Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

---

Tesi di Laurea

Progettazione e sviluppo di un'applicazione mobile in Flutter con Realtà Aumentata a supporto di attività formative

Design and development of a mobile application in Flutter with Augmented Reality to support educational activities

Relatore
Prof. Emanuele Storti

Candidato
Nicola Sebastianelli

Anno Accademico 2023-2024

Il vero viaggio di scoperta non consiste nel cercare nuove terre, ma nell'avere nuovi occhi.

Marcel Proust

Indice

Elenco delle figure

Introduzione

Nell'epoca digitale in cui il mondo è immerso, la tecnologia agisce come un catalizzatore per il cambiamento nei processi. L'aumento del numero di strumenti avanzati che si integrano nell'ambiente formativo crea nuove opportunità di apprendimento per gli studenti, permettendo loro di assimilare contenuti in modo altamente efficiente.

Questa tesi ha lo scopo di progettare e sviluppare un'applicazione mobile in Flutter basata sulla Realtà Aumentata (AR) [1], con l'obiettivo di arricchire l'esperienza educativa di qualsiasi studente. Per realtà aumentata si intende l'arricchimento della percezione sensoriale umana mediante informazioni, in genere manipolate e convogliate elettronicamente, che non sarebbero percepibili con i cinque sensi. L'integrazione della realtà aumentata con le tecnologie educative è un passo verso l'obiettivo di creare un ambiente educativo interattivo e coinvolgente.

Utilizzando Flutter, un framework di sviluppo versatile multipiattaforma, l'applicazione ottenuta, chiamata Learn AR, unisce l'esperienza dell'utente alla realtà aumentata. Questo progetto va oltre i confini associati alla fruizione tradizionale di materiale didattico per permettere allo studente che lo utilizza di visualizzare un oggetto complesso in modo che diventi reale.

Il caso di studio scelto per questa tesi è un libro che si occupa dell'architettura dei calcolatori. Attraverso questa applicazione gli studenti saranno in grado di visualizzare modelli 3D che arricchiscono i testi tradizionali e facilitano l'apprendimento pratico.
Nel caso di questa applicazione verranno utilizzati i modelli 3D di GPU, CPU e RAM. L'uso della realtà aumentata rende concetti complessi come la struttura di una GPU, più comprensibili. Questo caso di studio è stato scelto per dimostrare l'efficacia di un approccio educativo innovativo che combina contenuti teorici e pratici con un unico strumento di formazione.

L'applicazione Learn AR offrirà le seguenti funzionalità principali:

- **Marker AR**: pagine specifiche del libro conterranno marker che, inquadrati con l'applicazione, permetteranno di visualizzare modelli 3D relativi ai contenuti della pagina.

- **Visualizzazione di Modelli 3D**: gli utenti potranno esplorare rappresentazioni tridimensionali di componenti hardware.

- **Interazione con i Modelli 3D**: gli utenti avranno la possibilità di interagire con i modelli 3D, ad esempio ruotandoli, ingrandendoli o esplorandoli in dettaglio per comprendere meglio le diverse parti e il loro funzionamento.

- **Quiz**: l'applicazione conterrà quiz interattivi per verificare la comprensione degli utenti e rafforzare l'apprendimento.

La tesi esplorerà le fasi di analisi, progettazione e implementazione dell'applicazione, analizzando le sfide tecniche e le scelte progettuali necessarie per garantire un'applicazione ad alte prestazioni.
Attraverso questa applicazione, si è mirato a contribuire alla trasformazione digitale dell'istruzione superiore, proponendo applicazioni concrete e innovative in grado di ampliare gli orizzonti.
L'obiettivo del lavoro descritto nel presente documento di tesi è stato quello di sviluppare un'applicazione per dispositivi Android per consentire agli studenti di esplorare concetti complessi in modo tangibile.

Lo sviluppo dell'applicazione è stato realizzato passando per diverse fasi, descritte in modo accurato nei capitoli di questo documento di tesi.

- Capitolo 1: si discute la raccolta dei requisiti dell'app, distinguendo quelli funzionali da quelli non funzionali; i requisiti funzionali sono le caratteristiche che l'app deve avere, mentre quelli non funzionali rappresentano i vincoli ad essi collegati di cui tenere conto.

- Capitolo 2: è descritto il diagramma dei casi d'uso, in cui vengono riassunte le possibili interazioni tra l'utente e l'applicazione. Si è deciso di implementare i collegamenti logici, immaginando i possibili percorsi all'interno dell'app causati dall'interazione dell'utente.

- Capitolo 3: vengono elencati gli strumenti scelti per la scrittura effettiva del codice. In quanto l'applicazione in futuro dovrà essere cross-Platform, si è deciso di optare per il framework Flutter con il linguaggio di programmazione Dart, integrato nell'ambiente di sviluppo Android Studio. Poi si sono sfruttati i servizi della piattaforma Firebase per la gestione dei dati. Sono stati usati Unity e Vuforia Engine per la gestione della parte relativa alla realtà aumentata; è stato usato anche l'editor Blender per la modifica dei modelli 3D. Infine vengono elencati i vari plugin e package utilizzati per l'applicazione.

- Capitolo 4: riassume la fase di progettazione dell'applicazione. Questa fase è composta da diverse parti. Prima di tutto si è definita l'architettura, distinguendo le componenti modulari dell'app. In seguito, si è passati alla

progettazione dei dati, e infine si è proceduto con lo sviluppo dell'applicazione, parlando in modo più tecnico delle varie classi e metodi implementati. Nell'ultima sezione di questo capitolo sono mostrati gli screenshot dell'app

- Capitolo 5: propone delle considerazioni conclusive su lavoro svolto.

Capitolo 1

Analisi dei Requisiti

In accordo con la teoria dell'ingegneria del software, è essenziale la raccolta dei dati in requisiti funzionali e non funzionali.
Un requisito può essere definito come una specifica che dovrebbe essere implementata. I requisiti sono essenzialmente un'affermazione di ciò che il sistema dovrebbe fare e non di come lo deve fare.
I requisiti funzionali contengono le esigenze esplicite espresse dagli stakeholder e sono quelli che descrivono i servizi, o funzioni, offerti dall'applicazione.
Al contrario, i requisiti non funzionali appartengono ad aspetti più tecnici, come l'adozione di tecnologie specifiche, che sono più strettamente legati al dominio della soluzione piuttosto che a quello del problema.

1.1 Requisiti funzionali

I requisiti funzionali che l'app Learn AR dovrà garantire ai suoi utenti sono i seguenti:

1. il sistema dovrà gestire la registrazione di un nuovo utente all'interno del sistema;
2. il sistema dovrà gestire l'autenticazione degli utenti registrati;
3. il sistema dovrà gestire il riconoscimento di un'immagine target;
4. il sistema dovrà gestire la visualizzazione del modello 3D associato all'immagine target;
5. il sistema dovrà permettere all'utente di visualizzare le informazioni associate al modello 3D;
6. il sistema dovrà permettere all'utente di interagire con un modello 3D;
7. il sistema dovrà permettere di visualizzare le informazioni dell'utente;
8. il sistema dovrà permettere all'utente di eseguire un quiz relativo a uno specifico capitolo di un libro;

9. il sistema dovrà gestire il riconoscimento di un QR-CODE per l'avvio di uno specifico quiz.
10. il sistema dovrà gestire il salvataggio delle statistiche di uno quiz;
11. il sistema dovrà visualizzare le statistiche dei quiz eseguiti da un utente;
12. il sistema dovrà permettere all'utente di disconnettersi dall'applicazione.

1.2 Requisiti non funzionali

I requisiti non funzionali sono i seguenti:

1. l'applicazione dovrà essere realizzata attraverso la piattaforma Flutter;
2. il linguaggio di programmazione utilizzato per la realizzazione dell'applicazione dovrà essere Dart;
3. l'applicazione dovrà integrare all'interno Unity per l'interazione con i modelli 3D;
4. il sistema dovrà essere dotato di un'interfaccia grafica;
5. al sistema dovranno essere garantiti il permesso all'uso della fotocamera per permettere il corretto funzionamento dell'applicazione;
6. è necessario l'utilizzo di Firebase per memorizzare le informazioni degli utenti, delle domande e dei modelli 3D;

Capitolo 2

Casi d'uso

Questo capitolo tratta dell'analisi dei casi d'uso, una tecnica usata nei processi di ingegneria del software per effettuare in maniera esaustiva e non ambigua, la raccolta dei requisiti al fine di produrre software di qualità.

Una volta definiti i requisiti dell'applicazione da migliorare o da aggiungere, in particolare quelli funzionali, è stato possibile realizzare i diagrammi dei casi d'uso, ovvero lo schema in cui vengono rappresentate tutte le possibilità che l'applicazione fornisce.
Innanzitutto, prima di presentare il diagramma dei casi d'uso, definiamo cosa sono gli attori, ossia i ruoli assumibili dagli utenti o da qualsiasi entità che interagisce con l'applicazione, e i casi d'uso, ossia le possibili azioni che possono essere compiute dagli attori.

Il diagramma dei casi d'uso [2] è uno strumento che rappresenta la relazione tra un utente, definito come attore, e le sue richieste o aspettative rispetto al sistema. Questo diagramma è adatto a rappresentare le funzioni principali e/o gli obiettivi di un sistema software in maniera chiara. Il diagramma dei casi d'uso specifica come un sistema si comporterà, ed è per questo che mostra solo la funzionalità del sistema.

Nella Figura 2.1 è raffigurato il diagramma dei casi d'uso relativo all'applicazione. Come in ogni diagramma di questo tipo:

- Gli attori sono rappresentati tramite degli omini stilizzati.
- I casi d'uso sono rappresentati tramite delle ellissi.
- Le relazioni di inclusione ed estensione sono rappresentate tramite delle frecce direzionali.

2.1 Diagramma dei casi d'uso

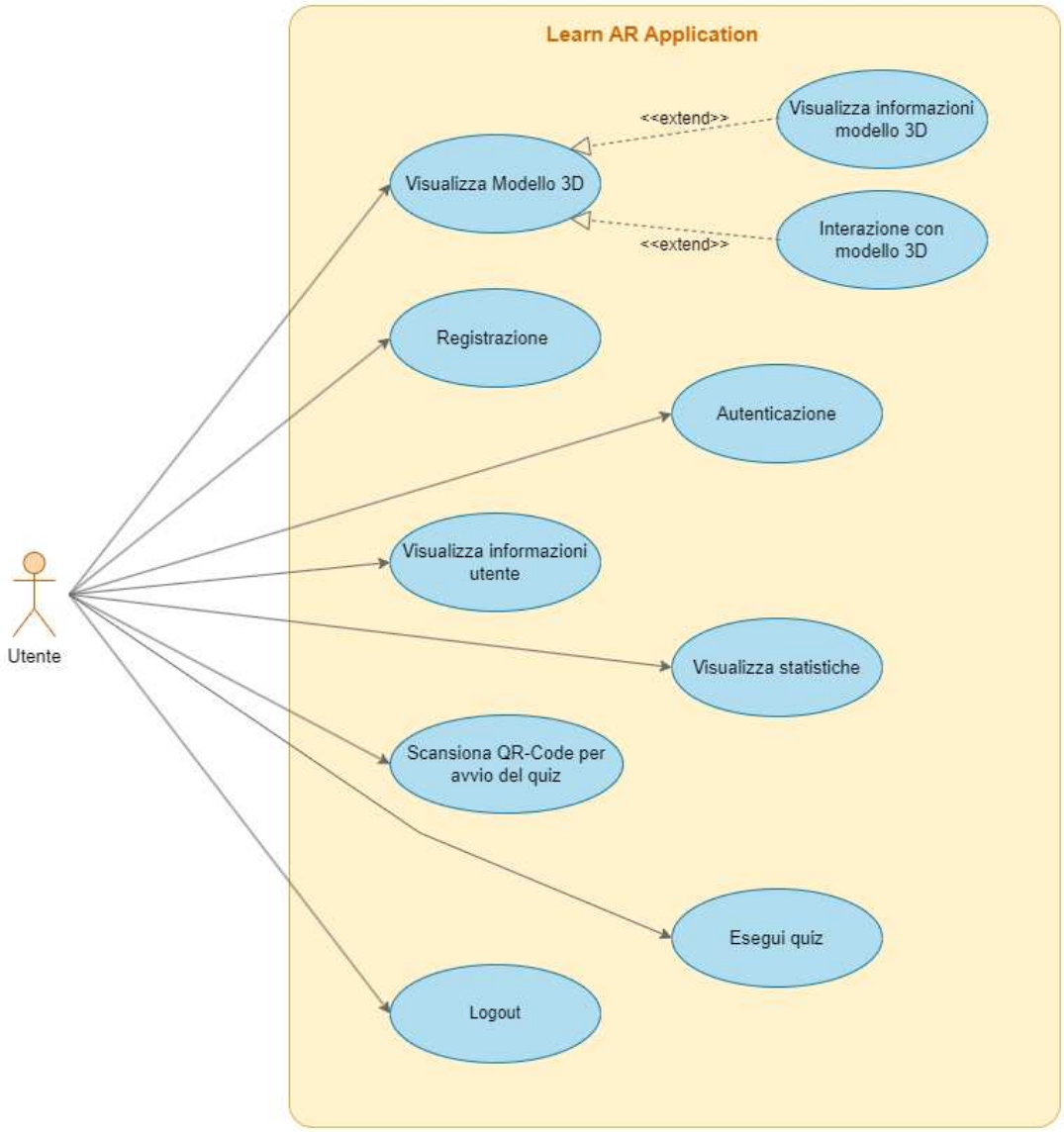


Figura 2.1: Diagramma dei casi d'uso

2.2 Descrizione tabellare dei casi d'uso

Dopo aver creato un diagramma dei casi d'uso è necessario definire le specifiche di ciascun caso d'uso.
Uno standard semplice ed efficace per la specifica dei casi d'uso può assicurare che il progetto raggiunga i suoi obiettivi per quanto riguarda l'analisi dei casi d'uso. Il modulo per specificare i casi d'uso contiene le seguenti informazioni: nome, attori coinvolti, precondizioni, postcondizioni, sequenza degli eventi principale, sequenza degli eventi alternativa.

Caso d'uso	**Visualizza Modello 3D**
Attori	Utente
Pre-condizioni	L'utente deve aver effettuato l'autenticazione.
Post-condizioni	Nessuna
Sequenza degli eventi principale	1. Il caso d'uso inizia quando l'utente inquadra un'immagine target. 2. Il sistema riconosce l'immagine collegata al modello 3D. 3. Il sistema visualizza a schermo il modello 3D sopra l'immagine target.
Sequenza degli eventi alternativa	Nessuna

Caso d'uso	**Visualizzazioni Informazioni Modello 3D**
Attori	Utente
Pre-condizioni	• L'utente deve aver effettuato l'autenticazione. • Il sistema deve aver visualizzato il modello 3D.
Post-condizioni	Nessuna
Sequenza degli eventi principale	1. Il caso d'uso inizia quando l'utente vuole visualizzare le informazioni del modello 3D. 2. L'utente accede alla schermata per la visualizzazione delle informazioni del modello 3D. 3. Il sistema visualizza a schermo le informazioni del modello 3D.
Sequenza degli eventi alternativa	Nessuna

Caso d'uso	**Interazione Modello 3D**
Attori	Utente
Pre-condizioni	• L'utente deve aver effettuato l'autenticazione. • Il sistema deve aver visualizzato il modello 3D.
Post-condizioni	Nessuna
Sequenza degli eventi principale	1. Il caso d'uso inizia quando l'utente vuole interagire con il modello 3D. 2. L'utente accede alla schermata per l'interazione con il modello 3D. 3. Il sistema visualizza a schermo il modello 3D. 4. L'utente interagisce con il modello 3D.
Sequenza degli eventi alternativa	Nessuna

Caso d'uso	Registrazione Utente
Attori	Utente
Pre-condizioni	L'utente non è registrato nel sistema.
Post-condizioni	L'utente è registrato nel sistema (o non è stato possibile aggiungerlo)
Sequenza degli eventi principale	1. Il caso d'uso inizia quando l'utente vuole effettuare la registrazione e creare un nuovo account. 2. Il sistema visualizza la schermata di login. 3. L'utente richiede di accedere alla schermata di registrazione. 4. Il sistema visualizza la schermata di registrazione. 5. L'utente inserisce tutte le informazioni richieste: e-mail, password, nome, cognome, data di nascita. 6. L'utente conferma le informazioni inserite. 7. Il sistema aggiunge con successo il nuovo utente a sistema.
Sequenza degli eventi alternativa	Inizia al punto 6 7. Le informazioni inserite dall'utente sono incomplete o l'utente già esiste a sistema (esiste un dipendente con stessa e-mail). 8. L'inserimento del nuovo utente nel sistema non va a buon fine.

