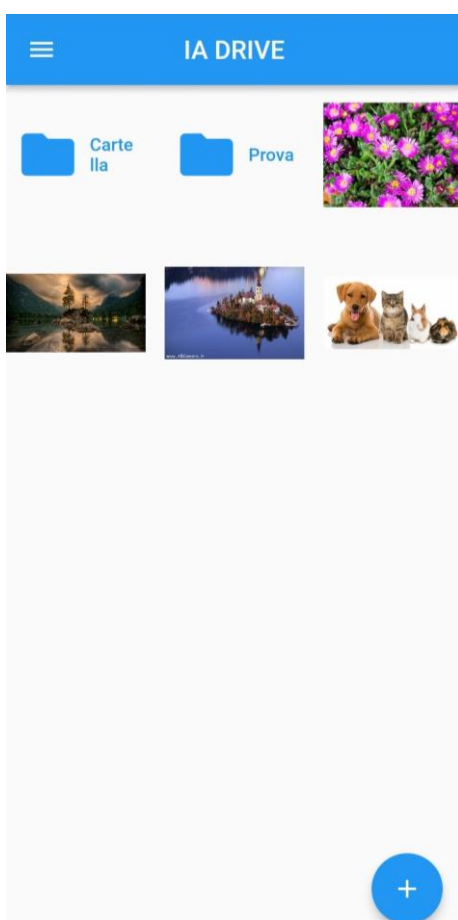


Figura 9 Esempio di schermata principale

Nella parte centrale dell'interfaccia, è presente una griglia da tre elementi, per riga per la visualizzazione dei file e delle cartelle. Le cartelle sono rappresentate da icone con il loro nome associato, mentre le immagini vengono mostrate come miniature. Questa visualizzazione consente agli utenti di accedere rapidamente alle cartelle e di identificare le immagini senza sforzo.

L'utente può interagire con gli elementi a schermo: selezionare una cartella per aprirla o tenere premuto su un'immagine per eliminarla.

2.3.1 HomePage

La classe che si occupa della gestione dell'interfaccia principale è `homepage`. All'interno di questa classe vengono racchiuse tutte le possibilità che l'applicazione offre.

La schermata principale viene gestita tramite uno scaffold che va a contenere i 3 elementi fondamentali della schermata: il bottone, le cartelle e le immagini.

In basso a destra dello schermo è presente un `Floating Action Button` (un bottone che compare elevato rispetto agli altri elementi all'interno dello schermo). Quando premuto viene fornita all'utente la possibilità di scegliere tra due opzioni: creare una cartella o caricare una foto.

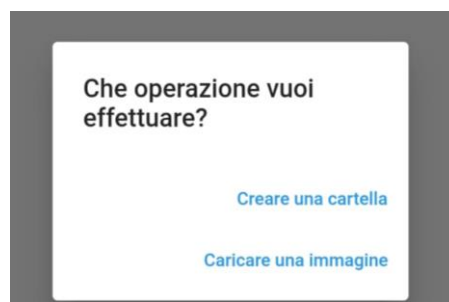


Figura 10 Pop-Up di scelta operazione

Se l'utente decide di creare una cartella deve indicare anche il nome con cui desidera salvarla. Questo viene gestito tramite un `TextEditingController` e salvato su un array (una struttura di dati che può contenere un insieme ordinato di dati dello stesso tipo). L'utilizzo di uno strumento simile permette di semplificare operazioni necessarie utili per lo sviluppo di un'applicazione, in particolare:

- Accesso in qualsiasi momento ad elementi specifici
- Inserimento di nuovi elementi senza eliminare quelli precedenti
- Modificare o eliminare specifici elementi lasciando gli altri inalterati
- Facilitazione del ripristino degli elementi eliminati
- Grazie all'uso dell'array viene mostrata a schermo un'icona a forma di cartella per ogni elemento presente al suo interno, rendendo semplice la gestione degli

elementi a schermo.

Se l'utente decide invece di caricare una foto, viene utilizzato il metodo `pickImage`:

```
Future pickImage() async {
  final List selectedImages = await ImagePicker().pickMultiImage();
  if (selectedImages!.isNotEmpty) {
    imageFileList!.addAll(selectedImages);
  }
  setState((){
    imageFileList;
  });
}
```

Figura 11 Codice flutter per la gestione del metodo "pickImage"

Questo metodo, direttamente implementato dal sito flutter [14], consente di selezionare una o più immagini dalla galleria del telefono e salvarle tutte all'interno dell'array `imageFileList`. Quando una delle immagini presenti nella schermata vengono cliccate viene chiamato `ZoomImage`.

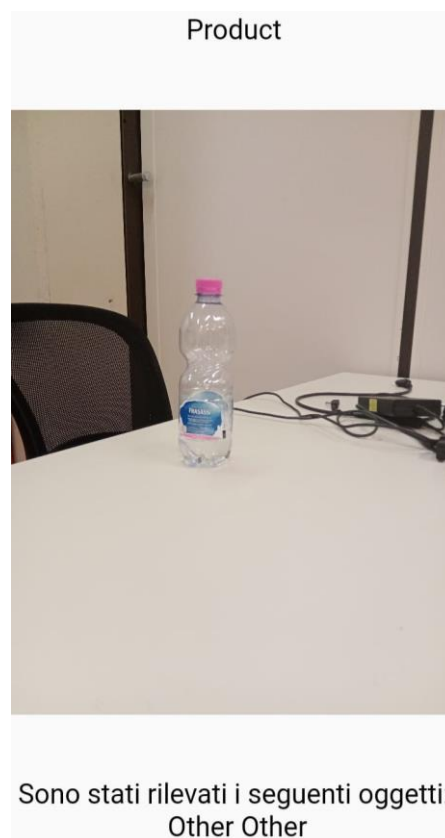


Figura 12 Esempio di schermata ottenibile con "ZoomImage"

Questo metodo ha bisogno di una immagine di riferimento e restituisce uno scaffold contenente tre cose: la classificazione, l'immagine ingrandita e l'object detection.

