

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche
Corso di Laurea in Informatica

Creazione di una applicazione mobile per
dispositivi Android e IOS mediante la tecnologia
Flutter e Firebase

Incerti Marco

Tesi di Laurea

Relatore
Il Professore Paolo Burgio

Anno Accademico 2020/2021

AL MIO FUTURO E A TE.

Introduzione

La mia grande passione si affaccia sulla tecnologia e più precisamente sulle applicazioni mobile, complice anche la grande importanza che queste hanno avuto negli ultimi anni nella vita di tutte le persone.

Questa tesi si basa su un progetto nato da una chiacchierata fatta con un amico durante una serata insieme. Abbiamo notato che nell'ultimo periodo c'è stata un grande interesse, soprattutto da parte dei giovani, nell'informarsi su argomenti che vanno oltre il loro percorso di studio universitario o che riguardano temi non trattati a fondo durante le lezioni. Esempio concreto sono alcune pagine Instagram che trattano singoli argomenti di interesse e che hanno ottenuto una crescita esponenziale tale da diventare startup affermate.

L'idea di base è quella di offrire ai giovani un modo per ottenere informazioni e aggiornamenti sugli eventi a sfondo culturale e sociale in tutta Italia, dandogli così la possibilità di approfondire i loro interessi personali partecipando a seminari o manifestazioni, guardando video o ascoltando podcast, oppure seguendo corsi.

L'obbiettivo di questa applicazione è quella di proporre alle persone delle alternative per occupare i tempi morti, rimpiazzando così film, serie tv e altre distrazioni sostituendole con eventi formativi che abbracciano i loro interessi. Magari diminuendo il loro tempo trascorso su Instagram e TikTok, rimpiazzandolo con eventi di cui non conoscerebbero l'esistenza.

Abbiamo poi pensato di aggiungere una sezione dedicata esclusivamente ai corsi, permettendo in questo modo alle persone più intraprendenti di affrontare anche percorsi da aggiungere al proprio curriculum.

Mentre affrontavamo la ricerca per la fattibilità di questa parte abbiamo notato che ci sono numerosi corsi offerti anche da prestigiose università e che il 90% delle persone non conoscono.

In conclusione il nostro obiettivo è quello di raccogliere tutti gli eventi di interesse culturale e sociale su una piattaforma, rendendoli disponibili a tutti, a partire da seminari e corsi, passando per concorsi ed eventi sportivi (come le gare organizzate da google o altri enti per mettere in risalto le competenze dei giovani) e infine gli eventi di svago come le dirette su alla piattaforma streaming Twitch. Inoltre la nostra speranza è quella di riuscire a creare situazioni nelle quali sarà possibile stringere nuove amicizie.

Indice

Studio del mercato	7
Di cosa si tratta	7
La mia indagine di mercato	8
Tecnologie per sviluppo app mobile	9
Applicazioni mobile native	10
App Ibride	11
App cross-platform	11
Linguaggio Dart	12
Gestore dei pacchetti Pub	12
Principali meccanismi	13
Dynamic	13
Future	13
Async e Await	14
Stream	14
Flutter	15
Il framework	16
Widget	16
Librerie Cupertino e Material	17
Composizione di widget	17
Firebase	18
Servizi che offre firebase	18
FirebaseAuthentication	19
CloudFirestore	20
Database Rules	23
Analytics	24
Differenza tra database relazionale e NoSQL	25
Perché usare un database NoSQL	27
Il plugin FlutterFire	28
Progettazione e studio di fattibilità del progetto	30

AWS vs Firebase	33
Autenticazione	40
Homepage	46
Tags	49
Il grafico dei tags di interesse dell'utente	53
Richiesta di pubblicazione di un evento	54
I tipi di eventi	55
TabBar	56
Eventi offline e online	58
Visualizzazione degli eventi nella Listview	59
Sincronizzazione dei dati con il database	60
Pagina dell'evento	64
Test della applicazione	65
Team e traguardi raggiunti	67
Conclusioni	69