Caso d'uso	Autenticazione
Attori	Utente
Pre-condizioni	L'utente è registrato nel sistema.
Post-condizioni	L'utente accede all'applicazione (o non è stato possibile permettere l'accesso)
Sequenza degli eventi principale	1. Il caso d'uso inizia quando l'utente vuole effettuare l'autenticazione per utilizzare l'applicazione. 2. Il sistema visualizza la schermata di login. 3. L'utente inserisce le proprie credenziali di accesso (e-mail e password) 4. L'utente avvia la procedura di autenticazione. 5. Il sistema visualizza a schermo la home page dell'utente.
Sequenza degli eventi alternativa	Inizia al punto 4 5. Le informazioni inserite dall'utente non sono valide. 6. La procedura di autenticazione non va a buon fine.

Caso d'uso	Visualizzazioni Informazioni Utente
Attori	Utente
Pre-condizioni	L'utente deve aver effettuato l'autenticazione.
Post-condizioni	Nessuna
Sequenza degli eventi principale	1. Il caso d'uso inizia quando l'utente vuole visualizzare le informazioni del profilo. 2. L'utente accede alla schermata del profilo. 3. Il sistema visualizza a schermo le informazioni dell'utente.
Sequenza degli eventi alternativa	Nessuna

Caso d'uso	Visualizzazioni Statistiche
Attori	Utente
Pre-condizioni	L'utente deve aver effettuato l'autenticazione.
Post-condizioni	Nessuna
Sequenza degli eventi principale	1. Il caso d'uso inizia quando l'utente vuole visualizzare le statistiche relative ai quiz svolti. 2. L'utente accede alla schermata delle statistiche. 3. Il sistema visualizza a schermo le statistiche.
Sequenza degli eventi alternativa	Nessuna

Caso d'uso	Scansione QR-Code per Avvio del Quiz
Attori	Utente
Pre-condizioni	L'utente deve aver effettuato l'autenticazione.
Post-condizioni	Nessuna
Sequenza degli eventi principale	1. Il caso d'uso inizia quando l'utente vuole scansionare il QR-Code, posto alla fine di ogni capitolo, per l'avvio del quiz. 2. L'utente inquadra il QR-Code. 3. Il sistema avvia il quiz.
Sequenza degli eventi alternativa	Inizia al punto 2 3. Il capitolo collegato al QR-Code è bloccato. 4. Il sistema notifica all'utente che il quiz per il capitolo è bloccato.

Caso d'uso	Esegui Quiz
Attori	Utente
Pre-condizioni	L'utente deve aver effettuato l'autenticazione.
Post-condizioni	La statistica viene inserita a sistema.
Sequenza degli eventi principale	1. Il caso d'uso inizia quando l'utente vuole l'avviare un quiz. 2. L'utente accede alla schermata delle domande. 3. L'utente svolge il quiz. 4. L'utente conferma il quiz. 5. Il sistema inserisce il punteggio ottenuto dall'utente a sistema.
Sequenza degli eventi alternativa	Inizia al punto 4 5. Il punteggio delle domande corrette è minore del 60%. 6. Il sistema consente solo di risvolgere il quiz o tornare alla schermata principale dei quiz.
Sequenza degli eventi alternativa	Inizia al punto 3 4. L'utente decide di riavviare il quiz. 5. L'utente svolge il quiz. 6. L'utente conferma il quiz. 7. Il sistema inserisce il punteggio ottenuto dall'utente a sistema.
Sequenza degli eventi alternativa	Inizia al punto 3 4. L'utente decide di tornare alla schermata principale dei quiz. 5. Il sistema visualizza la schermata principale dei quiz.

Caso d'uso	Logout
Attori	Utente
Pre-condizioni	L'utente ha effettuato il login in precedenza.
Post-condizioni	L'utente viene disconnesso dall'applicazione.
Sequenza degli eventi principale	1. Il caso d'uso inizia quando l'utente vuole effettuare il logout. 2. Il sistema accede alla schermata home. 3. L'utente avvia la procedura di logout. 4. L'utente avvia la procedura di autenticazione. 5. Il sistema disconnette l'utente dall'applicazione.
Sequenza degli eventi alternativa	Nessuna

Capitolo 3

Strumenti Utilizzati

In questo capitolo verranno analizzati gli strumenti e le tecnologie che sono state utilizzate per la realizzazione dell'applicazione Learn AR. Nello specifico:

- **Android Studio** come IDE di programmazione.
- **Flutter** come framework di lavoro.
- **Dart** come linguaggio di programmazione.
- **Firebase** per la memorizzazione e la gestione dei dati.
- **Unity** per l'integrazione della realtà aumentata.
- **Vuforia Engine** per lo sviluppo AR.
- **Blender** come editor per i modelli 3D.

3.1 Ambiente di sviluppo

Android Studio [3] è l'ambiente di sviluppo integrato (IDE) ufficiale per lo sviluppo di app Android. Basato sui potenti editor di codice e sugli strumenti per sviluppatori di IntelliJ IDEA, Android Studio offre ancora più funzionalità che migliorano la tua produttività durante la creazione di app Android, ad esempio:

- Un sistema di build flessibile basato su Gradle
- Un emulatore veloce e ricco di funzionalità
- Un ambiente unificato in cui è possibile sviluppare applicazioni per tutti i dispositivi Android
- Live Edit per aggiornare in tempo reale i componibili in emulatori e dispositivi fisici
- Modelli di codice e integrazione di GitHub per creare funzionalità comuni dell'app e importare codice

- Ampi strumenti e framework di test
- Strumenti lint per rilevare prestazioni, usabilità, compatibilità della versione e altri problemi
- Supporto C++ e NDK
- Supporto integrato per Google Cloud Platform, facilitando l'integrazione di Google Cloud Messaging e App Engine

La finestra principale di Android Studio contiene le diverse aree logiche mostrate in Figura 3.1:

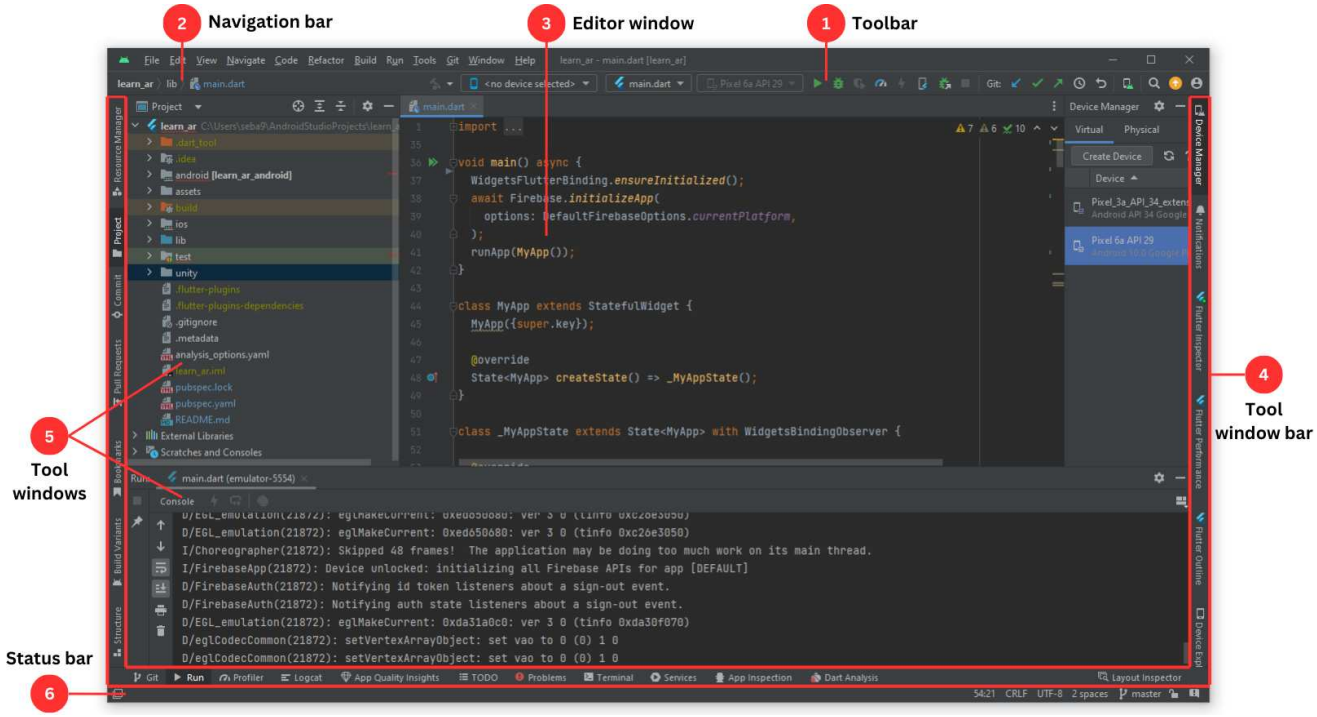


Figura 3.1: Interfaccia utente Android Studio

1. La **Toolbar** ci offre un'ampia gamma di azioni, tra cui l'esecuzione di app e l'avvio di strumenti Android.
2. La **Navigation bar** aiuta a navigare nel nostro progetto e ad aprire i file per la modifica. Fornisce una vista compatta della struttura visibile nella Finestra Progetto.
3. La **Editor window** è uno spazio in cui possiamo creare e modificare il nostro codice. In base al tipo di file corrente l'editor può cambiare. Durante la visualizzazione di un file di layout, l'editor visualizza l'editor di layout.
4. La **Tool window bar** è all'esterno della finestra IDE e contiene pulsanti che consentono di espandere e comprimere le singole finestre degli strumenti.

5. Le **Tool windows** ci forniscono l'accesso ad attività specifiche come la ricerca, la gestione dei progetti, il controllo della versione e altro ancora. Possiamo espanderli e comprimerli.

6. La **Status bar** mostra lo stato del nostro progetto e dell'IDE stesso, nonché eventuali messaggi o avvisi.

3.2 Flutter

Flutter [4] [5] è un progetto open source completamente gratuito per la creazione rapida di app native di alta qualità su iOS e Android e per il supporto di interfacce native. Il progetto è sviluppato e mantenuto da Google e dalla community.

Flutter è composto da due parti importanti:

- Un SDK (Software Development Kit): una raccolta di strumenti che aiutano a sviluppare le applicazioni. Ciò include strumenti per compilare il codice in codice macchina nativo (codice per iOS e Android).

- Un framework (UI Library basata su widget): una raccolta di elementi dell'interfaccia utente riutilizzabili (pulsanti, input di testo, dispositivi di scorrimento e così via) che si possono personalizzare in base alle proprie esigenze.

L'obiettivo di Flutter è quello di creare nuove app tramite:

- una fase di sviluppo rapida con funzionalità quali l'**hot reload**, che non richiede una compilazione completa del codice, ma solo di una parte;

- interfacce utente espressive e flessibili con un set di widget componibili, librerie per animazioni e un'architettura stratificata ed estensibile;

- performance molto vicine a quelle native;

- una codebase unica per le applicazioni Android e iOS.

Per rendere ciò possibile, Flutter si compone di due macro strati:

- uno strato scritto in C/C++;

- uno strato scritto in Dart, un moderno linguaggio orientato agli oggetti che definisce la maggior parte del suo sistema (gesture, animazioni, framework, widget, ecc.) e offre agli sviluppatori un grande controllo sul sistema stesso.

3.3 Dart

Dart [6] [7] è un linguaggio di programmazione open source, generico e orientato agli oggetti con sintassi in stile C sviluppato da Google nel 2011. È facile da imparare e comprendere perché utilizza una sintassi e una struttura semplici. Dart è orientato agli oggetti, il che significa che si concentra sulla creazione di pezzi di codice riutilizzabili chiamati oggetti. Viene utilizzato per creare applicazioni Web, mobili e desktop. Dart può essere utilizzato per creare programmi autonomi o per sviluppare app utilizzando framework come Flutter. È ispirato ad altri linguaggi di programmazione come Java, JavaScript, C# ed è fortemente tipizzato. Poiché Dart è un linguaggio compilato, non è possibile eseguire direttamente il codice; invece, il compilatore lo analizza e lo trasferisce nel codice macchina. Supporta la maggior parte dei concetti comuni dei linguaggi di programmazione come classi, interfacce, funzioni, a differenza di altri linguaggi di programmazione.

3.4 Firebase

Firebase [8] [9] è una piattaforma serverless per lo sviluppo di applicazioni mobili e web.
Open source ma supportata da Google, Firebase sfrutta l'infrastruttura di Google e il suo cloud per fornire una suite di strumenti per scrivere, analizzare e mantenere applicazioni cross-platform. Firebase infatti offre funzionalità come analisi, database (usando strutture NoSQL), messaggistica e segnalazione di arresti anomali per la gestione di applicazioni web, iOS e Android.
Per lo sviluppo della app oggetto di questo elaborato, sono stati impiegati tre dei suoi componenti: Realtime Database per il database, Firebase Storage per l'archiviazione dei file e Firebase Authentication per la gestione dell'autenticazione degli utenti.

Firebase Authentication mira a semplificare la creazione di sistemi di autenticazione sicuri, migliorando al tempo stesso l'esperienza di accesso per gli utenti finali. Fornisce una soluzione di identità end-to-end, supportando account di posta elettronica e password, autenticazione telefonica, accesso a Google, Twitter, Facebook e GitHub. È stato integrato nell'applicazione con la funzione di autentificare e registrare i vari utenti.

Firebase Realtime Database è un database NoSQL ospitato sul cloud che consente di archiviare e sincronizzare i dati tra i gli utenti in tempo reale. Ciò significa che ogni volta che i dati vengono aggiornati sul server, i client connessi ricevono gli aggiornamenti in millisecondi. Questa funzionalità facilita la creazione di app altamente reattive che mostrano sempre informazioni aggiornate. Esistono diversi motivi convincenti per considerare l'utilizzo del database Realtime:

- sincronizzazione istantanea: per le app che richiedono la condivisione istantanea dei dati tra utenti;
- funzionalità offline: gli SDK Firebase gestiscono connessioni di rete intermittenti. Memorizza nella cache i dati che possono essere sincronizzati quando la rete è nuovamente disponibile;
- scalabilità: non è necessario gestire server o scrivere codice lato server. Firebase si ridimensiona automaticamente per supportare gli utenti dell'app;
- sicurezza: Firebase offre solide funzionalità di autenticazione utente e convalida dei dati;
- compatibilità con Flutter: gli SDK Firebase sono completamente compatibili con Flutter, sfruttando le funzionalità di Flutter per offrire un codice pulito e conciso.

Firebase Storage fornisce un cloud storage scalabile e sicuro per contenuti come immagini, video e file. È possibile organizzare i file in cartelle, semplificando la gestione dei dati multimediali. Questo cloud è stato utilizzato per lo storage dei modelli 3D (formato .gbl).

3.5 Unity

Unity [10] [11] è un motore di gioco 2D e 3D nato nel 2005.
Sviluppato da Unity Technologies, è stato realizzato per fornire a più sviluppatori l'accesso agli strumenti di sviluppo di giochi, che a quei tempi era una nuova impresa. Nel corso della sua lunga vita, il motore è cambiato e si è espanso notevolmente, riuscendo a stare al passo con le pratiche e le tecnologie più recenti.

Quando si tratta di Realtà Virtuale (VR) e Realtà Aumentata, che sono le tecnologie più recenti, Unity è uno dei principali sostenitori dello sviluppo con esse.
Per la Realtà Virtuale, sono disponibili numerosi pacchetti che supportano quasi tutti i visori VR disponibili e vengono costantemente aggiornati e mantenuti flessibili con questa tecnologia in evoluzione.

Per la Realtà Aumentata sono disponibili numerosi pacchetti per ARCore e ARKit. Unity offre anche AR Foundation, creata da Unity per consentire agli sviluppatori Unity di creare app AR per Android e iOS contemporaneamente, eliminando la necessità di progetti separati.

Inoltre, Unity ora dispone anche dell'XR Interaction Toolkit per rendere ancora più semplice lo sviluppo di giochi VR e AR.
Sebbene esistano alcuni motori che supportano VR e AR, come Unreal Engine, pochi lo fanno altrettanto bene di Unity. Unity supporta attivamente queste tecnologie in ogni modo possibile ed è al loro fianco sin dall'inizio della loro ascesa. Sebbene ci siano sicuramente miglioramenti che potrebbero essere apportati

al modo in cui Unity implementa questi aspetti, l'enorme quantità di supporto e integrazione con il motore lo rendono una valida scelta se si è interessati a sviluppare progetti per queste tecnologie.