In questo modo l'utente può, con un semplice click, ottenere tutte le informazioni

chiave che l'immagine ha da offrire, oltre alla possibilità di “pizzicare” (il famoso pinch-to-zoom) per ingrandirla ulteriormente. L'analisi e l'object detection utilizzano procedure analoghe al caso di Log-in e Sign-up, vengono infatti chiamati due metodi della classe DBManager che mi restituiscono in formato testuale, le informazioni ottenute dall'analisi dell'immagine. Ritornando alla schermata principale, in alto a sinistra, di fianco al nome dell'applicazione, sono presenti tre linee che, se cliccate, danno accesso ad un drawer (una componente visuale che si apre lateralmente o dal basso dello schermo e fornisce un menu o una serie di azioni che l'utente può eseguire).

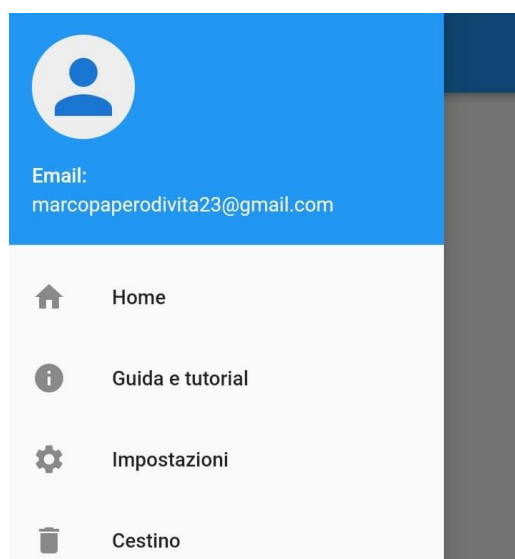


Figura 13 Menu a comparsa

Questa interfaccia viene gestita dalla classe MenuPage. Per la visualizzazione delle opzioni del menu vengono utilizzati i ListTile. Ognuno di questi ha un'icona e un testo che descrivono l'opzione di schermata associata e se premuti vanno ad aggiornare un parametro di tipo intero. Questo indice opera un ruolo fondamentale all'interno del programma, perché si preoccupa di aggiornare la schermata visualizzata. Tramite uno switch (una struttura di controllo che consente di eseguire azioni diverse in base al valore di una variabile) viene visualizzata una opzione diversa in base a cosa abbiamo scelto nel menù. Questo tipo di procedura permette di rendere più veloce e veloce il passaggio da una schermata all'altra, in base all'operazione che desidera compiere.

2.4 Classi utili

Di seguito viene descritta l'implementazione e la struttura delle "Classi utili". Queste sono progettate con lo scopo di semplificare determinate operazioni, senza andare a scrivere all'interno della stessa porzione di codice, in modo da rendere il tutto di più semplice scrittura e correzione.

2.4.1 Cestino

Lo scopo della classe Cestino è molto semplice: ogni qualvolta un utente elimina una foto, deve essere disponibile all'interno di questa schermata per poter essere recuperata o eliminata definitivamente. Per fare ciò, nella schermata principale, all'interno della griglia che contiene tutte le foto presenti a schermo, viene implementato un `GestureDetector` (banalmente un "rilevatore di gesti"), permettendomi di eseguire una determinata azione al rilevamento di uno specifico gesto. In particolare, quando l'utente tiene premuto su una foto, si andrà ad eliminare dall'array che contiene le immagini eliminate a schermo, l'elemento selezionato, che viene contemporaneamente salvato sull'array `deleteImage`.

```
onLongPress: () async {
  setState(() {
    deleteImage.add(i);
    imageFileList.remove(i);
  });
},
```

Figura 14 Codice che gestisce l'eliminazione di una foto

In questo modo, infatti, ogni qualvolta viene selezionata l'opzione Cestino, viene chiamata l'omonima classe passando come parametro l'array degli elementi cancellati. La classe Cestino ha struttura analoga ad altre classi sopra esposte: Restituisce uno Scaffold che visualizza una griglia contenente tutte le immagini presente nell'array `deleteImage` e, quando viene premuto il bottone posto in basso a destra, tutti gli elementi vengono definitivamente eliminati.

La particolarità di questa Classe è la funzione `Restore`:

Se un utente vuole ripristinare una foto eliminata, è sufficiente tenerla premuta per ritrovarla all'interno della schermata principale. Per fare ciò, è scritta una funzione apposita che viene implementata cestino e aggiornata ogni volta che viene rilevato il gesto "long Press" su un'immagine.

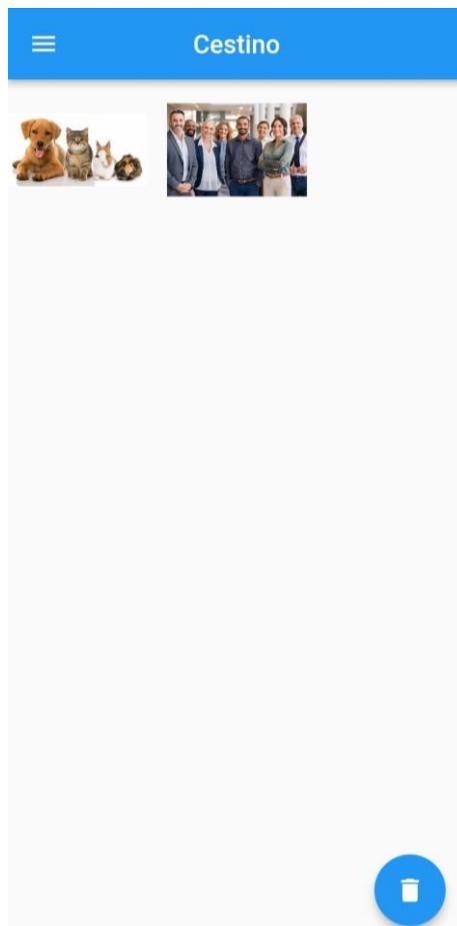


Figura 15 Esempio di schermata del Cestino

Venendo passata come parametro ogni volta che viene chiamata la classe `Cestino`, infatti, la stessa funzione va ad aggiornare l'array delle immagini presente nella `HomePage`, rendendo di nuovo visibili le immagini nella schermata principale.

2.4.2 Impostazioni

Una classe di particolare interesse è quella riguardante le Impostazioni. Sono contenute infatti alcune operazioni che posso essere di particolare interesse per l'utente, come il cambio della password o della e-mail. Le opzioni che l'utente può scegliere sono in totale quattro e sono rappresentate tramite un ListTile con il titolo dell'operazione.