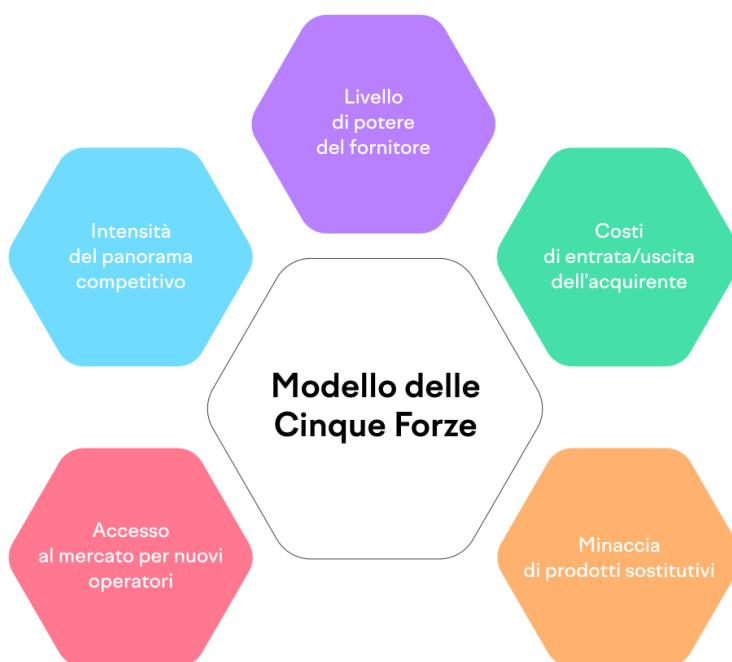
Studio del mercato

Di cosa si tratta

Condurre una ricerca di mercato, oggiorno, si rivela essere semplice grazie ai dati disponibili online. L'analisi di mercato è un modo affidabile e conveniente per raccogliere informazioni dei tuoi mercati di destinazione.

Prima di avviare una startup ricorrere all'analisi di mercato è importantissimo, per comprendere se quello a cui ci si sta rivolgendo sia il giusto pubblico e che tipologia di risposta si potrà avere da quel target. Il metodo migliore utilizzato per svolgere l'analisi di mercato è sicuramente quello che coniuga l'indagine quantitativa, quella qualitativa, strumenti statistici e analitici, e discipline psico-sociologiche. Analizzando il mercato sarà possibile individuare i segmenti di pubblico ed i relativi comportamenti, e allo stesso tempo analizzare anche la concorrenza e più in generale il contesto economico di riferimento per la propria startup. La soluzione ideale sarebbe quella di ricorrere ai macro dati aggregati, recuperati da associazioni di categoria e unirli, per un'analisi più dettagliata, ai risultati dell'indagine specifica realizzata sul proprio fabbisogno.

In questo modo sarà possibile recuperare informazioni utili a definire le strategie di azione per la propria startup, relativamente al mercato, ai trend, al target clienti e al competitor.



La mia indagine di mercato

Inizialmente l'indagine di mercato l'avevo saltata a piedi pari, quando però mi sono confrontato con il professore che mi ha aiutato e seguito per tutta la durata del mio tirocinio, grazie al suo aiuto ne ho compresa l'importanza, soprattutto per potermi confrontare sulla fattibilità del progetto con dati affidabili.

In poche parole il mio studio di mercato non è stato troppo approfondito, ma nemmeno troppo superficiale. Inizialmente ho fatto una ricerca su internet descrivendo la mia idea per controllare che non esistessero altre applicazioni simili. Dalla mia ricerca, svolta non solo in italiano ma anche in lingua inglese, è risultato che non esistono applicazioni o piattaforme simili, ad eccezione di Eventbrite.

Eventbrite è una piattaforma leader nel mercato globale che raccoglie gli eventi di tutte le tipologie e generi in una unica app. Analizzandola ho notato che il suo punto di forza è contemporaneamente la sua più grande debolezza, ovvero la sua vastità , che si rispecchia nella grande quantità e diversa tipologia di eventi presenti, infatti si va dalla lezione di yoga alla mostra al museo.

Invece il mio progetto verte a una selezione di eventi a sfondo culturale e sociale da portare, inizialmente all'attenzione degli studenti universitari e conseguentemente allargare la proposta a tutti coloro che eventualmente possano rivelarsi interessati.

La seconda parte vede come protagonisti i miei amici, circa una trenta giovani, con la quale mi sono confrontato a fondo sulla mia idea facendo nascere preziosi spunti di riflessione per correggere e migliorare i meccanismi della piattaforma e renderla più user-friendly. Tuttavia troppe idee e spunti possono confondere cambiando la vera essenza di base della applicazione e tenendo conto della scarsa quantità di tempo per la realizzazione di questo progetto alcune sono state messe da parte per una aggiunta futura.