3.6 Vuforia Engine

Vuforia Engine [12] è un kit di sviluppo software (SDK) per la creazione di app di realtà aumentata, con supporto per la maggior parte di telefoni, tablet e occhiali. Permette facilmente di aggiungere funzionalità avanzate di visione artificiale alle app Android, iOS e permette di creare esperienze AR che interagiscono realisticamente con gli oggetti e l'ambiente.
La libreria Vuforia Engine è organizzata in 2 principali categorie: Immagini e oggetti. Vuforia fornisce strumenti per creare target, gestire database di target e testarli. I target immagine rappresentano le immagini che Vuforia Engine può rilevare e tracciare.
Il motore rileva e tiene traccia dell'immagine confrontando le caratteristiche naturali estratte dall'immagine della fotocamera con un database di risorse preesistente. Vuforia utilizza la tecnologia di visione artificiale per riconoscere e tracciare immagini planari e oggetti 3D in tempo reale. Questa funzionalità di registrazione delle immagini consente di posizionare e orientare oggetti virtuali, come modelli 3D, rispetto agli oggetti del mondo reale quando vengono visualizzati attraverso la fotocamera di un dispositivo mobile. L'oggetto virtuale traccia quindi la posizione e l'orientamento dell'immagine in tempo reale in modo che la prospettiva dell'osservatore sull'oggetto corrisponda alla prospettiva sul bersaglio. Sembra quindi che l'oggetto virtuale faccia parte della scena del mondo reale.

L'SDK Vuforia supporta una varietà di tipi di target 2D e 3D, inclusi target immagine "senza marker", target modello 3D e una forma di marcatore indirizzabile, noto come VuMark.
Ulteriori funzionalità dell'SDK includono la localizzazione del dispositivo a 6 gradi di libertà nello spazio, il rilevamento dell'occlusione localizzato tramite "pulsanti virtuali", la selezione del target dell'immagine di runtime e la possibilità di creare e riconfigurare i set di target a livello di codice in fase di runtime.

Vuforia fornisce API (Application Programming Interface) attraverso un'estensione al Game Engine Unity. In questo modo, l'SDK supporta lo sviluppo nativo per iOS, Android e consente anche lo sviluppo di applicazioni AR in Unity facilmente trasferibili su entrambe le piattaforme. L'integrazione di Vuforia da parte di Unity ti consente di creare app e giochi visivi per Android e iOS utilizzando un flusso di lavoro di creazione drag-and-drop.

Oltre agli smartphone e ai tablet, Vuforia supporta molti dispositivi di terze parti (come occhiali AR/MR) e dispositivi VR con fotocamere posteriori (come Gear VR).

Marker-based tracking
Nell'AR, i marcatori sono immagini o oggetti registrati con l'applicazione che fungono da trigger di informazioni nell'applicazione. Quando la fotocamera del dispositivo riconosce questi marcatori nel mondo reale (durante l'esecuzione di un'applicazione AR), ciò attiva la visualizzazione del contenuto virtuale sulla posizione del marcatore nella vista della fotocamera. Il tracciamento basato su marcatori di Vuforia può utilizzare una varietà di tipi di marcatori diversi, inclusi codici QR, marcatori fisici riflettenti, target immagine e tag 2D.
Nell' applicazione è stato usato come tipo di marker il target image.

Image Targets
Gli Image Targets sono un tipo specifico di marcatore utilizzato nel tracciamento basato su marcatore. Sono immagini che sono state registrate manualmente in uno specifico database e fungono da trigger per visualizzare il contenuto virtuale. Per la scelta delle immagini si predilige utilizzare immagini contenenti forme distinte con contorni complessi. Ciò rende più semplice il riconoscimento delle immagini e gli algoritmi di tracciamento.

3.7 Blender

Blender [13] [14] è una suite di creazione 3D gratuita e open source. Supporta l'intera pipeline 3D: modellazione, rigging, animazione, simulazione, rendering, compositing e tracciamento del movimento, persino editing video e creazione di giochi. Gli utenti avanzati utilizzano l'API di Blender per lo scripting Python per personalizzare l'applicazione e scrivere strumenti specializzati; spesso questi sono inclusi nelle versioni future di Blender. Blender è adatto a privati e piccoli studi che traggono vantaggio dalla sua pipeline unificata e dal processo di sviluppo reattivo. Blender è multipiattaforma e la sua interfaccia utilizza OpenGL per fornire un'esperienza coerente.

Essendo un progetto guidato dalla comunità sotto la GNU General Public License (GPL), la comunità degli sviluppatori ha il potere di apportare piccole e grandi modifiche al codice base, il che porta a nuove funzionalità, correzioni di bug reattive e migliore usabilità.

Nello sviluppo dell'applicazione è stato usato per modificare i modelli 3D, a cui sono state aggiunte frecce, scritte e colorate singole parti di un modello.

3.8 Package e Plugin

I package consentono la creazione di codice modulare che può essere condiviso facilmente. Un package è costituito da almeno:

- **pubspec.yaml**: un file di metadati che dichiara il nome del pacchetto, la versione, l'autore e così via.

- **lib**: la libdirectory contiene il codice pubblico nel pacchetto, almeno un singolo <package-name>.dartfile.

I package possono contenere più di un tipo di contenuto:

- **Dart packages (package)**
  Pacchetti generali scritti in Dart, ad esempio il path package. Alcuni di questi possono contenere funzionalità specifiche di Flutter e quindi avere una dipendenza dal framework Flutter, limitandone l'uso solo a Flutter.

- **Plugin packages (plugin)**
  Un package Dart specializzato che contiene un'API scritta in codice Dart combinata con una o più implementazioni specifiche della piattaforma. I pacchetti di plug-in possono essere scritti per Android (utilizzando Kotlin o Java), iOS (utilizzando Swift o Objective-C), Web, macOS, Windows o Linux o qualsiasi combinazione di questi.

Figura 3.2: Dipendenze dichiarate nel file pubspec.yaml

Scendendo più nel particolare, nel caso dell'app sono stati utilizzati, come mostrato in Figura 3.2, i seguenti package e plugin:

- firebase_core: un plugin per usare le API Firebase Core, che attiva la connessione a molteplici servizi di Firebase.

- firebase_auth: un plugin Flutter per usare le API Firebase Auth, che consente l'autenticazione tramite password, numeri di telefono e provider di identità come Google, Facebook e Twitter.

- firebase_storage: un plugin che consente di utilizzare lo Storage di Firebase in un progetto Flutter.

- firebase_database: un plugin Flutter che rende molto più di facile implementazione tutta la parte riguardante il login, il logout e la registrazione di un nuovo account.

- flutter_unity_widget: un plugin flutter che consente di incorporare Unity all'interno di una flutter app.

- provider: Si tratta di un pacchetto di gestione dello stato per Flutter che consente di gestire e condividere i dati tra i widget in modo efficiente.

- path_provider: un plugin Flutter per trovare le posizioni comunemente utilizzate sul filesystem.

- cupertino_icons: un repository di risorse contenente il set predefinito di risorse di icone utilizzate dai widget di Cupertino di Flutter

- intl: fornisce funzionalità di internazionalizzazione e localizzazione, tra cui traduzione di messaggi, formattazione e analisi di data/numero e testo bidirezionale.

- flutter-toast : libreria che permette di creare facilmente messaggi Toast in un'unica riga di codice.

- shared_preferences: pacchetto Flutter che consente di archiviare e recuperare facilmente tipi di dati semplici, ad esempio numeri interi, stringhe, valori booleani e float, in modo persistente sul dispositivo.

- permission_handler: un plugin che fornisce un'API multipiattaforma (iOS, Android) per richiedere e verificare le autorizzazioni.

- google_fonts: un packege Flutter per utilizzare i caratteri da fonts.google.com.

- http: un package Flutter che contiene un insieme di funzioni e classi di alto livello che semplificano l'utilizzo delle risorse HTTP.

- model_viewer_plus: un package per il rendering di modelli 3D interattivi nei formati glTF e GLB.

- gcloud: un package che fornisce un'interfaccia Dart idiomatica di alto livello per alcuni dei servizi Google Cloud Platform più utilizzati (Cloud Datastore, Cloud Storage, Cloud Pub/Sub).

- flutter_svg: una libreria di rendering e widget SVG per Flutter, che consente di disegnare e visualizzare file Scalable Vector.
- qr_code_scanner : un plugin di scanner di codici QR che può essere incorporato all'interno di Flutter.
- animated_splash_screen: permette di creare una schermata iniziale animata in modo completamente personalizzabile.
- basic_utils: un package per molti metodi di supporto adatti a varie situazioni.

3.8.1 Libreria flutter_unity_widget

Il plugin flutter_unity_widget [15] è stato già citato nel paragrafo precedente, ma poiché lo scopo centrale dell'applicazione mirava allo sviluppo di una app dove l'utente potesse visualizzare sopra un immagine target un modello 3D e interagire con i modelli 3D, tutto questo è stato possibile con l'ausilio di tale libreria. Vediamola ora in maniera più dettagliata.

Il Flutter Unity Widget gestisce la maggior parte delle attività nell'ambito dell'unione "Flutter e Unity". È uno strumento che facilita l'incorporamento di Unity in un'app Flutter. È più di un semplice widget standard; è un ponte che collega il "progetto Flutter" con un "progetto Unity", portando a una reale interazione tra i due ecosistemi.

La caratteristica principale di questo widget è il potere di controllare la vista Unity dall'app Flutter. Le sue solide basi del "Unity Engine" e dello "scripting backend" garantiscono una connessione fluida con il Unity, abilitando funzionalità specifiche, come mettere in pausa il lettore Unity o riprendere il lettore Unity, con precisione all'interno dell'app Flutter.

Flutter Unity Widget supporta sia Android che iOS e consente di abilitare comodamente comportamenti in background personalizzati su entrambe le piattaforme.

Capitolo 4

Sviluppo software

In questo capitolo verranno descritte le fasi di progettazione dell'app Learn AR. La progettazione del software è una delle fasi cruciali nello sviluppo del software.

Una volta stabilite le funzionalità da garantire e gli strumenti che verranno utilizzati, si deve andare a definire l'architettura dell'app, cioè le componenti software principali e le modalità con cui esse comunicheranno fra loro.
Successivamente verrà definita la struttura dei dati che l'app utilizzerà. Verrà cioè analizzato il modo in cui i dati sono salvati e quali sono i loro attributi.
La fase centrale sarà quella di definizione del funzionamento delle classi. In questa parte sarà descritta la logica dell'applicazione, quindi i funzionamenti dei vari metodi e di come essi possano utilizzare i dati e gestire le interazioni dell'utente per modificare dinamicamente l'interfaccia.
Infine, si dovrà andare a scrivere il codice, e l'ultima fase sarà quindi quella dell'implementazione. Questa fase consiste nell'utilizzare tutti gli strumenti a disposizione e, tramite il codice Dart, implementare dei metodi che svolgano il lavoro necessario al fine del funzionamento dell'app.

4.1 Architettura

In questa sezione l'obiettivo sarà quello di definire, in modo sintetico e chiaro, l'architettura alla base dell'applicazione Learn AR.
Considerando che tutta l'architettura viene ricondotta dal framework stesso ai Widget, si è scelto di associare a ogni funzionalità dell'applicazione un Widget specifico. Non si può quindi parlare di un vero e proprio modello architetturale, ma piuttosto di un insieme di Widget distinti interoperanti tra loro, essendo Flutter costituito da una serie di strati (layer), ognuno dei quali comunica con quello sottostante e ne sfrutta i servizi messi a disposizione.

Per quanto riguarda l'app si è scelto di usare la Provider Architecture [16]. Il modello di architettura Provider è un modello leggero e facile da usare che consente di gestire lo stato dell'app in modo scalabile e testabile. Il provider utilizza le classi InheritedWidget e ChangeNotifier per gestire lo stato e supporta una varietà di casi d'uso come l'inserimento delle dipendenze, la gestione dello stato globale e la composizione dei widget.

Provider è un'architettura Flutter che fornisce il modello di dati corrente nel luogo in cui ne abbiamo attualmente bisogno. Contiene alcuni dati e avvisa gli osservatori quando si verifica un cambiamento.
Nell'SDK Flutter, questo tipo è chiamato ChangeNotifier. Affinché l'oggetto di tipo ChangeNotifier sia disponibile per altri widget, abbiamo bisogno di ChangeNotifierProvider. Esso fornisce oggetti osservati per tutti i suoi discendenti. L'oggetto che è in grado di ricevere i dati correnti è il Consumer, che ha un'istanza ChangeNotifier nel parametro della sua funzione di build che può essere utilizzata per alimentare le visualizzazioni successive con i dati.

In questa architettura (Figura 4.1), un oggetto chiamato "widget provider" fornisce le dipendenze (ovvero i valori necessari) a un altro oggetto, il "widget dipendente". In altre parole, il widget provider mette a disposizione le risorse di cui il widget dipendente ha bisogno per funzionare correttamente. Questa è la dependency injection. Possiamo dire che i widget del provider inseriscono valori nei widget dipendenti.
Come abbiamo visto, il valore iniettato a volte è l'origine dello stato, come quando è un Listenable o un ChangeNotifier. In questo caso, il valore e l'origine dello stato si comportano come un modello. Il modello potrebbe essere, ad esempio, un User. Quando un provider inserisce un modello nei suoi dipendenti, implementa l'inserimento delle dipendenze convenzionale.
Tuttavia, il valore spesso non è l'origine dello stato, ad esempio quando l'origine dello stato è Stream, Future o ValueListenable. In questo caso, il provider designa l'origine dello stato ma fornisce ai suoi "widget dipendenti" i valori dello stato emessi. In questo caso, il provider inserisce effettivamente nei suoi "widget dipendenti" solo un'origine di stato, mentre in realtà inietta invece i valori di stato man mano che cambiano.

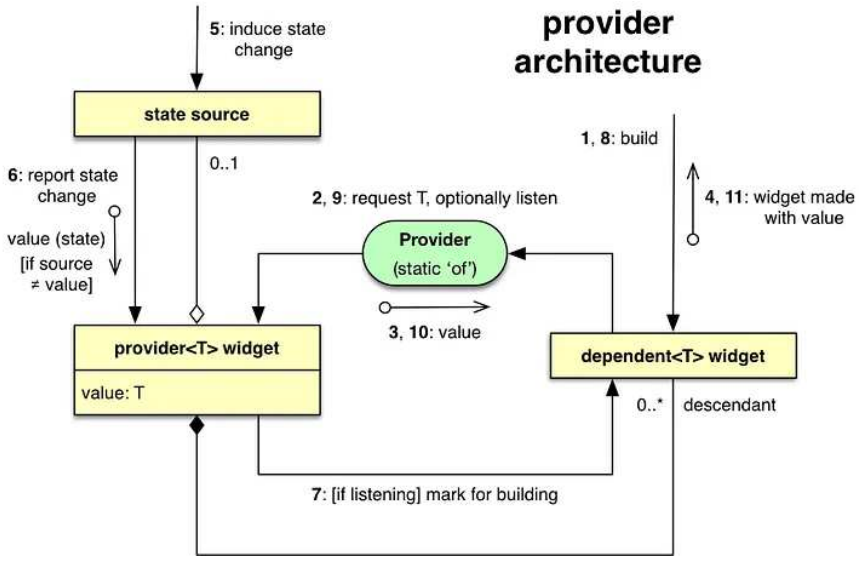


Figura 4.1: Provider Architecture

L'architettura del provider implementa quindi sia l'inserimento delle dipendenze che il data binding, utilizzando il data binding per la gestione dello stato.

Per quanto riguarda l'architettura dell'applicazione, si devono distinguere due componenti principali (Figura 4.2). La prima è la parte che racchiude logica e grafica dell'applicazione, costituita dalle classi definite nei file Dart del progetto Flutter.