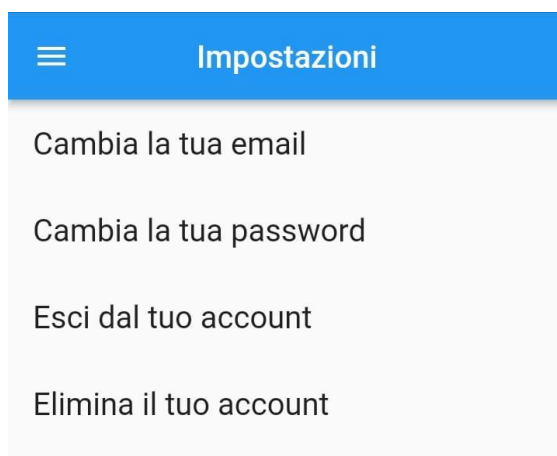


Figura 16 Schermata impostazioni

Ogni qualvolta che ne viene cliccato uno, tramite un Navigator.push, passiamo alla fase di ri-autenticazione con un indice particolare. Essendo l'applicazione basata sul servizio "Auth Service" di Huawei, per compiere operazioni delicate, è necessario fornire di nuovo e-mail e password con i quali è avvenuta la registrazione. Questa operazione è del tutto analoga alla fase di Log-In: tramite dei TextField e dei TextEditingController chiediamo e salviamo i dati necessari, per passarli alla funzione di ri-autenticazione nel DBManager. Essendo questa un'operazione essenziale, viene eseguita in modo automatico indipendentemente dall'opzione scelta. Grazie all'introduzione di uno switch, che analizza l'indice menzionato precedentemente, riusciamo a distinguere l'operazione iniziale scelta dall'utente. Nei due casi di cambio, all'utente viene chiesto di inserire la sua e-mail alla quale viene inviato un codice di conferma, successivamente viene visualizzata la pagina per l'inserimento della nuova e-mail, nel caso in cui volesse cambiarla o, nel caso in cui volesse cambiare la password, viene visualizzata la schermata per l'inserimento della suddetta. Le altre due opzioni sono per la cancellazione dell'account in modo permanente e il Log-Out. Per la prima opzione viene chiesta un'ulteriore conferma da parte dell'utente, per evitare errori accidentali e permanenti.

Qualsiasi sia l'opzione che l'utente desidera, tutta la logica verrà gestita tramite il DBManager, chiamando la funzione e passando i dati di interesse e, successivamente alla chiamata al DBManager, viene eseguito un Navigator.push che riporta l'utente alla schermata principale.

```
DBManager.changePass(  
    email.text, password.text, codice.text).then((  
    value) {  
        Navigator.push(context, MaterialPageRoute(  
            builder: (context) =>  
                HomePage2(),  
        ), // MaterialPageRoute
```

Figura 17 Codice che gestisce la chiamata a DBManager per il cambio password

2.4.3 Guida e tutorial

L'ultima opzione selezionabile nel menù è la guida. Questa è sicuramente la classe più semplice ma più utile per l'utente. Contiene infatti tutte le informazioni necessarie per capire il funzionamento dell'applicazione e di tutte le sue possibilità. La sua struttura è molto semplice:

Restituisce un container con un insieme ripetuto di Text contenenti delle varie sezioni di spiegazioni relative ad un determinato argomento, come la creazione di una cartella, l'eliminazione di una immagine o il suo ripristino.

Capitolo 3: Implementazione dell'applicazione

Il termine “implementazione di una applicazione” si riferisce al processo di trasformare un'idea o un concept di un'applicazione in una forma concreta e funzionante. In altre parole, è la fase in cui si scrive il codice e si realizzano le funzionalità e le caratteristiche dell'applicazione.

Durante l'implementazione, si traducono le specifiche in codice sorgente, utilizzando il linguaggio di programmazione appropriato per la piattaforma di destinazione (ad esempio, Flutter per lo sviluppo cross-platform, Kotlin per Android, Swift per iOS, etc.). Questo processo coinvolge la scrittura di algoritmi, la definizione delle classi, la gestione delle interazioni con l'interfaccia utente e l'integrazione con servizi esterni o API.

Comprende anche l'organizzazione del codice in moduli o componenti, la gestione delle dipendenze, la scrittura di test unitari per garantire la qualità del software e il debugging per individuare e risolvere eventuali errori o bug.

Durante l'implementazione, bisogna tenere conto di diversi aspetti, come la progettazione dell'interfaccia utente, l'architettura del software, le performance, la sicurezza e la compatibilità con le diverse versioni dei sistemi operativi o delle piattaforme di destinazione.

L'obiettivo finale dell'implementazione è quello di creare un'applicazione funzionante che risponda alle esigenze e alle specifiche stabilite durante la fase di progettazione. Una buona implementazione si caratterizza per la corretta traduzione dei requisiti in codice, la scalabilità, la manutenibilità e l'efficienza dell'applicazione.

3.1 Logica dell'applicazione

Si riferisce alla parte del codice o del software che implementa le regole, le operazioni e il flusso di lavoro dell'applicazione stessa. È la componente che definisce come l'applicazione deve elaborare i dati, interagire con l'utente, gestire gli eventi e fornire il comportamento desiderato.

La logica di un'applicazione può includere:

- **Elaborazione dei dati:** La logica definisce come i dati vengono elaborati, manipolati e trasformati. Ciò può includere la validazione dei dati, il calcolo di

risultati basati sui dati di input, la manipolazione dei dati in risposta a determinate azioni o eventi, la generazione di report o la preparazione dei dati per la visualizzazione.

- **Gestione del flusso di lavoro:** La logica controlla il flusso di lavoro dell'applicazione, determinando l'ordine in cui vengono eseguite le diverse operazioni e azioni. Può includere la gestione delle transizioni tra le schermate, il coordinamento delle operazioni asincrone, il controllo delle autorizzazioni e la gestione degli errori.
- **Interazione con l'utente:** La logica definisce come l'utente interagisce con l'applicazione e come l'applicazione risponde agli input dell'utente. Ciò può includere la gestione degli eventi utente come il tocco, il clic o la pressione di un pulsante, la gestione degli input da tastiera o da altri dispositivi di input, la presentazione di notifiche o avvisi all'utente e la gestione dei feedback utente.
- **Gestione degli errori:** le strategie e le regole che definiscono come l'applicazione gestisce le situazioni di errore o le eccezioni che possono verificarsi durante l'esecuzione. Ciò può includere la notifica degli errori agli utenti, il recupero dai fallimenti o la registrazione di informazioni di diagnostica per facilitare la risoluzione dei problemi.