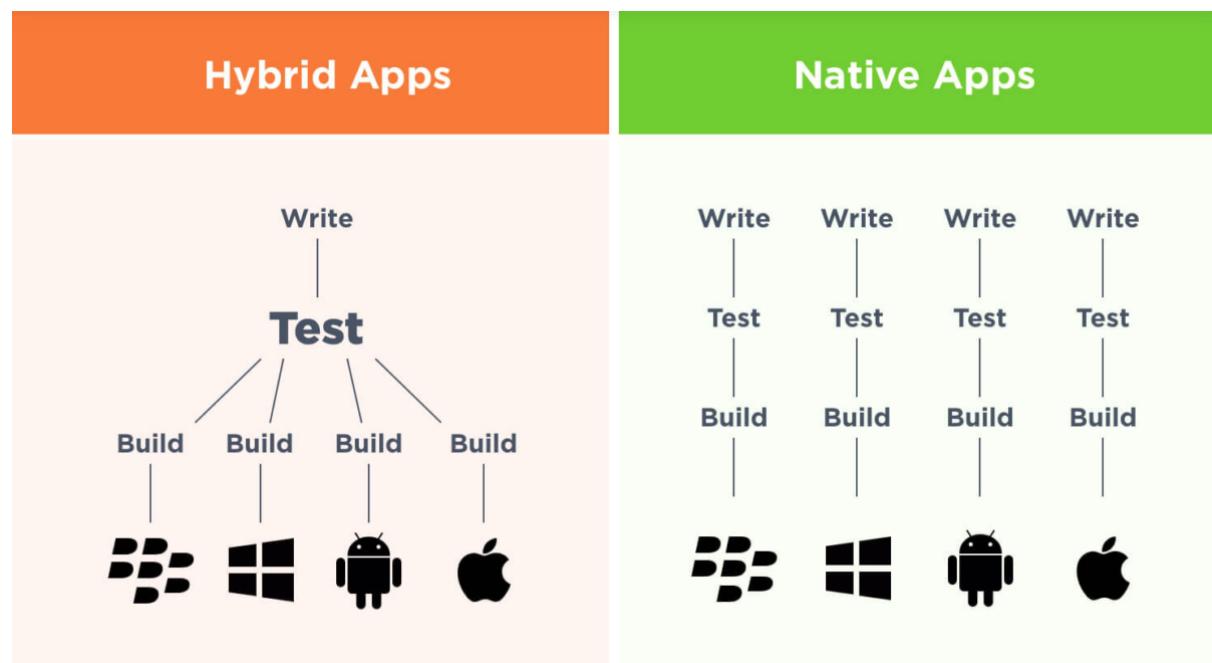
Quindi cosa si evince dalla mia analisi di mercato?

Innanzitutto non ci sono ancora aziende che propongono ai consumatori il prodotto che ho in mente, questo non implica però che si tratti di una buona idea poiché non ci sono molti metri di paragone.

Tecnologie per sviluppo app mobile

Android vs iOS: la sfida eterna della tecnologia che ogni sviluppatore mobile ha affrontato almeno una volta nella vita. La presenza di più linguaggi di programmazione diversi rappresenta un ostacolo allo sviluppo rapido di applicazioni compatibili con entrambi i sistemi operativi, ma con il paradigma cross-platform lo sviluppo combinato per entrambe le piattaforme si semplifica. Questa tesi si concentra su una delle soluzioni più giovani ed interessanti: Flutter, framework open-source di casa Google. Attraverso il linguaggio Dart e l'utilizzo dei widget è possibile creare interfacce adattive per Android e iOS integrando perfettamente le funzioni native e hardware dei vari smartphones, ottenendo così delle applicazioni fluide ed ottimizzate in tempi molto brevi.

Le applicazioni mobile sono software creati appositamente per dispositivi portatili, come smartphone e tablet. La presenza di più sistemi operativi per questi dispositivi ha portato alla necessità di creare diverse modalità di sviluppo per semplificare la creazione di applicazioni per le varie piattaforme.



Applicazioni mobile native

Lo sviluppo nativo è quello che include la maggioranza delle applicazioni mobile esistenti ad oggi, ovvero quelle sviluppate per un sistema operativo specifico. Ogni sistema operativo prevede l'uso di un linguaggio di programmazione dedicato, come Objective-C e Swift per iOS e Java per Android, e fornisce librerie dedicate per utilizzare lo stile standard e tutti i componenti, hardware e non, del dispositivo (GPS, fotocamera, bluetooth, connessione dati e altre) con cui ci si interfaccia direttamente.