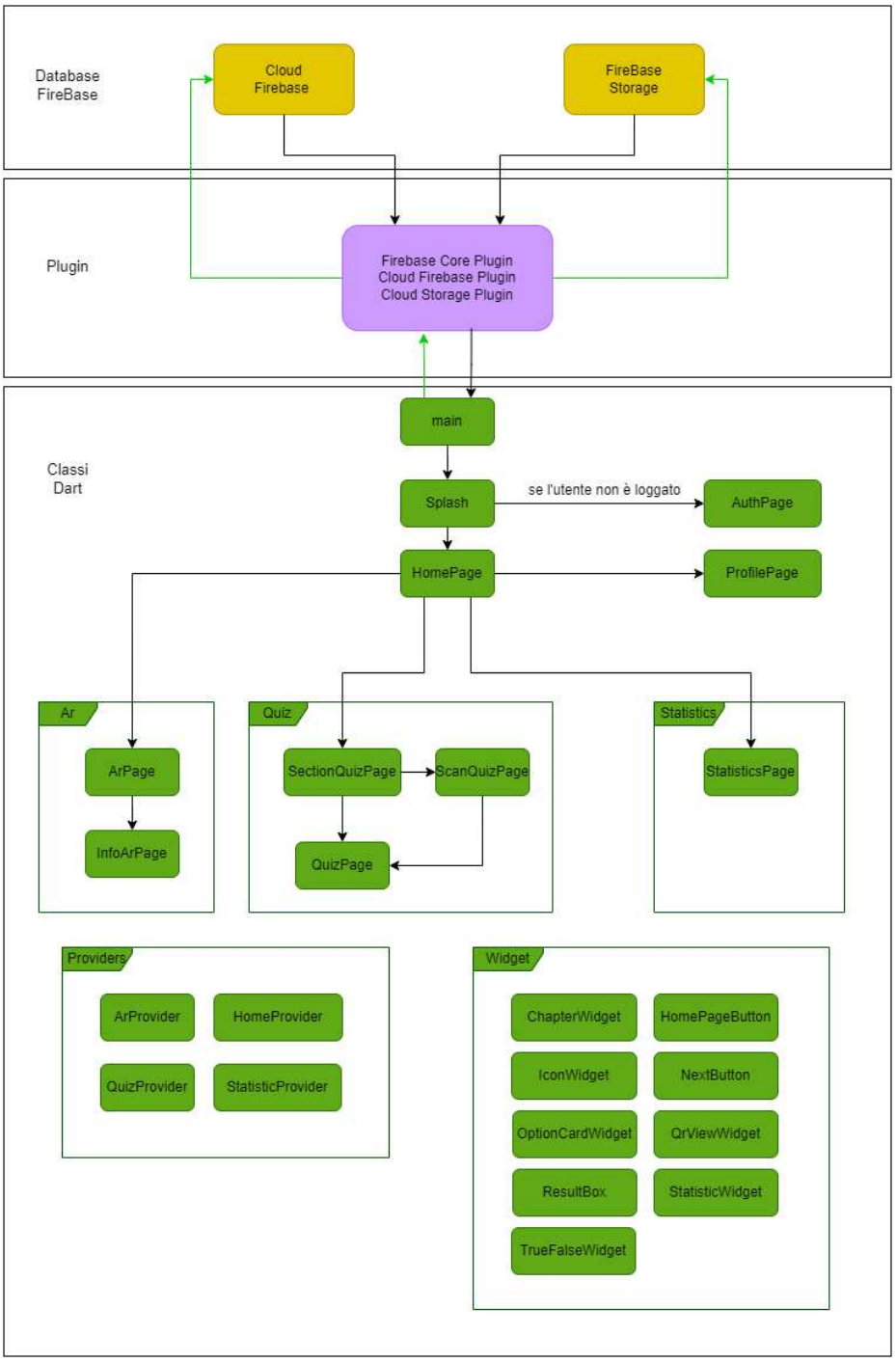


Figura 4.2: Architettura applicazione

La seconda componente fondamentale dell'applicazione è la sorgente persistente di dati, formata dai database messi a disposizione tramite i servizi Firebase. Le classi Dart interagiscono con i database (Realtime Database e Firebase Storage) utilizzando dei package disponibili in Flutter, e una volta prelevati i dati li possono utilizzare per popolare le varie schermate.

Più nello specifico, queste sono le classi facenti parte del progetto Flutter:

- Splash: classe che si occupa di gestire il login automatico dell'utente nel caso in cui è già registrato a sistema.

- AuthPage: classe che gestisce l'operazione di autenticazione dell'utente e/o la registrazione dell'utente, tramite Firebase Authentication che è un servizio fornito dalla piattaforma Firebase di Google. Offre un modo sicuro e semplice per gestire l'autenticazione degli utenti, incluse funzionalità come l'autenticazione e-mail/password, l'integrazione dell'accesso ai social media e altro ancora.

- HomePage: definisce la schermata principale dell'applicazione. Classe che gestisce l'operazione di navigazione fra tutte le view principali dell'applicazione.

- ProfilePage: classe che gestisce la visualizzazione delle informazioni dell'utente; vengono visualizzati: nome, cognome, email, data di nascita.

- ArPage: classe che gestisce del riconoscimento di immagini target e la visualizzazione in realtà aumentata di modelli 3D attraverso l'integrazione con Unity. La classe gestisce anche l'interazione con il modello 3D associato a l'immagine target.

- InfoArPage: classe che gestisce la visualizzazione delle informazioni di un immagine target/modello 3D. La classe si occupa di visualizzare i dati recuperati dal database Realtime Database.

- SectionQuizPage: definisce la schermata della sezione relativa ai quiz. Classe che gestisce l'operazione di navigazione fra tutte le view della sezione quiz.

- ScanQuizPage: classe che gestisce il riconoscimento di un qr-code per l'inizio di un quiz associato.

- QuizPage: classe che gestisce la visualizzazione e la logica di un quiz relativo a un capitolo. La classe gestisce sia la visualizzazione all'interno del quiz di modelli 3D recuperati dal Firebase Storage sia le domande recuperate dal Realtime Database.

- StatiscicPage: classe che gestisce la visualizzazione delle statistiche relative ai quiz divise per capitolo.

- ArProvider, HomeProvider, QuizProvider e StatisticProvider: classi simili, ma relativi a sezione diverse dell'applicazione, che servono a gestire lo stato dell'applicazione.

- Widget: sezione contenente una serie di widget personalizzati che vengono utilizzati all'interno dell'applicazione.

- Model: sezione contente le classi UserModel, Chapter, Question, Statistic, Info, per l'accesso ai dati.

L'applicazione Learn AR utilizza poi due fonti persistenti di dati:

- Realtime Database: è un database remoto messo a disposizione da Firebase, in cui è possibile salvare dati in documenti raccolti in delle collection (collezioni).

- Firebase Storage: è un servizio di archiviazione remoto di oggetti, particolarmente utilizzato per foto e video. Il database che l'app utilizza è utilizzato appunto per salvare i tutti modelli 3D.

La classe 'main' non può comunicare direttamente con i servizi Firebase, ma ha bisogno di alcuni plugin esterni aggiuntivi. Per questo vengono utilizzati dei package, ovvero delle componenti software aggiuntive che permettono la comunicazione tra il codice Dart e i database. In particolare, vengono utilizzati i seguenti package:

- firebase_core: serve per poter sfruttare le API di Firebase.

- firevasa_database: per poter comunicare con il database Realtime Database.

- firebase_storage: per comunicare con lo Storage.

- firebase_auth: per il login e la registrazione.

4.2 Progettazione dei dati

Questa sezione è dedicata a descrivere il modello utilizzato per la rappresentazione dei dati relativi ai quiz. Come detto in precedenza, l'applicazione utilizza due database che fanno parte dei servizi Firebase, ovvero Realtime Database e Firebase Storage. Per poter utilizzare i servizi Firebase, si deve prima creare un progetto e poi collegarlo al proprio progetto Flutter. Questo avviene inserendo nella directory android/app il file 'google-services.json', fornito automaticamente da Firebase.

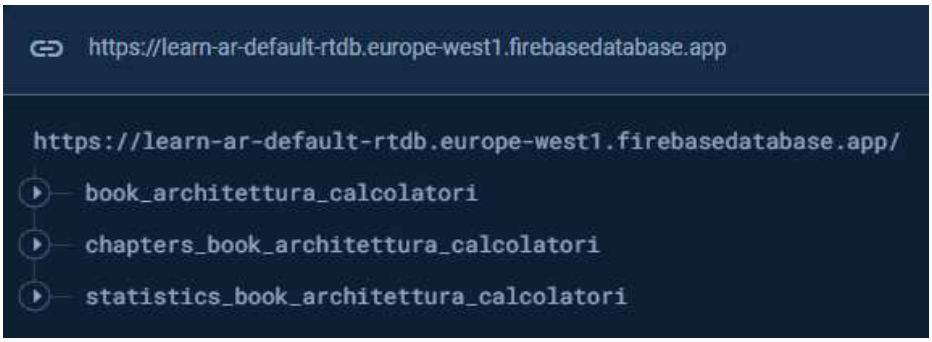


Figura 4.3: Architettura applicazione

Nel database RealTime è stata creata una collection di nome 'book_architettura _calcolatori', in cui sono raccolte le informazioni relative a capitoli (Figura 4.3). Ogni capitolo contiene al suo interno il nome, le informazioni e le domande del quiz per quel capitolo. Tutti i capitoli hanno la stessa struttura, ovvero sono caratterizzati dagli stessi attributi; un capitolo infatti è una serie di coppie chiave-valore, e le chiavi rimangono le stesse da un documento all'altro, mentre cambiano ovviamente i valori in base al reperto considerato. Un esempio di come è strutturato un capitolo è visibile nella Figura 4.4.

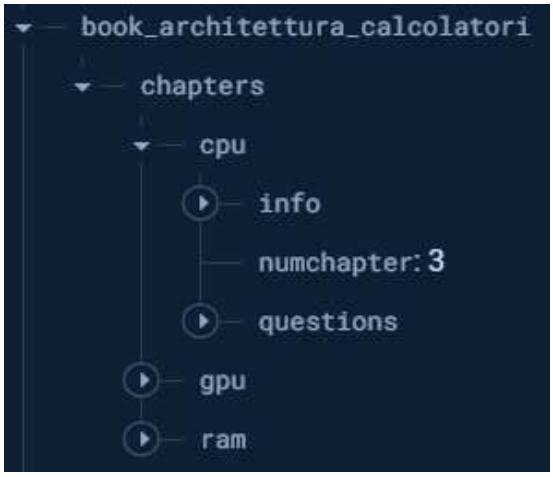


Figura 4.4: Esempio dei capitoli associati a un libro

Ogni capitolo è costituito da tre campi:

- name (stringa): il nome del capitolo
- info (stringa): la descrizione del capitolo
- questions (lista): le varie domande associate al capitolo

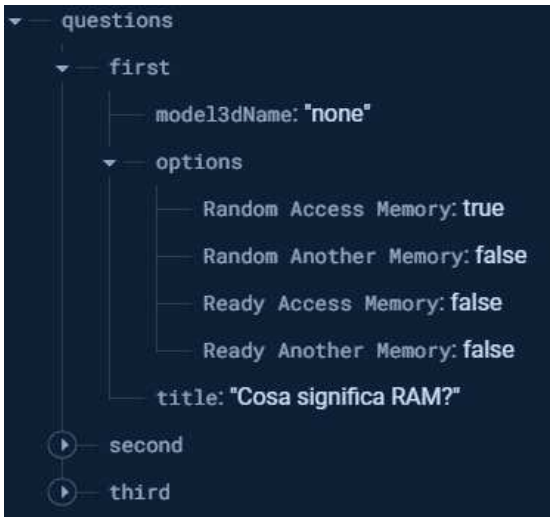


Figura 4.5: Esempio delle domande associate a un capitolo

Ogni domanda (Figura 4.5) è costituita da tre campi:

- model3dName (stringa): il nome del modello 3D associato alla domanda
- options (lista): le varie opzioni associate alla domanda
- title (stringa): il titolo della domanda

Nel database RealTime è presente anche una collection di nome 'statistics_book _architettura_calcolatori', in cui sono raccolte le statistiche e le informazioni per ogni utente registrato (Figura 4.6).

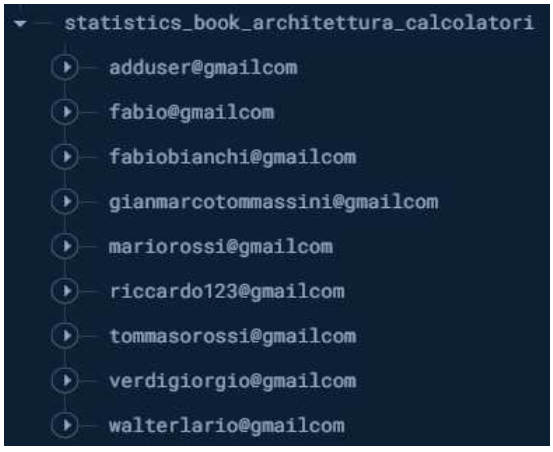


Figura 4.6: Esempio della lista degli utenti all'interno della sezione statistiche

Ogni statistica associata a un utente (Figura 4.7) è costituita da cinque campi:

- birthdate (stringa): la data di nascita

- email (stringa): l'email associata all'utente
- name (stringa): il nome dell'utente
- stats (lista): le percentuali di domande risposte correttamente per ogni capitolo
- surname (stringa): il cognome dell'utente

Figura 4.7: Esempio statistiche associate a un utente

Lo Storage è molto più semplice, in quanto viene utilizzato esclusivamente come servizio di archiviazione di oggetti, in questo caso in particolare per i modelli 3D. Nel database è presente una cartella, chiamata '3DModels' in cui sono presenti tutte i modelli 3D, in formato .glb e col nome che opzionalmente corrisponde al valore dell'attributo 'model3dName' nelle domande presenti nel RealTime Database. Nella figura 4.8 è presente uno screenshot dei dati all'interno dello Storage.

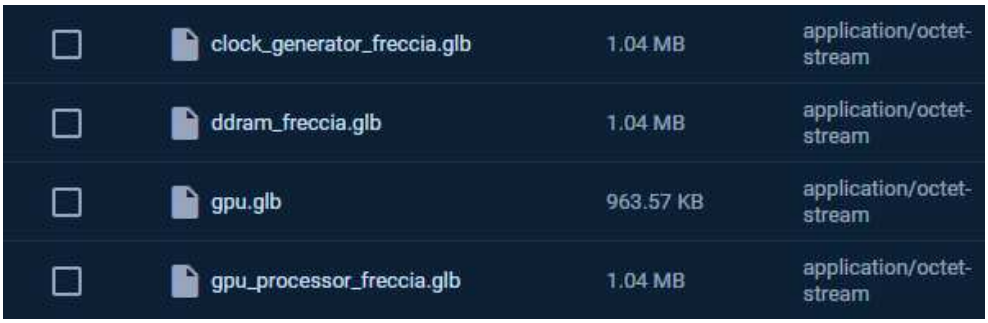


Figura 4.8: Screenshot all'interno del database Firebase Storage

4.3 Implementazione in Flutter

Questo progetto è stato sviluppato utilizzando Flutter che usa come linguaggio di programmazione Dart, che è sviluppato anch'esso da Google.

Ogni progetto Flutter ha una struttura di base, la quale evolve a seconda dell'applicazione che si sta progettando.

Nella Figura 4.9 è riportata la directory di progetto dell'app. I vari package che compongono il progetto sono stati suddivisi nel seguente modo:

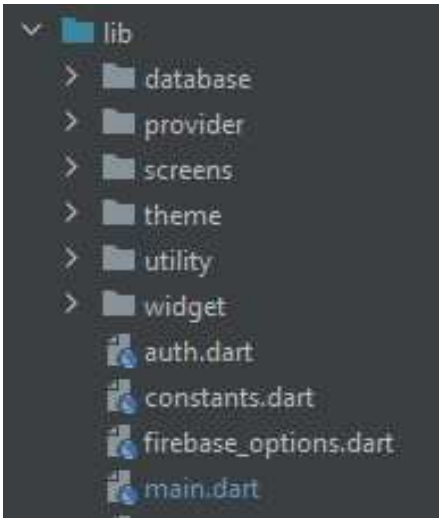


Figura 4.9: Screenshot della directory di progetto

- Nel package database sono presenti le classi degli oggetti utilizzati per gestire i dati e la gestione del collegamento con il repository per comunicare con la sorgente dei dati.
- Nel package provider sono presenti le classi che consentono di gestire lo stato dell'app in modo scalabile.
- Il package screens contiene le user interface dove verranno visualizzate le informazioni.
- Il package theme contiene la classe che gestisce il tema dell'applicazione.
- Il package utility contiene la classe per la gestione delle Permissions.
- Nel package widget sono presenti i metodi per la personalizzazione dei Widget stessi.

4.4 Logica dell'applicazione

Tale sezione tratta della logica di implementazione dell'app Learn AR, definendo per ogni singolo package tutta la logica presente all'interno di ogni classe.