3.1.2 MainActivity

Flutter utilizza il concetto di "plug-in" per accedere alle funzionalità native del sistema operativo, come fotocamera, sensori o API di terze parti. Quando si desidera utilizzare una funzionalità nativa che non è accessibile direttamente attraverso le API di Flutter, è possibile creare un plug-in personalizzato che funga da ponte tra il codice Dart e il codice nativo, il `MethodChannel`. Questo metodo è uno dei meccanismi disponibili per comunicare tra il codice Dart e il codice nativo all'interno di un plug-in Flutter. Consente di invocare metodi definiti nel codice nativo dal codice Dart e viceversa. Il codice Dart può inviare richieste al codice nativo tramite il `Method Channel`, specificando il nome del metodo e, se necessario, i parametri associati. Il codice nativo, una volta ricevuta una richiesta, può eseguire l'azione corrispondente e inviare una risposta al codice, se necessario. In questo modo, è possibile passare dati e informazioni tra il codice Dart e il codice nativo, consentendo un'interazione bidirezionale tra le due

parti. La gestione di questa procedura è gestita attraverso un metodo all'interno del MainActivity. Il primo parametro, `dartExecutor`, indica il `binaryMessenger`, che è responsabile della comunicazione tra Dart e la piattaforma nativa. Il secondo parametro "CHANNEL" rappresenta il nome del canale dei metodi. Viene chiamato il metodo `setMethodCallHandler` sull'oggetto `MethodChannel` appena creato. Questo metodo imposta un gestore di chiamate di metodi per il canale dei metodi, in modo che possa gestire le chiamate in arrivo dal codice Dart. Per gestire queste chiamate viene utilizzata una struttura di controllo "when" per determinare l'azione da intraprendere in base al metodo chiamato.

```
MethodChannel(flutterEngine!!.dartExecutor.binaryMessenger, CHANNEL).setMethodCallHandler {
    call, result ->
    when (call.method) {
        "registerEmail" -> {
            val email = call.argument<String>( key: "email")!!
            val code = call.argument<String>( key: "code")!!
            val password = call.argument<String>( key: "password")!!
            registerEmail(email, code, password){
                uid ->
                result.success(uid)
            }
        }
    }
}
```

Figura 18 Codice Kotlin per la gestione del MethodChannel

Andando a considerare l'esempio del metodo per la registrazione della e-mail, se il valore di `call.method` corrisponde a `registerEmail`, viene eseguito il blocco di codice associato a questa condizione. All'interno di questo blocco di codice, vengono estratti i parametri richiesti, ovvero "email", "code" e "password", utilizzando il metodo `call.argument`.

```
final String uid = await _channel.invokeMethod('registerEmail',{
    'email':email,
    'code': code,
    'password': password,
});
```

Figura 19 Codice Flutter per la chiamata del metodo "registerEmail" su Kotlin

Viene quindi chiamato il metodo `registerEmail` passando i valori estratti come argomenti. Viene creato un oggetto `emailUser` utilizzando il costruttore

EmailUser.Builder(), impostando i valori come parametri.

Viene chiamato il metodo createUser dell'istanza di AGConnectAuth per creare un nuovo account utente utilizzando l'oggetto emailUser.

```
private fun registerEmail(email: String, code: String, password: String, done: (String) -> Unit) {
    val emailUser = EmailUser.Builder() EmailUser.Builder
        .setEmail(email) EmailUser.Builder!
        .setVerifyCode(code)
        .setPassword(password)
        .build()
    AGConnectAuth.getInstance().createUser(emailUser).addOnSuccessListener { it: SignInResult!
        // A newly created user account is automatically signed in to your app.
        Log.i( tag: "AUTH", it.user.email)
        Log.i( tag: "AUTH", it.user.uid)
        done(it.user.uid)
    }.addOnFailureListener { it: Exception!
        // onFail
        done("")
    }
}
```

Figura 20 Codice Kotlin che gestisce la registrazione dell'e-mail tramite servizi Huawei

Viene utilizzato un listener addOnSuccessListener per gestire il caso in cui la creazione dell'account utente avvenga con successo. In questo caso, viene chiamata la funzione “done” con l'UID dell'utente come parametro. Nel caso in cui si verifichi un errore durante la creazione dell'account dell'utente, viene utilizzato il listener addOnFailureListener e viene chiamata la funzione “done” passandole una stringa vuota come parametro.

3.1.2 DBManager

La classe DBManager permette la gestione dei dati e l'accesso ai metodi presenti nel MainActivity.

```
static const MethodChannel _channel = MethodChannel('it.ppndrd.text_explorer.huawei');
```

Figura 21 Codice per collegare Flutter al codice nativo

All'inizio della classe, viene creato un oggetto MethodChannel chiamato _channel che rappresenta un canale di comunicazione tra il codice Flutter e un'implementazione nativa specifica. Una volta che il canale viene istanziato, può essere utilizzato per inviare richieste ai metodi definiti nell'implementazione nativa e ricevere

le risposte corrispondenti.

Prendendo nuovamente come esempio il metodo per la registrazione dell'account tramite e-mail:

```
static Future<String> registerEmail(String email, String code, String password) async {
  try {
    final String uid = await _channel.invokeMethod('registerEmail',{
      'email':email,
      'code': code,
      'password': password,
    });
    return uid;
  } on PlatformException catch (e) {
    print("Failed to sign in: ${e.message}");
    return "";
  }
}
```

Figura 22 Metodo registerEmail presente nel DBManager

È una funzione asincrona (una modalità di esecuzione del codice in cui le operazioni possono essere avviate in modo non bloccante e l'esecuzione può proseguire senza attendere il completamento di tali operazioni) che si occupa di registrare una e-mail all'interno del sistema Huawei e che restituisce un oggetto "Future<String>". Questo metodo ha bisogno di tre parametri in ingresso: e-mail, codice e password. Questo perché il metodo per la registrazione dell'account necessita i medesimi parametri per eseguire la sua funzione che gli vengono passati proprio dal metodo presente nel DBManager. Quando il codice nativo completa l'operazione richiesta, viene restituito una stringa corrispondente all'identificatore univoco associato all'utente registrato.

```
DBManager.uid = await DBManager.registerEmail(
  mail.text, codice.text,
  password.text);
```

Figura 23 Esempio di chiamata al metodo registerEmail nel DBManager

Questo valore viene quindi restituito come risultato del metodo registerEmail.