Le applicazioni native sono app che si installano e si utilizzano interamente sul dispositivo mobile e sono create appositamente per uno specifico sistema operativo. Tali applicazioni essendo ottimizzate per un determinato sistema garantiscono prestazioni ottimali e sono assolutamente complete, ossia il linguaggio fornisce il pieno controllo sul dispositivo e i suoi sensori. Essendo tali app sviluppate per un sistema operativo specifico, nel caso si voglia trasportarle su altre piattaforme è necessario riscrivere completamente l'applicazione. Le app native comunemente sono realizzate in Java o Kotlin per Android e in Swift per iOS.

In poche parole i vantaggi delle app native sono:

- **Alte prestazioni:** una App Nativa gira molto velocemente nel Sistema Operativo per il quale è stata sviluppata. La perfetta integrazione con il dispositivo nel quale viene eseguita, incide positivamente anche sull'esperienza utente (UX);
- **Funzionalità avanzate:** le App Native hanno accesso a tutte le funzionalità e caratteristiche del dispositivo.

I lati negativi invece sono:

- **Costi di sviluppo elevati:** lo sviluppo di una App Nativa costa molto. Questo è dovuto principalmente al fatto che dovrà sviluppare per la tua azienda almeno due App separate, una per il Sistema Operativo iOS, e l'altra per Android. Spesso questo significa dover ingaggiare due team di sviluppo con competenze diverse e specializzate;
- **Difficoltà nel mantenimento delle versioni:** proprio per il discorso fatto in precedenza, la presenza di più team di sviluppo, o comunque di applicazioni con una base tecnologica diversa, potrebbe causare problemi nell'allineamento delle versioni tra un Sistema Operativo e l'altro.

App Ibride

Le applicazioni ibride sono l'anello evolutivo intermedio tra le app native e le Web App, ovvero sono realizzate mediante linguaggi adibiti allo sviluppo di applicazioni web, ma per funzionare necessitano di un software applicativo che venga scaricato e installato sul device. Questa soluzione garantisce all'utente un'esperienza molto simile alle Progressive Web App, ma intacca l'utilizzo di memoria del dispositivo e di conseguenza le sue performance. Anche questa tipologia fa largamente uso di linguaggi cross-platform come HTML5 e JavaScript.

App cross-platform

Le applicazioni cross-platform, anche dette platform-independent, sono applicazioni a tutti gli effetti, installabili su device, che, come spiega il nome stesso, funzionano su più piattaforme senza il bisogno di riscrivere l'applicazione per adattarla a diversi sistemi operativi, al contrario delle app native. L'utilizzo di tale soluzione abbassa notevolmente i tempi di realizzazione di un'applicazione, garantendo buone performance. Tuttavia alcune funzionalità dei dispositivi mobile sono difficilmente accessibili senza utilizzare codice nativo, come ad esempio l'utilizzo di tutti i sensori, può quindi essere necessaria un'integrazione nativa.

I vantaggi si possono racchiudere in:

- **Costi di sviluppo contenuti:** il più grande vantaggio dello sviluppo di una App Cross-Platform riguarda i costi di realizzazione. La presenza di un unico sviluppatore o team di sviluppo garantisce tempi di realizzazione minori e quindi costi molto contenuti rispetto alle App Native;
- **Supporto e manutenzione:** nelle App Cross-Platform, La presenza di un solo codice sorgente facilita la modifica e l'aggiornamento dell'App. Il team di sviluppo può rilasciare aggiornamenti per più piattaforme in modo simultaneo.

I lati negativi:

- **Esecuzione più lenta rispetto alle App Native:** è giusto sottolineare che la velocità di esecuzione delle App Cross-Platform, per i motivi già discussi, non è ai livelli di quella delle App Native. Tuttavia rispetto al passato sono stati fatti passi da gigante, e molte importanti aziende tra le quali PayPal e Facebook hanno deciso di implementare delle App Cross-Platform per l'erogazione dei loro servizi;
- **Funzionalità limitate:** al contrario delle dirette concorrenti, le applicazioni Cross-Platform non hanno accesso a tutte le API native, questo significa che alcune specifiche funzionalità del Sistema Operativo potrebbero non poter essere sfruttate.