4.4.1 La funzione main

```dart
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(MyApp());
}

class MyApp extends StatefulWidget {
  MyApp({super.key});

  @override
  State<MyApp> createState() => _MyAppState();
}
```

Figura 4.10: Screenshot della funzione main

Nel metodo main si vanno a inizializzare i servizi Firebase, cosicché poi durante l'esecuzione sia possibile utilizzarli.
Successivamente, sempre dentro la funzione main, viene lanciata l'applicazione; questo avviene con il metodo runApp(MyApp()). MyApp, che rappresenta l'istanza dell'applicazione, è, come tutto in Flutter, uno Widget.
Nel metodo main (Figura 4.10), nella prima istruzione viene eseguito il metodo 'WidgetsFlutterBinding.ensureInitialized'. WidgetFlutterBinding è una componente utilizzata dal Framework Flutter per comunicare con il Flutter Engine. Siccome 'Firebase.initializeApp' ha bisogno di utilizzare codice nativo per inizializzare i servizi Firebase, passando tramite i Platform Channels, prima si deve verificare che ci sia un'istanza di WidgetsBinding. Infatti, i Platform Channels si trovano nell'Engine, per questo risulta necessario avere a disposizione WidgetsBinding.

```
routes: {
  '/homepage': (context) => const Intro(),
  '/profile': (context) => const ProfilePage(),
  '/auth': (context) => const AuthPage(),
  '/ar': (context) => const ArPage(),
  '/quizhomepage': (context) => const SectionQuizPage(),
  '/statistics': (context) => const Statistics(),
},
```

Figura 4.11: Screenshot delle rotte della funzione main

In MyApp sono definite delle rotte, che serviranno poi durante l'esecuzione dell'app per navigare da una schermata all'altra (Figura 4.11).
La rotta iniziale è '/homepage', la quale istanzia il widget Intro, cioè la schermata in cui è possibile scegliere quale funzionalità dell'applicazione utilizzare.
La rotta '/profile' istanzia il widget ProfilePage in cui vengono visualizzate le informazioni dell'utente.
La rotta '/auth' istanzia il widget AuthPage in cui si può eseguire il login o la registrazione.
La rotta '/ar' istanzia ArPage, la sezione dedicata alla realtà aumentata, dove inquadrando un'immagine target viene visualizzato il modello 3D associato.
La rotta '/quizhomepage' istanzia SectionQuizPage, la sezione dedicata alla verifica delle conoscenze, in cui è possibile scegliere per quale capitolo avviare un nuovo quiz.
La rotta '/statistics' istanzia il widget Statistics, la sezione dedicata alla visualizzazione delle statistiche relative ai quiz svolti per ogni capitolo.

```
return MultiProvider(
  providers: [ChangeNotifierProvider(create: (context) => QuizProvider()),
              ChangeNotifierProvider(create: (context) => StatisticProvider()),
              ChangeNotifierProvider(create: (context) => ArProvider()),
              ChangeNotifierProvider(create: (context) => HomeProvider())],
```

Figura 4.12: Screenshot del multiprovider presente nella funzione main

E' stato utilizzato uno specifico provider (multiprovider) che unisce più provider in un unico albero di widget lineare (Figura 4.12).
Viene utilizzato per migliorare la leggibilità e ridurre la necessità di annidare più livelli di provider nel codice standard.
Il provider è un'estensione di InheritedWidget che rende più semplice la gestione delle informazioni nell'albero dei widget.

```
onGenerateRoute: (settings) {
  // If you push the PassArguments route
  if (settings.name == '/quizpage') {
    // Cast the arguments to the correct
    // type: ScreenArguments.
    final args = settings.arguments as ScreenArguments;
    // Then, extract the required data from
    // the arguments and pass the data to the
    // correct screen.
    return MaterialPageRoute(
      builder: (context) {
        return QuizPage(
          title: args.title,
          message: args.message,
          origin: args.origin,
        );  // QuizPage
      },
    );  // MaterialPageRoute
  }
```

Figura 4.13: Screenshot del passaggio di parametri nella funzione main

All'interno della funzione main è stato implementato anche il passaggio di dati tra schermate (Figura 4.13). All'interno di onGenerateRoute viene verificato che il nome della rotte si uguale a '/quizpage', viene fatto il cast degli arguments nel corretto tipo (ScreenArguments). Poi vengono estratti i dati dagli arguments e infine vengono passati allo screen corretto. Per questo passaggio viene utilizzato MaterialPageRoute.

```
home: AnimatedSplashScreen(
  splash: Column(
    children: [
      Image.asset('assets/learn_ar_logo.png',width: 60.0, height: 60.0,),
      Text('Learn AR',style: TextStyle(fontSize: 40.0, fontWeight: FontWeight.bold),),
    ],
  ),  // Column
  splashIconSize: 150.0,
  splashTransition: SplashTransition.fadeTransition,
  nextScreen: StreamBuilder(
      stream: Auth().authStateChanges,
      builder: (context, snapshot){
        if(snapshot.hasData){
          return Intro();
        }
        else {
          return AuthPage();
        }
      }),  // StreamBuilder
)  // AnimatedSplashScreen
```

Figura 4.14: Screenshot dello splash screen nella funzione main

L'ultima parte di codice presente all'interno della funzione main è l'implementazione dello splash screen (Figura 4.14), una schermata che viene visualizzata

all'avvio di un'app che mostra il logo dell'app ed è progettato per dare all'utente un'idea di cosa tratta l'app.

Oltre al logo è stato implementato il nextScreen, la schermata che viene visualizzata dopo la schermata di avvio dell'app. È stato utilizzaato il widget StreamBuilder che si costruisce da solo in base all'ultima istantanea di interazione con uno Stream; si possono verificare due condizioni: se nello snapshot sono presenti dati il nextScreen sarà la homepage altrimenti sarà la authpaage.

4.4.2 Il package model

Gli oggetti presenti all'interno del sistema sono i seguenti:

- **Chapter**: rappresenta un singolo capitolo presente nel database (Figura 4.15).

```dart
class Chapter{

  late final String id;
  late final String name;
  late final bool? isLock;

  // create constructor
  Chapter({
    required this.id,
    required this.name,
    this.isLock,
  });

  @override
  String toString(){
    return 'Chapter(id: $id, title: $name,)';
  }
}
```

Figura 4.15: Screenshot del model Chapter

Per questo oggetto l'id rappresenata l'identificativo del capitolo.

Gli attributi:

- name ⇒ nome del capitolo.
- isLock ⇒ indica se il capitolo è bloccato.

- **Question**: rappresenta una singola domanda presente nel database (Figura 4.16).

```dart
class Question{

  late final String id;
  late final String title;
  late final Map<String, bool> options;
  late final String model3dName;

  Question({
    required this.id,
    required this.title,
    required this.options,
    required this.model3dName
  });

  @override
  String toString(){
    return 'Question(id: $id, title: $title, options: $options)';
  }
}
```

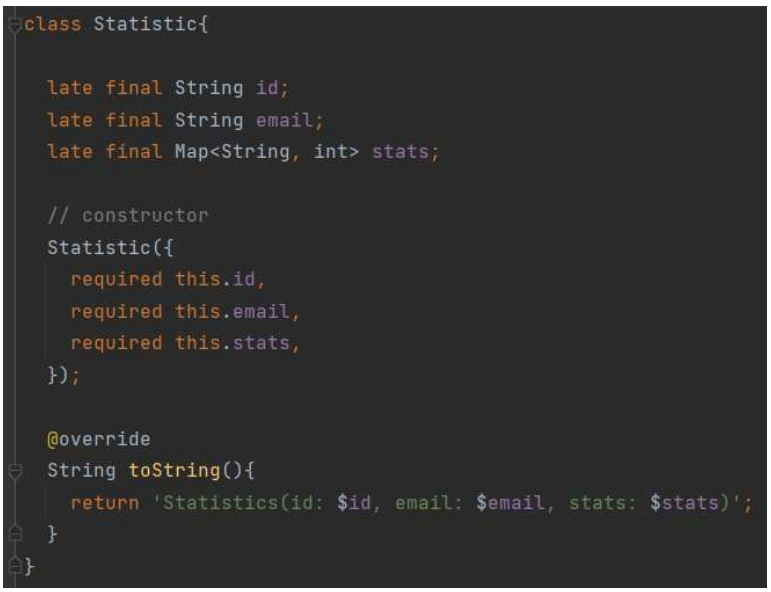
Figura 4.16: Screenshot del model Question

Per questo oggetto l'id rappresenta l'identificativo della domanda.

Gli attributi:

- title ⇒ titolo della domanda.
- options ⇒ mappa composta dalle possibili risposte con associato se la risposte è corretta.
- model3dName ⇒ nome del modello 3D associato alla domanda.

- **Statistic**: rappresenta la statistica associata a un singolo utente presente nel database (Figura 4.17).



```dart
class Statistic{

  late final String id;
  late final String email;
  late final Map<String, int> stats;

  // constructor
  Statistic({
    required this.id,
    required this.email,
    required this.stats,
  });

  @override
  String toString(){
    return 'Statistics(id: $id, email: $email, stats: $stats)';
  }
}
```

Figura 4.17: Screenshot del model Statistic

Per questo oggetto l'id rappresenata l'identificativo della statistica.

Gli attributi:

- email ⇒ email dell'utente.
- stats ⇒ mappa composta dal nome del capitolo con associato la percentuale di riposte corrette.
- model3dName ⇒ nome del modello 3D associato alla domanda.

- **UserModel**: rappresenta chiunque usufruisca dei servizi forniti dall'applicazione (Figura 4.18).

```dart
class UserModel{

  late final String id;
  late final String email;
  late final String name;
  late final String surname;
  late final String birthDate;

  // constructor
  UserModel({
    required this.id,
    required this.email,
    required this.name,
    required this.surname,
    required this.birthDate,
  });

  @override
  String toString(){
    return 'User(id: $id, email: $email, name: $name, surname: $surname, birthdate: $birthDate)';
  }
}
```

Figura 4.18: Screenshot del model UserModel

A ogni user viene assegnato un id al momento della registrazione per identificarlo. Gli attributi e-mail e password rappresentano insieme le credenziali di accesso da utilizzare al momento del login. Gli attributi name, surname, birthDate rappresenatano informazioni dell'utente.

- **Info**: rappresenta una singola informazione (Figura 4.19).

```dart
class Info{

  late final String id;
  late final String descr;

  // create constructor
  Info({
    required this.id,
    required this.descr,
  });

}
```

Figura 4.19: Screenshot del model Info

Per questo oggetto l'id rappresenta l'identificativo dell'informazione e descr rappresenta la descrizione.

4.4.3 Login, Registrazione e Logout

La parte riguardante l'autenticazione è stata di facile implementazione, in quanto Firebase mette a disposizione il plugin firebase_auth che rende molto più semplice effettuare il login, il logout e la registrazione semplicemente usando dei metodi che sono già predefiniti nel plugin (Figura 4.20).

```dart
class Auth{
  final FirebaseAuth _firebaseAuth = FirebaseAuth.instance;

  User? get currentUser => _firebaseAuth.currentUser;

  Stream<User?> get authStateChanges => _firebaseAuth.authStateChanges();

  // sign in with email and password
  Future<void> signInWithEmailAndPassword({required String email, required String password}) async{
    await _firebaseAuth.signInWithEmailAndPassword(email: email, password: password);
  }

  // create new user with email and password
  Future<void> createUserWithEmailAndPassword({required String email, required String password}) async{
    await _firebaseAuth.createUserWithEmailAndPassword(email: email, password: password);
  }

  // sign out
  Future<void> signOut() async{
    await _firebaseAuth.signOut();
  }
}
```

Figura 4.20: Screenshot della classe Auth

Per registrarsi sarà sufficiente inserire Nome, Cognome, indirizzo email (univoco), una password (con minimo 6 caratteri), e la data di nascita, mentre il Login è definito nel file 'authpage.dart', e per autenticarsi sarà sufficiente inserire email e password.

I dati dell'utente sono passati tramite un'oggetto, cioè un'istanza della classe 'UserModel', che si trova nel file 'user_model.dart'.

Una volta eseguito l'accesso l'utente avrà la possibilità si eseguire il logout premendo sull'apposito tasto dell'AppBar.

Dal punto di vista implementativo, anche se la maggior parte dei metodi è intrinseco nei plugin di firebase_auth è sicuramente importante anche solo accennare i metodi che permettono all'utente di effettuare il login, registrarsi ed effettuare il logout. La schermata di Login altro non è che una form in cui inserire email e password per autenticarsi. Una volta premuto il bottone di Login partirà il metodo signIn, che prende come parametri l'email e la password inseriti nella form, poi li valuta e manda i valori della form a Firebase mediante il metodo signInWithEmailAndPassword che andrà a controllare se esiste un utente nel database che corrisponda a questi dati, dopodiché l'utente verrà portato nella schermata di profilo. Durante questo processo si verificano gli eventuali codici di errore e, se presenti, vengono comunicati all'utente tramite SnackBar. Tutto ciò è visibile nella Figura 4.21.

```
Future<void> signIn() async {
  try {
    await Auth().signInWithEmailAndPassword(
        email: _email.text, password: _password.text);

  } on FirebaseAuthException catch (error) {
    // Handle Errors here.
    var errorCode = error.code;
    var errorMessage = error.message;
    log('errorcode -> $errorCode');
    if (errorCode == 'INVALID_LOGIN_CREDENTIALS') {
      //alert('Wrong password.');
      ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
        content: Text('Email or Password not correct'),
        behavior: SnackBarBehavior.floating,
        margin: EdgeInsets.symmetric(vertical: 20.0, horizontal: 20.0),
      ));  // SnackBar
    } else if(errorCode == 'invalid-email'){
      ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
        content: Text('Email not correct'),
        behavior: SnackBarBehavior.floating,
        margin: EdgeInsets.symmetric(vertical: 20.0, horizontal: 20.0),
      ));  // SnackBar
    } else if(errorCode == 'too-many-requests'){
      ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
        content: Text('We have blocked all requests from this device due to unusual activity. Try again later'),
        behavior: SnackBarBehavior.floating,
        margin: EdgeInsets.symmetric(vertical: 20.0, horizontal: 20.0),
      ));  // SnackBar
    }
  }
```

Figura 4.21: Screenshot del metodo signIn

Nella schermata di Login sarà anche possibile accedere alla schermata di registrazione nel caso in cui non si possedesse l'account per autenticarsi tramite un bottone. Per quanto riguarda la schermata di registrazione, avremo un'ulteriore form dove potremmo inserire: Nome, cognome, email, password e data di nascita. Al click sul pulsante 'Registrati' verrà eseguito il metodo signUp, che prende come parametri sempre l'email e la password, verifica che questi campi siano validi e verrà eseguito il metodo createUserWithEmailAndPassword (Figura 4.22).

```
Future<void> createUser() async {
  try {
    errorCodeCreate = '';
    await Auth().createUserWithEmailAndPassword(
        email: _email.text, password: _password.text);
  } on FirebaseAuthException catch (error) {
    var errorMessage = error.message;
    errorCodeCreate = error.code;
    if (errorCodeCreate == 'email-already-in-use') {
      //alert('Wrong password.');
      ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
        content: Text('The email address is already in use by another account'),
        behavior: SnackBarBehavior.floating,
        margin: EdgeInsets.symmetric(vertical: 20.0, horizontal: 20.0),
      ));  // SnackBar
    } else if(errorCodeCreate == 'too-many-requests'){
      ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
        content: Text('We have blocked all requests from this device due to unusual activity. Try again later'),
        behavior: SnackBarBehavior.floating,
        margin: EdgeInsets.symmetric(vertical: 20.0, horizontal: 20.0),
      ));  // SnackBar
    }
  }
}
```

Figura 4.22: Screenshot del metodo createUser

Infine, una volta giunti nella homepage sarà anche presente il metodo che permetterà all'utente di effettuare il logout semplicemente utilizzando il metodo signOut, che poi manderà l'utente nuovamente nella schermata di Login (Figura 4.23).

```
  // sign out
  Future<void> signOut() async{
    await _firebaseAuth.signOut();
  }
}
```

Figura 4.23: Screenshot del metodo signOut

4.4.4 Permissions

Per l'utilizzo completo dell'applicazione Learn AR è necessario l'utilizzo della fotocamera, per questo, è stata implementata la gestione della richiesta di accesso alla camera del telefono.

All'interno della classe PermissionUtility (Figura 4.24) sono presenti due metodi: requestPermission e checkPermissionStatus.

Il metodo requestPermission verifica se il permesso di utilizzo della fotocamera è negato. Se il permesso è negato ('isDenied'), viene mostrata una finestra di dialogo all'utente per richiedere il permesso di localizzazione. La funzione attende (grazie ad await) fino a quando l'utente non risponde alla richiesta di permesso.

Il metodo checkPermissionStatus restituisce true se l'autorizzazione viene concessa e false in caso contrario.

```
class PermissionUtility{
  Future<void> requestPermission() async {
    const permission = Permission.camera;

    if (await permission.isDenied) {
      await permission.request();
    }
  }

  Future<bool> checkPermissionStatus() async {
    const permission = Permission.camera;

    return await permission.status.isGranted;
  }
}
```

Figura 4.24: Screenshot della classe Permissions

4.4.5 Il metodo fetchStatistic

La funzione fetchStatistic recupera le statistiche di un utente dal database, filtra i dati in base all'email, gestisce eventuali chiavi 'no data yet', e restituisce l'oggetto Statistic appropriato. Questo permette all'applicazione di ottenere e visualizzare le statistiche dell'utente in modo efficiente e coerente.

La funzione è dichiarata come asincrona (async) e restituisce un Future<Statistic> e accetta un parametro email di tipo String.
Inizialmente, viene creato un riferimento al nodo /statistics_book_architettura_calcolatori nel database Firebase.
Converte una stringa URL (urlStringStatistics) in un oggetto Uri, che rappresenta l'endpoint da cui verranno recuperate le statistiche.
Viene eseguita una richiesta HTTP GET all'URL specificato. Una volta ricevuta la risposta, il codice nel blocco then viene eseguito.
La risposta HTTP, in formato JSON, viene decodificata in una Map<String, dynamic> (Figura 4.25).

```
Future<Statistic> fetchStatistic(String email) async{
  var userNameRef =  myRootRef.child('/statistics_book_architettura_calcolatori');
  var urlStatistics = Uri.parse(urlStringStatistics);
  return http.get(urlStatistics).then((response){
    var data = json.decode(response.body) as Map<String, dynamic>;

    List<Statistic> newStatistics = [];
```

Figura 4.25: Screenshot del metodo fetchStatistic (prima parte)

Poi viene utilizzato un foreach per iterare su ogni coppia chiave-valore nella mappa dei dati decodificati.
Per ogni elemento, viene verificato se l'email associata ai dati corrisponde all'e-mail fornita come parametro.
Viene controllato se esistono dati statistici per l'utente.