Se si verifica un'eccezione di tipo PlatformException (è un tipo specifico di eccezione che viene lanciata in Flutter quando si verifica un errore durante la

comunicazione tra il codice Dart e il codice nativo di una piattaforma specifica) durante l'invocazione del metodo, viene presa e gestita. Viene stampato un messaggio di errore e viene restituita una stringa vuota ("") come risultato. Questo metodo, una volta implementato correttamente all'interno della classe `DBManager` è pronto per essere chiamato ogni qualvolta sia necessario.

3.2 Feature principali

Il termine “feature principali” indica le caratteristiche più importanti o rilevanti di un prodotto, servizio o sistema. Rappresentano gli elementi essenziali su cui basa il valore l'utilità del prodotto o del servizio. Le feature principali dell'applicazione studiata in questa tesi sono tre: Auth service, Classificazione immagini e object detection ed infine il Cloud Storage. Questi sono gli elementi che caratterizzano l'applicazione e sono implementati grazie alle procedure scritte nelle guide Huawei, descritte in seguito.

3.2.1 Auth Service

Le fasi principali dell'Auth Service sono due: la registrazione e l'accesso. Essendo la prima già stata trattata e analizzata nel paragrafo precedente, in questa parte si analizza il processo di implementazione della fase di accesso.

```
private fun loginPassword(email: String,password: String, done:(String) -> Unit){
    val credential = EmailAuthProvider.credentialWithPassword(email, password)
    Log.i("EMAIL", email)
    Log.i("PASSWORD", password)
    AGConnectAuth.getInstance().signIn(credential).addOnSuccessListener {
        // Obtain sign-in information.
        done(it.user.uid)
    }.addOnFailureListener {
        done("")
    }
}
```

Figura 24 Codice Kotlin per l'implementazione della funzione di log-in tramite password

Il metodo utilizzato è una funzione di login che utilizza l'e-mail e la password come credenziali per autenticare l'utente e restituisce tre parametri: "email" e "password" di tipo String e "done" (La funzione "done" viene utilizzata per notificare il chiamante della funzione di login sull'esito dell'operazione).

Viene creato un oggetto di tipo `credential` utilizzando il metodo statico `EmailAuthProvider.credentialWithPassword` che restituisce un'istanza di `EmailAuthCredential`, utilizzata per autenticare l'utente tramite e-mail e password. Viene utilizzato l'oggetto `AGConnectAuth` per eseguire il login chiamando il metodo `signIn` con il parametro `credential` appena creato. Questo avvia il processo di autenticazione dell'utente e tramite un listener viene gestito il successo dell'operazione. Quando l'autenticazione ha successo, viene chiamata la funzione "done" passando l'ID dell'utente autenticato come argomento. Se il login fallisce, viene gestito l'errore utilizzando `addOnFailureListener`. Quando si verifica un errore, viene chiamata la funzione "done" passando una stringa vuota come argomento.

```
static Future<String> loginPassword(String email, String password) async {
  try {
    final String uid = await _channel.invokeMethod('loginPassword',{
      'email':email,
      'password': password,
    });
    return uid;
  } on PlatformException catch (e) {
    print("Failed to login the pass: ${e.message}");
    return "";
  }
}
```

Figura 25 Codice nel DBManager che mi permette di chiamare la funzione di log-in

Andando ad analizzare l'implementazione sul `DBManager`:

Viene utilizzato un metodo asincrono chiamato `loginPassword`. Il metodo ha bisogno di due parametri di tipo `String`: "email" e "password", rappresentando l'e-mail e la password fornite per il login. Viene utilizzato il metodo `_channel.invokeMethod` per invocare il metodo denominato `loginPassword` presente nel `MainActivity` passando un oggetto mappa contenente gli argomenti. Se l'invocazione ha successo, viene restituito un valore di tipo `String` chiamato "uid" (l'ID utente). In caso di errore, viene catturata un'eccezione di tipo `PlatformException` mediante il blocco `catch`. L'errore viene stampato utilizzando il messaggio dell'eccezione e restituendo una stringa vuota come risultato.

3.2.2 Classificazioni immagini e object detection

La classificazione dell'immagine è un processo che permette di determinare la classe o la categoria a cui un'immagine appartiene. In altre parole, si tratta di assegnare un'etichetta o una categoria predefinita a un'immagine in base al suo contenuto visivo.

L'object detection, o rilevamento degli oggetti, è una tecnica nell'ambito dell'elaborazione delle immagini e dell'apprendimento automatico che consiste nel riconoscere e localizzare gli oggetti all'interno di un'immagine o di un video. A differenza della classificazione delle immagini che assegna una singola etichetta a un'intera immagine, l'object detection identifica e individua le posizioni degli oggetti specifici all'interno di un'immagine. Nonostante le due definizioni presentino delle differenze, a livello di implementazione ci sono molte similitudini. In particolare, essendo che i metodi presenti nel MainActivity hanno bisogno di ByteArray, bisogna fare una conversione:

```
File image = File(i.path);  
List<int> imageBytes = await image.readAsBytes();  
String label = await DBManager.getLabel(imageBytes) ;  
List<String> detection = await DBManager.objectDetection(imageBytes) ;
```

Figura 26 Codice che permette la conversione da un file immagine ad una lista di interi

Viene creato un oggetto File chiamato image con il percorso del file immagine passato come argomento i.path. Questo passaggio serve per creare un'istanza della classe 'File' che rappresenta l'immagine da elaborare. Viene letto il contenuto dell'immagine come un array di byte utilizzando il metodo readAsBytes() della classe File. Questo metodo restituisce un oggetto di tipo List<int> contenente i byte dell'immagine. Una volta ottenuta questa lista, si possono chiamare i metodi getLabel e objectDetection presenti nel DBManager.

Questi due metodi sono implementati nella stessa maniera e presentano una unica differenza: il valore che restituiscono. Questo perché la classificazione restituisce una stringa contenente la "label" dell'immagine mentre l'object detection una lista di oggetti contenenti nella foto.

Le differenze di implementazioni si trovano nel MainActivity. Partendo con l'implementazione della classificazione dell'immagine:

```
private fun imageClassificationAnalyzer(imageBytes: ByteArray, done: (String)-> Unit){
    val bitmap = BitmapFactory.decodeByteArray(imageBytes, 0, imageBytes.size)
    val analyzer = MLAnalyzerFactory.getInstance().localImageClassificationAnalyzer
    val frame = MLFrame.fromBitmap(bitmap)
    val task = analyzer!!.asyncAnalyseFrame(frame)
    task.addOnSuccessListener { res ->
        // Recognition success.
        val app = ArrayList(res)
        app.sortByDescending { it.possibility }
        done(app.first().name)
    }.addOnFailureListener { e ->
        // Recognition failure.
        // Recognition failure.
        done("")
    }
}
```

Figura 27 Codice Kotlin per l'implementazione dell'ImageClassification

Viene creata una funzione chiamata `imageClassificationAnalyzer` che utilizza un'immagine rappresentata come un array di byte per eseguire un'analisi di classificazione. La funzione prende due parametri: `imageBytes`, un array di byte che rappresenta l'immagine, e `done`, per rappresentare l'esito dell'operazione. Viene utilizzato il metodo `BitmapFactory.decodeByteArray` per decodificare l'array di byte `imageBytes` in un oggetto `Bitmap`, questo rappresenta l'immagine che verrà analizzata. Viene creato un'istanza di un analizzatore di classificazione delle immagini locale utilizzando `MLAnalyzerFactory.getInstance`. L'analizzatore viene utilizzato per analizzare l'immagine e classificarla in diverse categorie o classi. Viene creato un oggetto "frame" di tipo `MLFrame` utilizzando l'oggetto `Bitmap` appena creato. `MLFrame` rappresenta il frame dell'immagine che verrà analizzato. Viene eseguita l'analisi asincrona del frame utilizzando il metodo `asyncAnalyseFrame` dell'analizzatore. Questo avvia il processo di analisi dell'immagine, dove viene aggiunto un listener per gestire il successo dell'analisi utilizzando `addOnSuccessListener`. Quando l'analisi ha successo, i risultati dell'analisi vengono ordinati in base alla probabilità di appartenenza alle diverse classi. Viene quindi chiamata la funzione "done" passando il nome della prima classe più probabile come argomento. Se l'analisi fallisce, viene gestito l'errore utilizzando `addOnFailureListener`. Viene chiamata

la funzione "done" passando una stringa vuota per indicare un'analisi non riuscita.
Per quanto riguarda l'implementazione dell'object detection:

```
private fun objectDetection(imageBytes: ByteArray, done: (String)-> Unit){
    val bitmap = BitmapFactory.decodeByteArray(imageBytes, 0, imageBytes.size)
    // Use MLObjectAnalyzerSetting.TYPE_PICTURE for static image detection.
    val setting = MLObjectAnalyzerSetting.Factory() MLObjectAnalyzerSetting.Factory!
        .setAnalyzerType(MLObjectAnalyzerSetting.TYPE_PICTURE) MLObjectAnalyzerSetting.Factory!
        .allowMultiResults()
        .allowClassification()
        .create()
    val analyzer = MLObjFactory.getInstance().getLocalObjectAnalyzer(setting)
    // Create an MLFrame object using the bitmap, which is the image data in bitmap format.
    val frame = MLFrame.fromBitmap(bitmap)
    // Create a task to process the result returned by the object detector.
    val task = analyzer!!.asyncAnalyseFrame(frame)
    // Asynchronously process the result returned by the object detector.
    task.addOnSuccessListener { res ->
        val types: List<String> = res.map { it: MLObject!
            when(it.typeIdentity) {
                MLObject.TYPE_OTHER -> "Other" ^map
                MLObject.TYPE_FACE -> "Face" ^map
                MLObject.TYPE_FOOD -> "Food" ^map
                MLObject.TYPE_FURNITURE -> "Furniture" ^map
                MLObject.TYPE_PLACE -> "Place" ^map
                MLObject.TYPE_PLANT -> "Plant" ^map
                MLObject.TYPE_GOODS -> "Goods" ^map
                else -> "No match" ^map
            }
        }
        done(types.joinToString( separator: ", "))
    }.addOnFailureListener { it: Exception!
        // Detection failure.
        done("")
    }
}
```

Figura 28 Codice Kotlin per l'implementazione dell'object detection

Viene creata una funzione chiamata `objectDetection` che utilizza un'immagine rappresentata come un array di byte per eseguire un'analisi di rilevamento degli oggetti localmente. Inizialmente la funzione è analoga a quella della classificazione. Le differenze iniziano quando viene creato un oggetto "setting" di tipo `MLObjectAnalyzerSetting` utilizzando il factory method della classe `MLObjectAnalyzerSetting.Factory()`. Viene impostato il tipo di analizzatore come `MLObjectAnalyzerSetting.TYPE_PICTURE` per il rilevamento degli oggetti in immagini statiche. Viene creato un'istanza di un

analizzatore di oggetti locali utilizzando `MlAnalyzerFactory.getInstance`, che viene utilizzato per analizzare l'immagine e rilevare gli oggetti presenti. Viene creato un oggetto "frame" di tipo `MlFrame`, utilizzando l'oggetto `Bitmap`, che rappresenta il frame dell'immagine che verrà analizzato. Viene eseguita l'analisi asincrona del frame utilizzando il metodo `asyncAnalyseFrame` dell'analizzatore che avvia il processo di rilevamento degli oggetti nell'immagine. Viene aggiunto un listener per gestire il successo dell'analisi utilizzando `addOnSuccessListener`: Quando l'analisi ha successo, i risultati dell'analisi vengono mappati in una lista di stringhe corrispondenti al tipo di oggetto rilevato. Viene quindi chiamata la funzione "done" passando la lista di tipi di oggetti rilevati concatenati come una stringa separata da virgole. Se l'analisi fallisce, utilizzando `addOnFailureListener`, viene chiamata la funzione "done" passando una stringa vuota per indicare un'analisi non riuscita.