Se i dati statistici contengono la chiave 'no data yet', questa chiave viene rimossa dal database per l'utente specifico.
Se i dati statistici non contengono 'no data yet', viene creato un nuovo oggetto Statistic con i dati e aggiunto alla lista newStatistics.

Viene creato un oggetto Statistic di default con la chiave 'no data yet'. Se la lista newStatistics contiene elementi, viene restituito il primo elemento della lista, altrimenti viene restituita la statistica di default (Figura 4.26).

```
    data.forEach((key, value){
      if(value['email'] == email){
        if(value['stats'] != null){
          if(Map.castFrom(value['stats']).keys.contains('no data yet') ){
            userNameRef.child('${email.replaceAll('.', '')}').child('stats').child('no data yet').remove();
          }
          else{
            var newStatistic = Statistic(
                id: key,
                email: value['email'],
                stats: Map.castFrom(value['stats']));
            newStatistics.add(newStatistic);

          }
        }
      }
    });
    var defaultStatistic = Statistic(id: '', email: '', stats: <String, int>{'no data yet' : 0});
    return newStatistics.isNotEmpty ? newStatistics[0] : defaultStatistic;
  });
}
```

Figura 4.26: Screenshot del metodo fetchStatistic (seconda parte)

4.4.6 Il metodo addStatistic

La funzione addStatistic aggiorna le statistiche di un utente nel database. Se la statistica è già presente, viene aggiornata; altrimenti, viene aggiunta una nuova statistica. La funzione gestisce anche la rimozione di un placeholder ('no data yet') se esiste, assicurandosi che le statistiche dell'utente siano sempre aggiornate correttamente.

Piu nel dettaglio, la funzione addStatistic è una funzione asincrona (async) che non restituisce nulla (Future<void>) e prende un parametro di tipo Statistic. Nella prima istruzione viene creato un riferimento a una particolare posizione nel database remoto (myRootRef.child('/statistics_book_architettura_calcolatori')).
La funzione fetchStatistic viene chiamata con l'email contenuta nell'oggetto statistic. Questo recupera le statistiche associate a quell'email e le assegna alla variabile s.
Poi, viene inizializzata a false la variabile booleana isPresent. La variabile map è una mappa vuota (Map<String, int>) che verrà utilizzata per memorizzare le statistiche (Figura 4.27).

```
//add statistic, update statistic
Future<void> addStatistic(Statistic statistic) async {
  var userNameRef =  myRootRef.child('/statistics_book_architettura_calcolatori');

  var s = await fetchStatistic(statistic.email);
  var isPresent = false;
  Map<String, int> map ={};
```

Figura 4.27: Screenshot del metodo addStatistic (prima parte)

Successivamente viene eseguita la verifica e l'aggiornamento delle statistiche (Figura 4.28).
Se s.stats non è vuoto (s.stats.isNotEmpty):

- viene iterata ogni coppia chiave-valore nelle statistiche recuperate (s.stats.forEach((key, value) ...),
- ogni chiave-valore viene aggiunta alla mappa map,
- se la chiave della statistica corrente corrisponde alla chiave della statistica da aggiungere (statistic.stats.keys == key), viene aggiornata la mappa con il nuovo valore e isPresent viene impostato a true .

```
if(s.stats.isNotEmpty){
  s.stats.forEach((key, value) {
    map[key] = value;
    if (statistic.stats.keys == key) {
      map.update(key, (value) => statistic.stats.values.first);
      isPresent = true;
    }
  });
```

Figura 4.28: Screenshot del metodo addStatistic (seconda parte)

Infine, viene eseguito l'aggiornamento del database remoto di Firebase (Figura 4.29).
Se isPresent è true, viene aggiornato il nodo nel database con la mappa delle statistiche (userNameRef.child('$statistic.email.replaceAll('.', '')').update('stats': map,);).
Se isPresent è false: viene aggiunta la nuova coppia chiave-valore alla mappa, viene rimossa la chiave 'no data yet' dalla mappa (se presente) e, infine, viene quindi aggiornato il nodo nel database con la nuova mappa delle statistiche.

```
    if(isPresent == true){
      userNameRef.child('${statistic.email.replaceAll('.', '')}').update({
        'stats': map,
      });
    }
    else{
      map[statistic.stats.keys.first] = statistic.stats.values.first;
      userNameRef.child('${statistic.email.replaceAll('.', '')}').update({
        'stats': map,
      });
    }
  }
}
```

Figura 4.29: Screenshot del metodo addStatistic (terza parte)

4.4.7 Connessione Flutter con Unity

Nel capitolo 3 è stata citata la libreria flutter_unity_widget [15] descrivendone le funzionalità l'architettura e i servizi che venivano messi a disposizione.

La creazione di un'applicazione completa di realtà aumentata (AR) Flutter comporta diversi passaggi e può essere piuttosto estesa. La combinazione di Unity con Flutter consente agli sviluppatori di creare applicazioni AR che sfruttano la facilità d'uso di Flutter e le potenti capacità di rendering e simulazione 3D di Unity.

L'integrazione della realtà aumentata nelle applicazioni Flutter offre il vantaggio dello sviluppo multipiattaforma, in particolare Flutter consente di sviluppare app AR che funzionano su dispositivi Android e iOS, riducendo i tempi e gli sforzi di sviluppo. In questo capitolo verrà discusso in maniera approfondita l'implementazione della realtà aumentata nell'applicazione.

Inizializzazione e configurazione

UnityWidget è un widget che consente di integrare contenuti di Unity all'interno di un'app Flutter. La sua inizializzazione è visibile nella Figura 4.30.

- **onUnityCreated**

  Il parametro onUnityCreated prende una funzione di callback onUnityCreated. La funzione onUnityCreated viene chiamata quando il widget Unity è stato creato e inizializzato.

  Questa funzione viene usata per eseguire ulteriori configurazioni o inizializzazioni necessarie dopo la creazione del widget Unity.

- **onUnityMessage**

  Un altro parametro è onUnityMessage, che prende una funzione di callback onUnityMessage. La funzione onUnityMessage viene chiamata quando Unity invia un messaggio al codice Flutter.

  È utile per la comunicazione tra l'app Flutter e il contenuto Unity, ad esempio per ricevere dati o eventi generati da Unity.

- **onUnitySceneLoaded**

  Il parametro onUnitySceneLoaded prende una funzione di callback onUnitySceneLoaded. La funzione onUnitySceneLoaded viene chiamata quando una scena Unity è stata caricata completamente.

  Questa funzione viene usata per eseguire operazioni dopo che una scena è stata caricata, come iniziare l'interazione con gli oggetti della scena.

- **fullscreen**

  La proprietà fullscreen imposta se il widget Unity deve essere visualizzato a schermo intero. Nel caso specifico, fullscreen è impostato su false, quindi

il widget Unity non sarà a schermo intero ma verrà visualizzato come parte del layout Flutter.

```
children: <Widget>[
  UnityWidget(
    onUnityCreated: onUnityCreated,
    onUnityMessage: onUnityMessage,
    onUnitySceneLoaded: onUnitySceneLoaded,
    fullscreen: false,
  ),  // UnityWidget
```

Figura 4.30: Screenshot dell'UnityWidget

Comunicazione da Flutter a Unity

Questa sezione tratta della comunicazione da Flutter a Unity per cambiare la scena in Unity in base a determinate condizioni.

Come è visibile nella Figura 4.31, l'applicazione Learn AR permette all'utente di interagire con diverse parti di un computer (RAM, GPU, CPU) rappresentate in una scena Unity. In base all'interazione dell'utente, la variabile info viene impostata su 'ram', 'gpu' o 'cpu'. La funzione changeScene cambia la scena in Unity per mostrare la parte corrispondente del computer. Se l'utente vuole tornare alla scena iniziale, la scena viene reimpostata su 'SampleScene'.

Nello specifico, la funzione changeScene viene utilizzata per determinare quale scena deve essere caricata in Unity in base a determinate condizioni.

L'istruzione "if (scene == 'SampleScene')" verifica se la scena corrente è 'SampleScene'.

Se la scena corrente è 'SampleScene', il codice verifica il valore della variabile info:

- Se info è 'ram', viene usata setState per aggiornare la variabile scene a 'InteractWithObject'.
- Se info è 'gpu', viene usata setState per aggiornare la variabile scene a 'InteractWithGpu'.
- Se info è 'cpu', viene usata setState per aggiornare la variabile scene a 'InteractWithCpu'.

Invece, se la scena corrente non è 'SampleScene', usa setState per aggiornare la variabile scene a 'SampleScene'.

Infine eseguendo "_unityWidgetController.postMessage('Gamemanager', 'ChangeTheSceneNow', scene);" viene inviato un messaggio al Gamemanager in Unity, chiamando il metodo ChangeTheSceneNow e passando il nome della scena (scene) che deve essere caricata.

Il metodo _unityWidgetController.postMessage invia un messaggio da Flutter a Unity. I parametri richiesti sono:

- Il nome del GameObject in Unity che deve ricevere il messaggio ('Gamemanager').
- Il nome del metodo nel GameObject che deve essere chiamato ('ChangeTheSceneNow').
- Il messaggio da inviare, che in questo caso è il nome della scena (scene).

Un dettagli aggiuntivo va rivolto alla funzione setState dato che viene utilizzata per aggiornare lo stato interno di un widget Flutter e richiedere un rebuild del widget. Ogni volta che setState viene chiamato, Flutter ridisegna il widget con il nuovo stato.

```
// Communcation from Flutter to Unity
void changeScene () {
  if(scene == 'SampleScene'){
    if(info == 'ram'){
      setState(() {
        scene = 'InteractWithObject';
      });
    }
    if(info == 'gpu'){
      setState(() {
        scene = 'InteractWithGpu';
      });
    }
    if(info == 'cpu'){
      setState(() {
        scene = 'InteractWithCpu';
      });
    }
  }
  else{
    setState(() {
      scene = 'SampleScene';
    });

  }

  _unityWidgetController.postMessage('Gamemanager', 'ChangeTheSceneNow', scene);
}
```

Figura 4.31: Screenshot del metodo changeScene

In Unity, viene utilizzato il seguente script C# per eseguire il cambiamento di scena (Figura 4.32).

```
using System.Collections;
using System.Collections.Generic;
using FlutterUnityIntegration;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MyGameManager: MonoBehaviour
{
    public void ChangeTheSceneNow(string sceneName)
    {
        Debug.Log("Let's change scene to: " + sceneName);
        SceneManager.LoadScene(sceneName, LoadSceneMode.Single);
    }
}
```

Figura 4.32: Screenshot del GameManager

Comunicazione da Unity a Flutter

Questa parte tratta la comunicazione da Unity a Flutter per gestire i messaggi ricevuti da Unity. Quando Unity invia un messaggio a Flutter, questo viene ricevuto e gestito dalla funzione onUnityMessage.

Come è visibile nella Figura 4.33, l'applicazione permette di interagire su diverse componenti di un computer (GPU, CPU, RAM) e che queste componenti sono rappresentate visivamente in una scena Unity. Quando l'utente interagisce con la scena Unity, Unity invia un messaggio a Flutter con il nome della componente selezionata ('gpu', 'cpu' o 'ram'). La funzione onUnityMessage riceve questo messaggio e aggiorna l'interfaccia utente di Flutter per mostrare informazioni dettagliate sulla componente selezionata.

Più nel dettaglio la funzione on UnityMessage viene chiamata quando Unity invia un messaggio a Flutter.

Per prima cosa attraverso l'istruzione

"if (message.toString() == 'gpu' || message.toString() == 'cpu' || message.toString() == 'ram')"

viene verificata che la provenienza del messaggio ricevuto è uno tra 'gpu', 'cpu' o 'ram'.
Se la condizione è vera, viene chiamato setState per aggiornare lo stato dell'applicazione (info = message.toString();).

La variabile info viene aggiornata con il valore del messaggio ricevuto. Poi, la variabile visibilityInfo viene impostata su true, il che rende visibile una parte dell'interfaccia utente fino a quel momento nascosta.

La funzione setState funzione viene utilizzata per notificare a Flutter che lo stato interno del widget è cambiato e che il widget deve essere ricostruito per riflettere i nuovi dati. In questo caso, aggiorna le variabili info e visibilityInfo e provoca il rebuild del widget per mostrare i cambiamenti.

```
// Communication from Unity to Flutter
void onUnityMessage(message) {
  log('unity1 ->'+ message.toString());
  print('Received message from unity: ${message.toString()}');
  if(message.toString() == 'gpu' || message.toString() == 'cpu' || message.toString() == 'ram' ){
    setState(() {
      info = message.toString();
      visibilityInfo = true;
    });
  }
}
```

Figura 4.33: Screenshot del metodo onUnityMessage

Il messaggio proveniente da Unity avviene grazie al metodo UnityMessageManager.SendMeassageToFlutter, un metodo utilizzato per inviare messaggi da Unity a Flutter. Questa funzione permette di stabilire una comunicazione bidirezionale tra un'applicazione Unity integrata e la parte Flutter dell'app.
Grazie a questa funzione una volta che viene riconosciuta un immagine target viene inviato a Flutter una stringa contente quale immagine è stata riconosciuta; questo è visibile nella Figura 4.34.

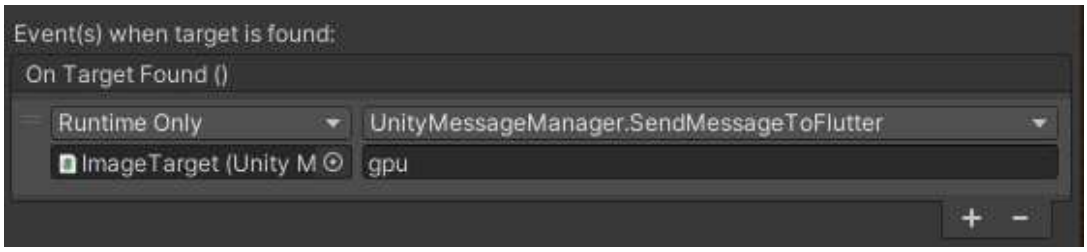


Figura 4.34: Screenshot UnityMessageManager.SendMeassageToFlutter

Callback

La funzione onUnityCreated fornisce una callback che viene chiamata quando il controller di Unity è stato creato. Questa funzione di callback associa il controller creato a una variabile di istanza in Flutter in modo che possa essere utilizzato altrove nel codice per interagire con Unity.

Per comunicare con Unity (ad esempio, per inviare messaggi o controllare la scena), è necessario un UnityWidgetController. Quando il widget Unity viene creato, il controller viene passato a questa funzione di callback, permettendo di memorizzarlo e utilizzarlo per future interazioni (Figura 4.35).

```
// Callback that connects the created controller to the unity controller
void onUnityCreated(controller) {
  this._unityWidgetController = controller;
}
```

Figura 4.35: Screenshot del metodo onUnityCreated

Scene Loaded

La funzione onUnitySceneLoaded viene chiamata quando una nuova scena Unity viene caricata in un'app Flutter. Questa funzione prende come parametro un oggetto SceneLoaded chiamato sceneInfo, che contenere informazioni sulla scena caricata (Figura 4.36).

In questo caso, monitora il caricamento delle scene Unity all'interno di un'app Flutter, facilitando il debug e la verifica del corretto funzionamento del caricamento delle scene.

```
// Communication from Unity when new scene is loaded to Flutter
void onUnitySceneLoaded(SceneLoaded? sceneInfo) {
  print('Received scene loaded from unity: ${sceneInfo?.name}');
  print('Received scene loaded from unity buildIndex: ${sceneInfo?.buildIndex}');
}
```

Figura 4.36: Screenshot del metodo onUnitySceneLoaded

4.4.8 Integrazione Vuforia Engine con Unity

Questa sezione tratta di come è stato utilizzato Vuforia Engine all'interno di Unity [17] [18].
Per prima cosa dopo aver creato il progetto Unity, è stato aggiunta l'estensione Vuforia Engine, che è un pacchetto di risorse Unity che può essere importato nel progetto Unity.
Il pacchetto è stato scaricato dal menu Assets -> Import Package -> Custom Package.
Dopo l'importazione il motore Vuforia compare nel menu GameObject.
Un aspetto cruciale è che ogni scena che traccia ed esegue il rendering del contenuto aumentato richiede un ARCamera GameObject, un componente essenziale per lo sviluppo di applicazioni di Realtà Aumentata. Essa è responsabile della cattura delle immagini del mondo reale e della sovrapposizione degli oggetti virtuali su di esse, creando così un'esperienza di AR.