3.2.3 Cloud Storage

La feature che presenta maggiori differenze a livello di implementazione è il `CloudStorage`. Prima di poterlo utilizzare in un'applicazione, bisogna inizializzare un'istanza di storage.

```
val storageManagement = AGCStorageManagement.getInstance()
```

Figura 29 Metodo per inizializzare una istanza di storage

Per questo va chiamato il metodo `AGCStorageManagement.getInstance` per creare un oggetto di `AGCStorageManagement` e inizializzare l'istanza di storage predefinita. I file vengono archiviati nelle istanze di storage sul cloud. Dopo aver creato un riferimento ad un file, è possibile utilizzarlo per caricare, scaricare o eliminare il file, oppure aggiornarne i metadati. Inoltre, è anche possibile creare un riferimento a una directory e utilizzare questo riferimento per elencare tutti i file in questa directory. Per caricare un file nell'istanza bisogna utilizzare il metodo `AGCStorageManagement.getStorageReference` per creare un riferimento al file da caricare e caricarlo nel percorso pianificato sul cloud.

Per caricare i file nell'istanza bisogna chiamare il metodo `StorageReference.putFile`. Per specificare attributi personalizzati per un file da caricare, bisogna utilizzare il metodo `putFile()`.

```
val reference = storageManagement!!.getStorageReference("images/demo.jpg")
```

Figura 30 Codice per creare un riferimento al file

Nel caso in cui si volesse utilizzare il caricamento, è possibile utilizzare il metodo `putFile(File srcFile, FileMetadata attributes, Long offset)`, in cui `offset` è impostato a 0.

```
val task = reference.putFile(File("path/images/test.jpg"))
task.addSuccessListener(OnSuccessListener { })
task.addFailureListener(OnFailureListener { })
```

Figura 31 Codice per caricare un file nell'istanza di storage

Se il processo viene interrotto in modo anomalo, è possibile chiamare nuovamente questo metodo per riprendere il processo. Il caricamento riprende dalla posizione in cui si era interrotto il precedente. Il valore di `offset` viene ottenuto utilizzando il metodo `UploadTask.UploadResult.getBytesTransferred`.

Viene infine aggiunto un listener per gestire il successo del caricamento utilizzando `addSuccessListener` e, nel caso in cui fallisse, viene gestito l'errore utilizzando `addFailureListener`.

Capitolo 4: Discussione dei risultati ottenuti

All'interno di questo capitolo si discute dell'analisi, l'interpretazione e la valutazione dei risultati ottenuti durante lo sviluppo e la valutazione dell'applicazione stessa.

Uno degli obiettivi principali di questa applicazione è quello di realizzare un sistema in grado di riconoscere automaticamente gli oggetti presenti nelle immagini caricate dagli utenti, consentendo loro di condividerle in modo rapido e intuitivo.

Questa sezione della tesi è dedicata all'approfondimento dei risultati emersi durante la fase di sperimentazione e valutazione dell'applicazione. Iniziando con la parte riguardante la performance, grazie all'integrazione degli algoritmi di intelligenza artificiale forniti da Huawei, l'applicazione ha dimostrato di essere efficiente nel riconoscimento degli oggetti presenti nelle immagini. Questo significa che gli utenti possono fare affidamento sul sistema per identificare correttamente gli oggetti e le caratteristiche delle immagini caricate. In conclusione, l'applicazione si è dimostrata utile e promettente. I risultati ottenuti dimostrano il potenziale di questa tecnologia per semplificare la gestione delle immagini e offrire nuove opportunità di interazioni con il mondo visivo.

4.1 Impatto dell'applicazione

La valutazione dell'impatto è importante per comprendere l'efficacia e l'utilità della stessa e per prendere decisioni informate riguardo al suo sviluppo, miglioramento o adozione. Misurare l'impatto consente di valutare il valore generato dall'applicazione e i benefici che essa porta agli utenti, alle organizzazioni e alla società nel suo insieme.

Può essere considerato da diverse prospettive:

- **Impatto sugli utenti:** Si riferisce agli effetti che l'applicazione ha sugli utenti che la utilizzano. L'applicazione offre agli utenti un'esperienza più fluida e intuitiva nella gestione e condivisione delle immagini. Grazie all'uso dell'intelligenza artificiale, l'applicazione semplifica l'organizzazione delle foto, grazie all'analisi dell'immagine del suo contenuto. Ciò rende l'interazione con le immagini più efficiente e piacevole.
- **Impatto sociale:** Indica come l'applicazione influisce sulla società. L'applicazione favorisce la condivisione di esperienze visive, facilitando la

comunicazione e la connessione tra le persone. Le immagini hanno un potere emotivo e possono contribuire a creare un senso di appartenenza e di comunità.

- **Impatto tecnologico:** Si riferisce all'influenza dell'applicazione sullo sviluppo e l'adozione di nuove tecnologie. Lo sviluppo di un'applicazione che sfrutta l'intelligenza artificiale per il riconoscimento di immagini contribuisce anche all'avanzamento tecnologico in questo campo. In particolare, l'applicazione usufruisce dei servizi offerti da Huawei e parteciperà ad un concorso indetto da quest'ultima per sviluppatori.

4.2 Utilità dell'applicazione

L'utilità di un'applicazione si riferisce alla sua capacità di fornire benefici o valore agli utenti. Indica quanto l'applicazione sia utile, funzionale o significativa per l'utente nel soddisfare le sue esigenze, risolvere i problemi o migliorare l'esperienza.