Un altro elemento essenziale è l'aggiunta della una chiave di licenza Vuforia Developer nel campo App License Key (Figura 4.37).

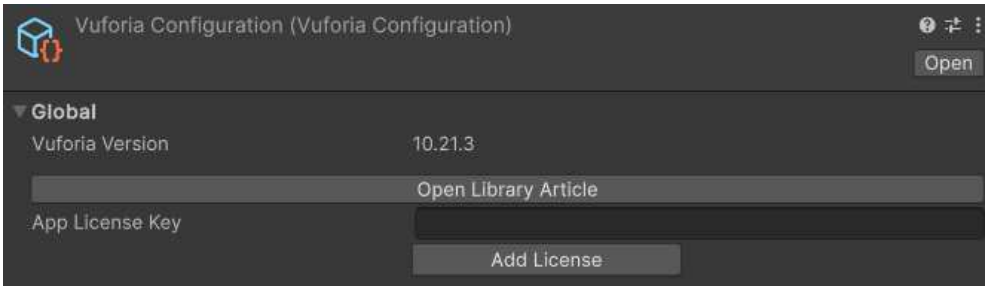


Figura 4.37: Screenshot dell'App License Key

Dopo aver attivato Vuforia Engine in Unity, sono state aggiunge funzionalità dal menu Vuforia Engine al progetto da Unity GameObject Menu.
La funzionalità utilizzata per l'applicazione è l'Image Target. Per aggiungere questa funzionalità si è seguito questo percorso GameObject -> Vuforia Engine -> Image Target (Figura 4.38).

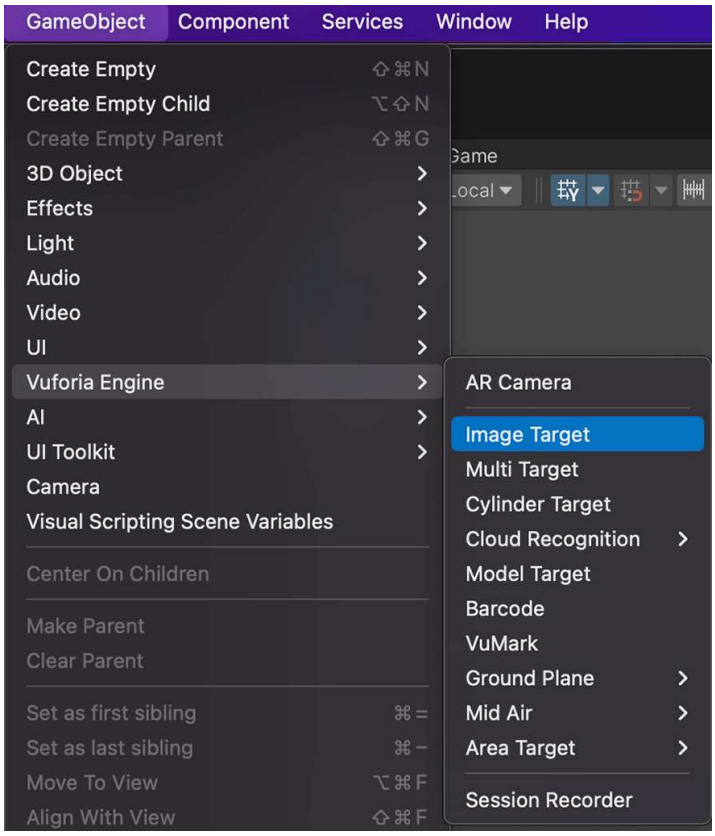


Figura 4.38: Screenshot del percorso per l'Image Target

Poi è stato selezionato l'Image Target GameObject dalla Gerarchia e scelto dal menù a discesa il tipo "From Database" indicando il database creato con il Target Manager, uno strumento Web che consente di creare e gestire database di destinazione sul portale degli sviluppatori di Vuforia.
La Figura 4.39 mostra il database creato con il Target Manager.

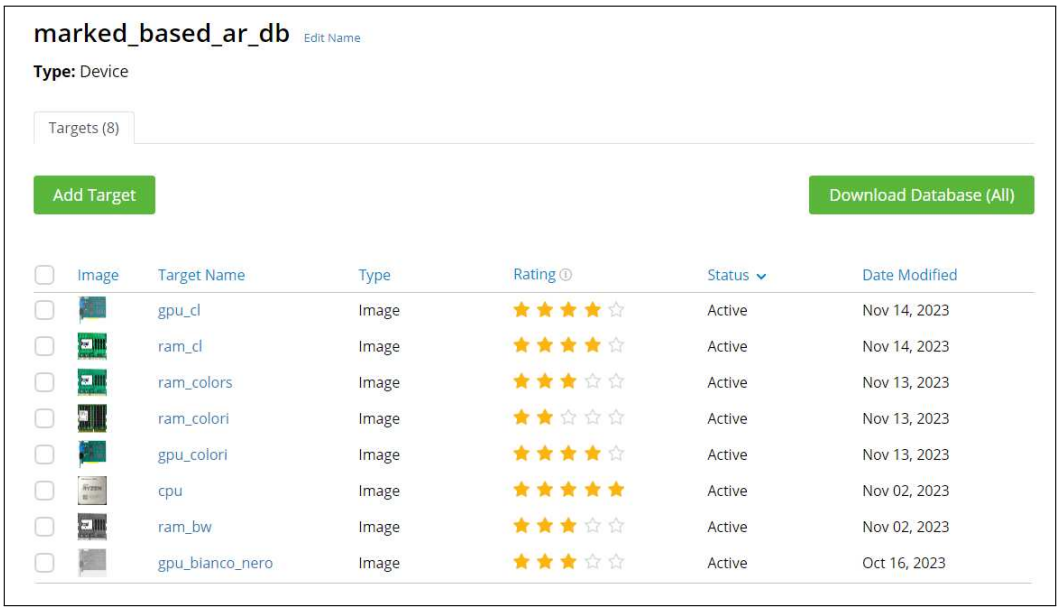


Figura 4.39: Screenshot del database creato con il Target Manager

Dopo aver associato il database all'Image Target GameObject è stato scelto quale immagine associare (Figura 4.40).

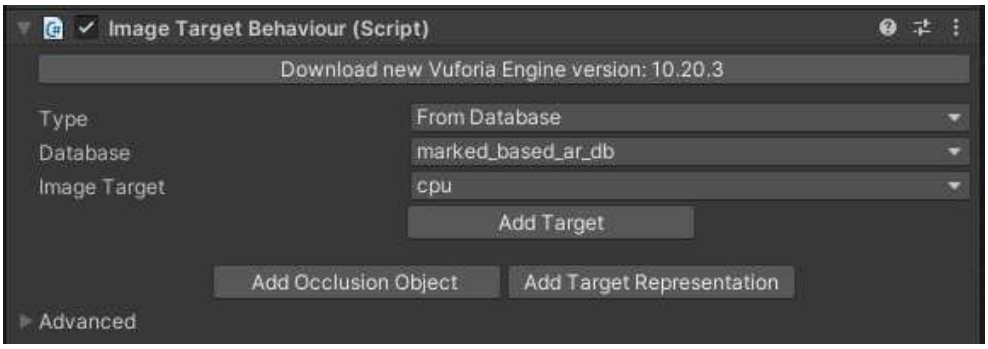


Figura 4.40: Screenshot dell'Image Target Behaviour

Il progetto Unity contiene tre Image Target diverse: una per il riconoscimento della GPU, una per la CPU e una per la RAM. Oltre a questo sono presenti l'ARCamera e il GameManager (Figura 4.41).

Figura 4.41: Screenshot del progetto Unity

4.5 ScreenShoot delle schermate

In questa sezione sono presenti gli screenshot delle varie schermate, al fine di mostrare il risultato finale dell'applicazione Learn AR.

Figura 4.42: Splash Screen

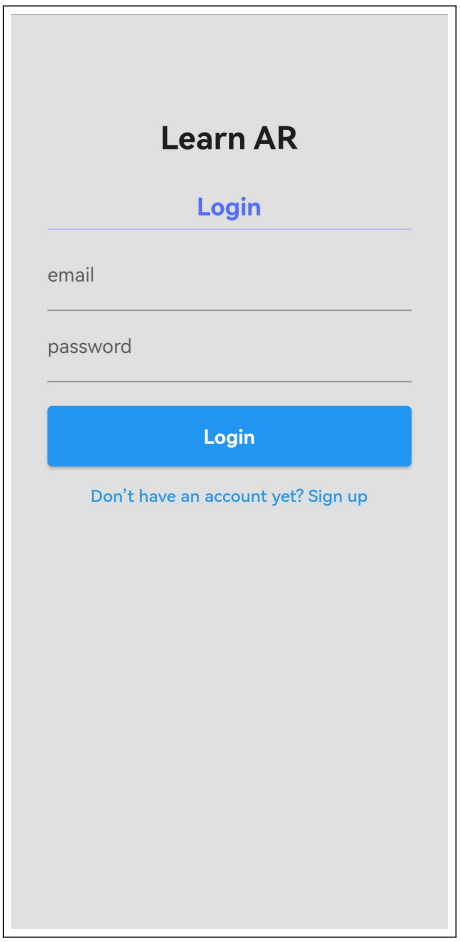


Figura 4.43: Autenticazione

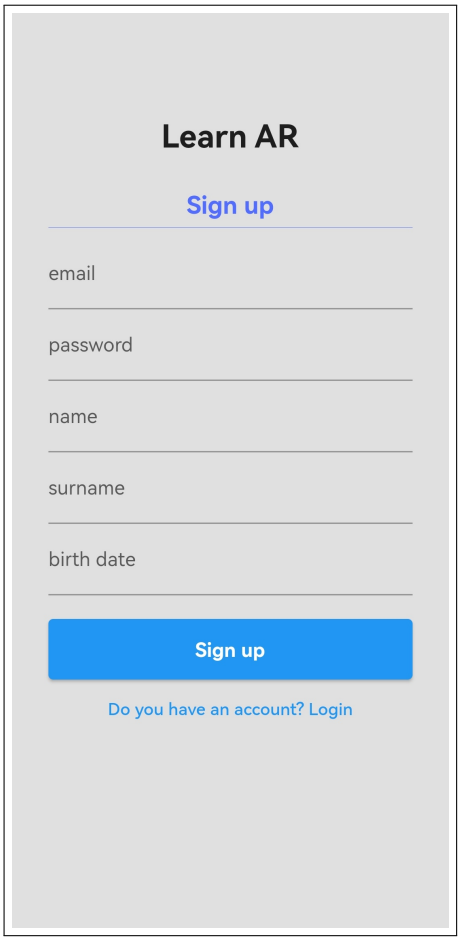


Figura 4.44: Registrazione

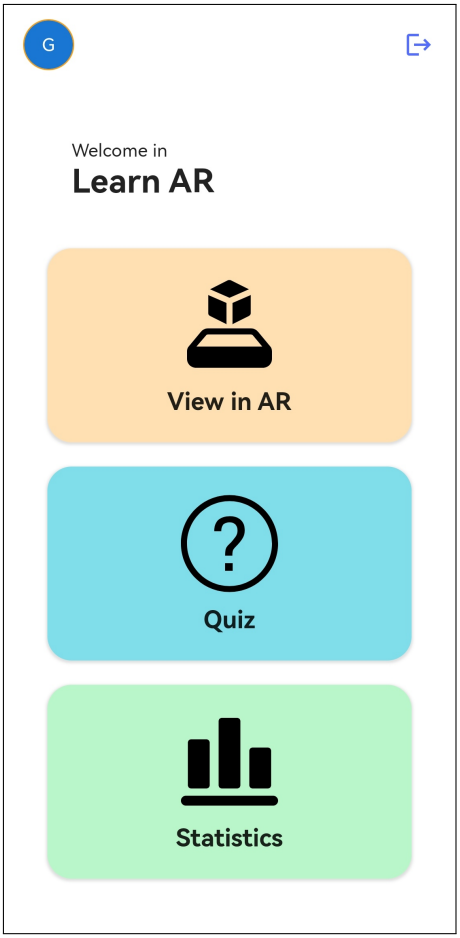


Figura 4.45: Homepage

Figura 4.46: Profile

Figura 4.47: Avvio Unity

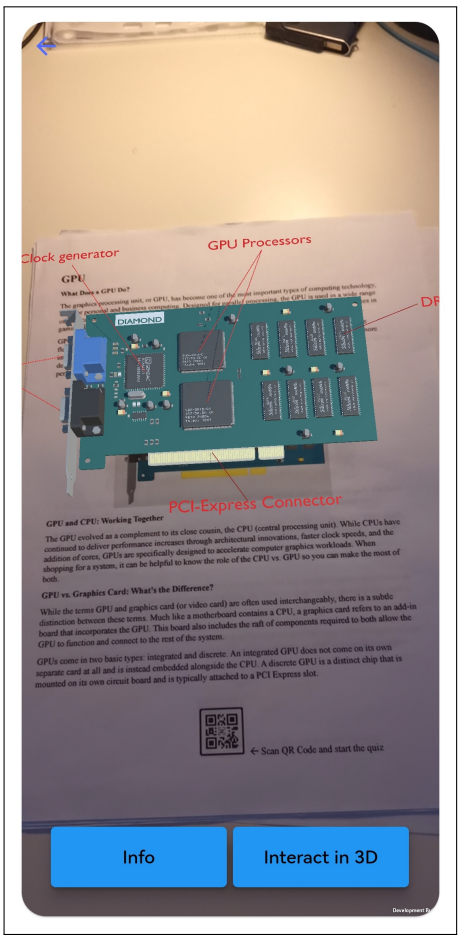


Figura 4.48: Visualizzazione Modello 3D

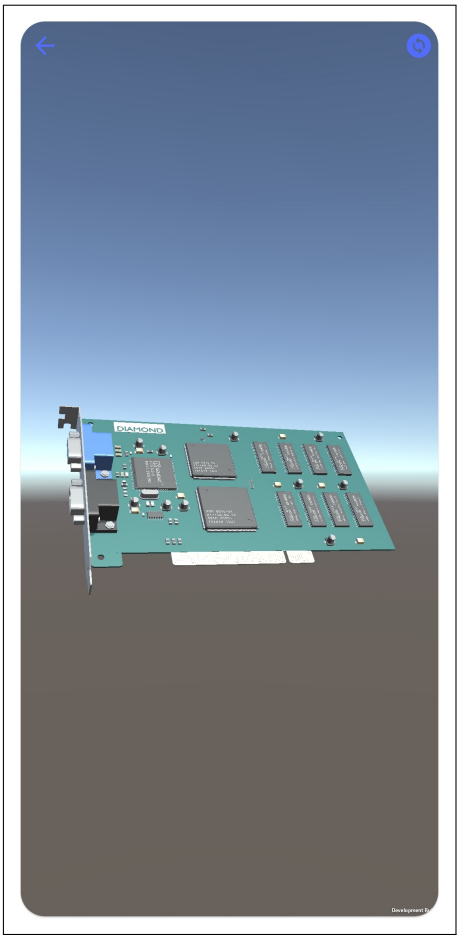


Figura 4.49: Interazione Modello 3D

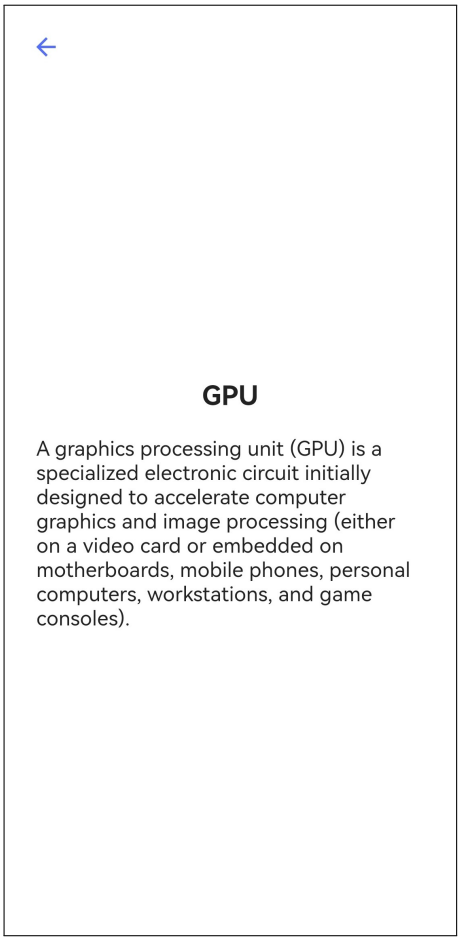


Figura 4.50: Informazioni Modello 3D

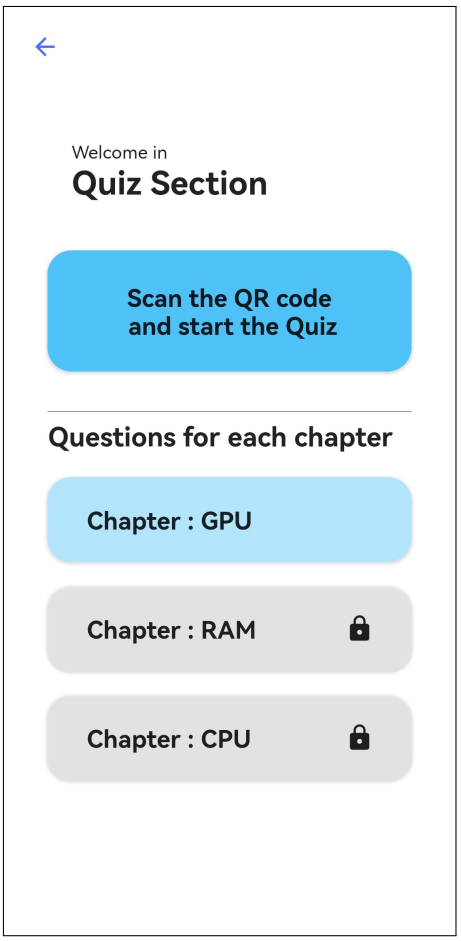


Figura 4.51: Sezione dei Quiz (domande)

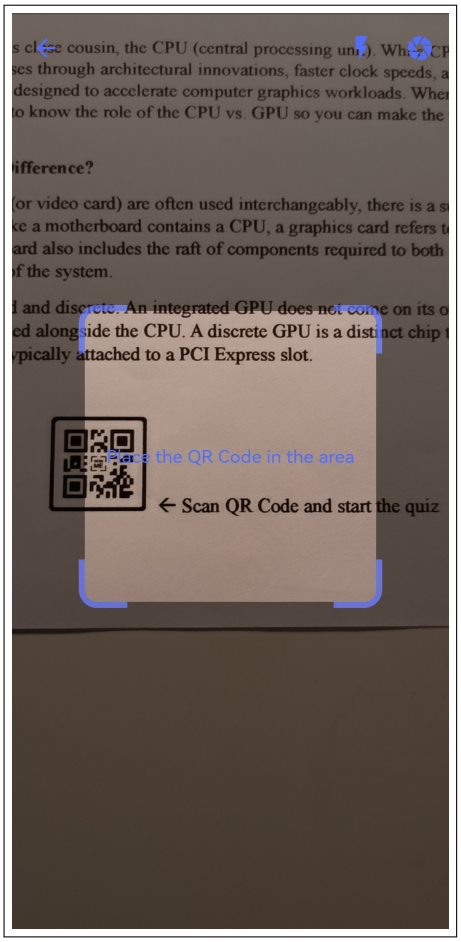


Figura 4.52: Avvio quiz attraverso Qr-Code

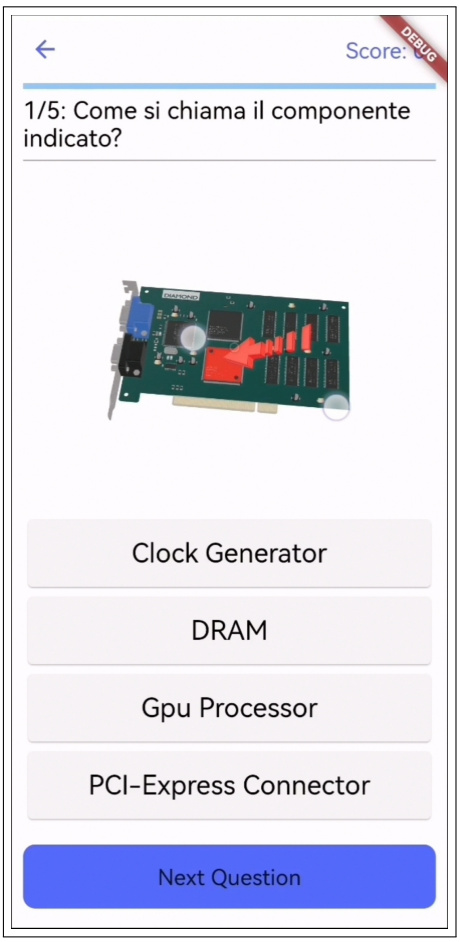


Figura 4.53: Domanda di un quiz

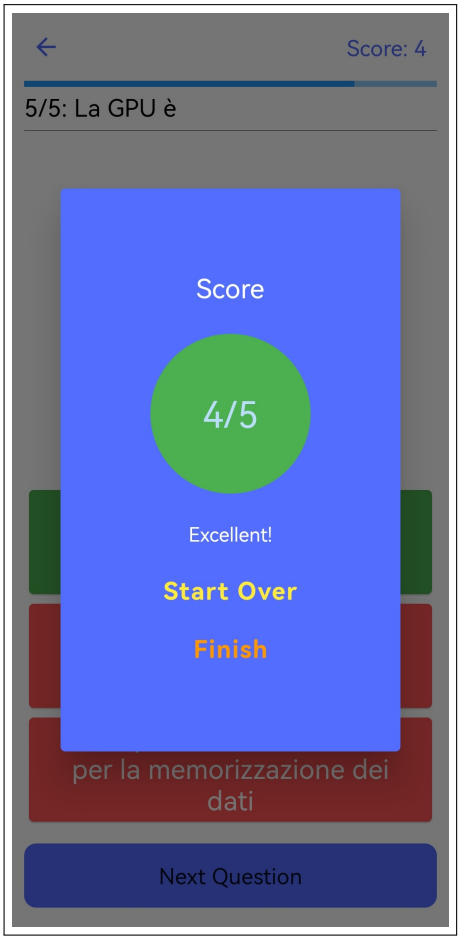


Figura 4.54: Risultato di un quiz

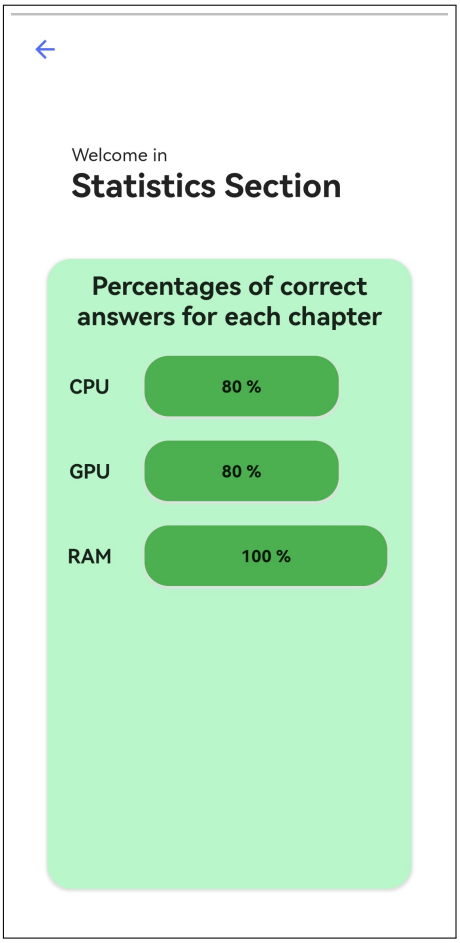


Figura 4.55: Sezione delle Statistiche

Capitolo 5

Discussione e conclusione

5.1 Discussione

L'obiettivo di questa tesi è stato quello di discutere le fasi di analisi, progettazione e realizzazione di un'applicazione mobile che supportasse lo studente nello studio attraverso l'utilizzo della realtà aumentata, esplorando concetti complessi in modo tangibile e coinvolgente.

Nel primo capitolo, nello specifico, è stata presentata una descrizione dell'applicazione mobile ed è stata effettuata un'analisi approfondita dei requisiti funzionali e non funzionali.

Nel secondo capitolo si è passati, invece, alla fase di progettazione, in cui sono stati studiati i vari casi d'uso.

Nel terzo capitolo sono stati elencati i principali strumenti utilizzati.

Infine, nel quarto capitolo è stata illustrata la struttura scelta per il database e l'architettura del progetto scelta per soddisfare i requisiti e poi si è passati allo sviluppo dell'applicazione, parlando in modo più tecnico delle varie classi e metodi implementati.

L'utilizzo di Android Studio, il linguaggio Dart con il framework Flutter e la piattaforma Firebase hanno giocato un ruolo fondamentale, in quanto hanno reso lo sviluppo dell'applicazione un'ottima occasione per accrescere le proprie competenze personali. L'applicazione sviluppata ha sicuramente margini di miglioramento, tuttavia rappresenta un'ottima base di partenza per un'eventuale applicazione di supporto allo studio.

5.1.1 Punti di forza

In questa sezione vediamo tutto ciò che può essere considerato un vantaggio dell'applicazione Learn AR.

Un punto di forza relativo all'app realizzata è, sicuramente, rappresentato dalla modalità di sviluppo dell'applicazione. Infatti, è stato utilizzato il framework Flutter; esso abbatte gli svantaggi delle altre tipologie di sviluppo e, rispetto allo sviluppo nativo, si riducono i costi, i tempi di sviluppo e la complessità di manutenzione dell'applicazione.
Un altro vantaggio è proprio lo scopo per cui è stata realizzata l'applicazione, ovvero quello di aiutare l'apprendimento, consentendo agli studenti di esplorare concetti complessi in modo tangibile e coinvolgente.
L'applicazione è molto semplice da utilizzare e, grazie a ciò, può essere adottata anche da chi ha scarsa dimestichezza con i dispositivi mobili.

5.1.2 Debolezze

Il principale problema dell'applicazione Learn AR è il fatto che l'applicazione ha sempre bisogno che il dispositivo sia connesso a Internet, altrimenti l'utente non può accedere alle funzionalità dell'app.

Altri due problemi sono i tempi di caricamento dei modelli 3D quando si esegue una valutazione su un capitolo e le condizioni d'illuminazione dell'ambiente per il riconoscimento dell'immagine target. Questi due aspetti sono trattati nella Sezione 5.2.

5.1.3 Limitazioni

Per la parte di realtà aumentata è stata usata la versione gratuita chiamata Vuforia Engine Free. Questo piano gratuito ha alcune limitazioni:

- **Numero di modelli:**
  La versione gratuita supporta un numero limitato di modelli target (circa 20). Questo include immagini, oggetti e marker. Se l'applicazione richiede un numero maggiore di target, è necessario passare a un piano superiore.

- **Numero di database:**
  Nel piano gratuito, il numero di database che possono essere creati e gestiti è limitato. I database sono utilizzati per memorizzare i target di immagini.

- **Licenza:**
  Le applicazioni sviluppate con la versione gratuita non possono essere commercializzate. È necessario passare a un piano a pagamento per rimuovere questa restrizione.

Per Unity è stata utilizzato il piano Unity Student che dà accesso alla piattaforma di sviluppo 3D in tempo reale con vantaggi speciali riservati esclusivamente agli studenti verificati. La limitazione più importante è che le applicazioni sviluppate con il piano per studenti non possono essere commercializzate.

5.2 Valutazione sperimentale

Questa sezione si occupa di valutare alcuni aspetti dell'applicazione.
Sono stati valutati i tempi di caricamento dell'applicazione, dato che è un processo cruciale per garantire che l'app offra una buona esperienza utente. Questa valutazione comporta la raccolta, l'analisi e l'interpretazione dei dati relativi ai tempi di caricamento per identificare eventuali colli di bottiglia e migliorare le prestazioni.
Oltre questo aspetto è stato valutato, in maniera più generica, la performance nel riconoscimento delle immagini target in base alle condizioni di illuminazione.

5.2.1 Tempi di caricamento

In questa sottosezione vengono valutati i tempi di caricamento di un modello 3D all'interno di una domanda della sezione quiz. Sono stati eseguiti diversi tipi di test:

- test sotto copertura Wifi con segnale ottimo e costante (circa 60 Mbps).
- test in rete 4G con segnale scarso (tra 1.5 e 2 Mbps).
- test in rete 4G con segnale pessimo (tra 0.20 e 0.30 Mbps).

Per ogni test sono state eseguite 10 rilevazioni, come visibile nella tabella sottostante.

Wifi (60 Mbps)	4G (1.5/2 Mbps)	4G (0.2/0.3 Mbps)
1.29 s	2.73 s	28.63 s
1.26 s	1.95 s	20.94 s
1.16 s	1.98 s	17.77 s
1.42 s	2.89 s	10.82 s
1.15 s	2.15 s	21.76 s
1.65 s	1.92 s	13.42 s
1.35 s	1.78 s	16.57 s
1.28 s	5.21 s	27.51 s
1.57 s	3.25 s	22.30 s
1.68 s	2.96 s	15.48 s

Nel test sotto copertura Wifi risulta una media di 1.38 secondi come tempo di caricamento e uno scarto quadratico medio di 0.193 secondi.
Invece, nel test in rete 4G con segnale scarso (1.5/2 Mbps) risulta una media di 2.65 secondi come tempo di caricamento e uno scarto quadratico medio di 1.03 secondi.
Infine, nel test in rete 4G con segnale pessimo (0.2/0.3 Mbps) risulta una media

di 19.55 secondi come tempo di caricamento e uno scarto quadratico medio di 5.78 secondi.

Da questi risultati si evince che per connessione fino a 1 Mbps si hanno tempi di caricamento non accettabili, nell'ordine delle decine di secondi, e saranno presenti variazioni tra un caricamento e l'altro. Per connessioni intorno ai 2 Mbps si hanno tempi di caricamento medi di circa 2.5 secondi e non sono presenti significative variazioni tra un caricamento e l'altro. Per connessioni intorno ai 60 Mbps si hanno tempi medi di circa 1.4 secondi e risultano assenti variazioni di rilievo tra un caricamento e l'altro.

In conclusione per connessioni sopra a 1.5 Mbps l'applicazione risulta apprezzabile da usare, senza eccessivi tempi di attesa nel caricamento.

5.2.2 Condizioni d'illuminazione

In questa sottosezione vengono valutate, in maniera concisa, le condizioni d'illuminazione dell'ambiente per il riconoscimento dell'immagine target.

Il test è stato svolto in maniera qualitativa, da ciò ne risulta che per il perfetto funzionamento dell'applicazione è necessaria una buona illuminazione proveniente dall'alto. In condizioni di scarsa illuminazione il riconoscimento dell'immagine target non va a buon fine e compromette una buona funzionalità dell'app.

Per questo è consigliato utilizzare l'app sono in presenza di una ottima illuminazione.

5.3 Lezioni Apprese

La lezione appresa più importante è proprio l'utilizzo del framework Flutter per lo sviluppo di un'applicazione mobile.
Durante la realizzazione dell'applicazione sono riuscito a capire come utilizzare al meglio la documentazione del linguaggio di programmazione Dart, ma soprattutto a comprendere eventuali errori grazie all'utilizzo del debugger. Molto importante è il fatto di essere ordinati durante la scrittura del codice perché, in questo modo, è sia più facile ricontrollare eventuali errori sia rendere il codice più comprensibile da parte di programmatori terzi.
Ho appreso, anche, come utilizzare Unity, strumento che non avevo mai utilizzato e, soprattutto, come integrare Unity all'interno di Flutter.

Bibliografia

[1] Wikipedia. *Realtà aumentata*. URL: https://it.wikipedia.org/wiki/Realt%C3%A0_aumentata.

[2] *UML Use Case Diagram Tutorial*. URL: https://www.lucidchart.com/pages/it/uml.

[3] *Android Studio*. URL: https://developer.android.com/studio/.

[4] *Flutter, features overview*. URL: https://flutter.dev/.

[5] *What is flutter?* URL: https://docs.flutter.dev/resources/faq#what-is-flutter.

[6] *Dart, features overview*. URL: https://dart.dev/.

[7] *What is dart programming?* URL: https://www.simplilearn.com/what-is-dart-programming-article.

[8] *Firebase, features overview*. URL: https://firebase.google.com/.

[9] *Google Firebase*. URL: https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase.

[10] *Unity, features overview*. URL: https://unity.com/.

[11] *What is Unity?* URL: https://gamedevacademy.org/what-is-unity/.

[12] *Vuforia Engine, features overview*. URL: https://developer.vuforia.com/home.

[13] *Blender, features overview*. URL: https://www.blender.org/features/.

[14] *What Is Blender (Software)? – Simply Explained*. URL: https://all3dp.com/2/what-is-blender-software-simply-explained/.

[15] *Flutter Unity Widget*. URL: https://pub.dev/packages/flutter_unity_widget.

[16] *Flutter Provider: What is it, what is it for, and how to use it?* URL: https://medium.com/bancolombia-tech/flutter-provider-what-is-it-what-is-it-for-and-how-to-use-it-47d6941860d7.

[17] *Getting Started with Vuforia Engine in Unity*. URL: https://developer.vuforia.com/library/getting-started/getting-started-vuforia-engine-unity.

[18] *Unity, features overview*. URL: https://docs.unity3d.com/2017.2/Documentation/Manual/vuforia-sdk-overview.html.