I principali punti chiave di questa applicazione sono:

- **Condivisione delle immagini:** L'applicazione offre agli utenti la possibilità di condividere le proprie immagini con altri utenti, facilitando la comunicazione visiva e l'interazione sociale. Ciò favorisce l'espressione creativa, la connessione tra le persone e la diffusione di contenuti di valore.
- **Riconoscimento di immagini:** L'intelligenza artificiale viene utilizzata per identificare e riconoscere automaticamente gli oggetti, le persone o gli elementi presenti nelle immagini. Questa funzionalità semplifica la classificazione e l'organizzazione delle foto, facilitando la ricerca di immagini specifiche o l'identificazione di persone o oggetti di interesse.
- **Esperienza utente migliorata:** L'utilizzo dell'intelligenza artificiale per il riconoscimento di immagini migliora l'esperienza complessiva dell'utente. Inoltre, essendo l'applicazione sviluppata mantenendo costante il concetto di "semplicità d'uso", questo permette all'utente di usufruire di tutte le possibilità offerte in maniera semplice ed intuitiva.

4.3 Lesson learned

Il termine "lesson learned" (lezione appresa) si riferisce a una conoscenza o una comprensione acquisita attraverso un'esperienza o un'attività. Possono derivare da successi o fallimenti, da situazioni positive o negative, e sono spesso utilizzate per migliorare le prestazioni future o guidare decisioni più informate.

È stato investito tempo ed energie nello studio e nella ricerca per realizzare un'applicazione all'avanguardia che potesse semplificare la vita degli utenti e offrire loro un'esperienza unica nel campo della condivisione e del riconoscimento di immagini.

Durante il processo di sviluppo, si sono presentate sfide affascinanti e complesse:

È stato necessario padroneggiare varie tecnologie e strumenti per implementare correttamente l'intelligenza artificiale nell'applicazione, dedicando molte ore a comprendere i principi fondamentali dell'apprendimento automatico, dell'elaborazione delle immagini e dei modelli di riconoscimento.

Nonostante siano sorte molte difficoltà, ogni sfida ha rappresentato un'opportunità di crescita e apprendimento per imparare a gestire i problemi di prestazioni dell'applicazione, adottando strategie di ottimizzazione e migliorando l'efficienza dei processi. Oltre all'aspetto tecnico, è stato molto importante studiare il design dell'interfaccia utente, per creare un'esperienza utente intuitiva e coinvolgente, in modo che gli utenti potessero facilmente navigare nell'applicazione e sfruttarne tutte le funzionalità in modo fluido. È stato molto importante imparare a conoscere il mondo della "user experience" durante questo percorso:

- Comunicare in modo chiaro ed efficace per condividere le proprie idee e visioni.
- Adattarsi ai feedback e alle critiche costruttive, utilizzandoli come opportunità per migliorare continuamente il proprio lavoro.

In definitiva, realizzare questa applicazione ha aiutato ad accumulare un tesoro di conoscenze e competenze. È stato necessario superare sfide tecniche e umane, affrontare problemi complessi e a migliorarsi costantemente.

Conclusioni

In questa tesi, è stato affrontato il tema del design e dello sviluppo di un'applicazione innovativa per la condivisione e il riconoscimento di immagini attraverso l'intelligenza artificiale. L'obiettivo principale era creare un'esperienza utente unica, semplificando la condivisione di immagini e fornendo funzionalità avanzate di riconoscimento.

Durante il percorso di ricerca e sviluppo, sono state esplorate varie tecnologie e approcci nel campo dell'apprendimento automatico, dell'elaborazione delle immagini e dell'intelligenza artificiale. L'obiettivo era quello di implementare algoritmi avanzati per il riconoscimento delle immagini e garantire risultati accurati ed efficienti.

Sono state impiegate metodologie di design centrato sull'utente per comprendere le esigenze degli utenti e creare un flusso di lavoro fluido all'interno dell'applicazione.

Le lezioni apprese durante lo sviluppo di questa applicazione sono state molteplici. La gestione delle sfide tecniche, come l'implementazione di algoritmi complessi e l'ottimizzazione delle prestazioni, ha richiesto un'approfondita conoscenza delle tecnologie coinvolte.

In conclusione, questa tesi ha dimostrato l'importanza del design e dello sviluppo di un'applicazione per la condivisione e il riconoscimento di immagini attraverso l'intelligenza artificiale. Grazie all'adozione di metodologie di design centrato sull'utente e all'utilizzo di algoritmi avanzati, è stato possibile realizzare un'applicazione che offre agli utenti un modo intuitivo ed efficiente per gestire le proprie immagini. L'applicazione rappresenta un importante contributo nel campo dell'intelligenza artificiale e apre nuove prospettive per lo sviluppo di soluzioni innovative in questo settore in continua evoluzione.

Bibliografia

- [1] Flutter: <https://flutter.dev/brand> (ultima visita 27/06/2023)
- [2] Dart: <https://dart.dev/> (ultima visita 27/06/2023)
- [3] Android Studio: <https://developer.android.com/studio> (ultima visita 27/06/2023)
- [4] AppGallery:
<https://developer.huawei.com/consumer/en/service/josp/agc/index.html>, [AppGalleryConnect](https://developer.huawei.com/consumer/en/service/josp/agc/index.html) (ultima visita 27/06/2023)
- [5] AuthService: <https://developer.huawei.com/consumer/en/agconnect/auth-service/> (ultima visita 27/06/2023)
- [6] CloudDB: <https://developer.huawei.com/consumer/en/agconnect/cloud-base/> (ultima visita 27/06/2023)
- [7] CloudStorage: <https://developer.huawei.com/consumer/en/agconnect/cloud-storage/> (ultima visita 27/06/2023)
- [8] TextEditingController: <https://api.flutter.dev/flutter/widgets/TextEditingController-class.html> (ultima visita 27/06/2023)
- [9] Container: <https://api.flutter.dev/flutter/widgets/Container-class.html> (ultima visita 27/06/2023)
- [10] MaterialButton: <https://api.flutter.dev/flutter/material/MaterialButton-class.html> (ultima visita 27/06/2023)
- [11] Scaffold: <https://api.flutter.dev/flutter/material/Scaffold-class.html> (ultima visita 27/06/2023)
- [12] Navigator.push: <https://docs.flutter.dev/cookbook/navigation/navigation-basics> (ultima visita 27/06/2023)
- [13] Image_Picker: https://pub.dev/packages/image_picker (ultima visita 27/06/2023)
- [14] AccountReauthentication:
<https://developer.huawei.com/consumer/en/doc/development/AppGallery-connect-Guides/agc-auth-android-reauthenticate-0000001127287737> (ultima visita 27/06/2023)