Linguaggio Dart

Dart è un linguaggio di programmazione di proprietà di Google, nato per lo sviluppo di applicazioni Web nel 2011. Dart è un linguaggio di programmazione orientato agli oggetti, per il web, completamente open source e sviluppato da Google: le apps realizzate con Flutter sono scritte con questo linguaggio, utilizzando le features più avanzate. Su Windows, MacOS e Linux attraverso il progetto semi-ufficiale: "Flutter Desktop Embedding Project" girano sulla virtual machine di Dart che contiene un motore di esecuzione in tempo reale. Quando un'app viene sviluppata e debuggata, Flutter usa compilazione in tempo reale, consentendo come detto precedentemente l' "hot reload", le cui modifiche al file sorgente possono essere inserite direttamente all'interno di un'applicazione running. Le versioni rilasciate delle app realizzate con Flutter vengono compilate AOT (ahead of time) sia su Android che su IOS, rendendo così possibili le elevate performance di Flutter sui dispositivi mobile.



Gestore dei pacchetti Pub

Pub è uno strumento per la gestione dei pacchetti per il linguaggio Dart. Si tratta di un vero e proprio raccoglitore di plugin open source che possono essere poi utilizzati nei progetti. Vengono messi a disposizione pacchetti sia per Dart Web che per la parte relativa a Flutter. Qualsiasi applicazione Dart fa largo uso di Pub per l'inclusione di pacchetti all'interno dell'applicazione stessa.

Ci sono intere librerie utilizzabili con una formidabile documentazione sempre aggiornate.

Alcuni pacchetti utilizzati all'interno del mio progetto sono:

- **url_launcher**: usato per aprire pagine safari o applicazioni per inviare email. URL: https://pub.dev/packages/url_launcher. Funziona su: iOS, Android, Web. Questo plugin ti aiuta ad avviare un URL. Gli URL possono essere dei seguenti tipi: HTTP, E-mail, Numeri di telefono, Messaggio di testo SMS;
- **fl_chart**: pacchetto usato per la creazione e la gestione dei grafici.

Principali meccanismi

Dynamic

Dart è un linguaggio fortemente tipizzato ed è dotato di inferenza. Quando però esplicitamente non è utilizzato alcun tipo viene implicitamente utilizzato il tipo dynamic. In Dart ogni cosa è un oggetto che deriva dalla classe Object, un dynamic invece rappresenta un tipo più complesso. L'uso di dynamic può stare a significare che è necessario rappresentare un tipo di dato non rientrante tra quelli consentiti o comunque al di fuori della portata del sistema di tipi statici, o che si desidera esplicitamente il dinamismo a runtime per quel dato. Concretamente significa che il sistema smette di applicare il controllo sul dato dynamic. Se si dichiara un dato di tipo Object, quando si prova a chiamare su quel dato un metodo non definito per il tipo Object, il sistema avvisa che si sta commettendo un errore. Sul tipo dynamic invece è possibile chiamare qualsiasi metodo, anche se questo creerà un errore a runtime, poiché il sistema smette di applicarvi il meccanismo di controllo statico.

Future

Dart possiede un modello a singolo thread basato su event loop. Del codice bloccante eseguito sul thread comporterebbe il blocco dello stesso. È possibile quindi fare uso della programmazione asincrona per eseguire operazioni in modo asincrono. Il tipo Future<T> è un'operazione asincrona che produce un risultato di tipo T. Se il Future non produce risultato, ma è semplicemente necessario effettuare un'operazione asincrona, si utilizza Future<void>.

Quando viene invocata una funzione che restituisce un Future accadono due cose:

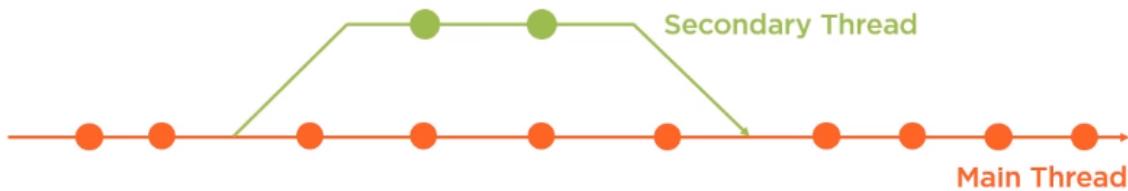
- la funzione accoda le operazioni da eseguire;
- al termine delle operazioni il Future viene completato con il risultato o con un errore.

Come in JavaScript, anche Dart fa uso di callbacks per poter utilizzare il risultato di un Future.

Async e Await

Le parole chiave `async` e `await` sono di supporto alla programmazione asincrona. `Async` viene utilizzata nella signature delle funzioni che restituiscono un `Future` o che ne fanno uso al loro interno, mentre `await` permette letteralmente di "aspettare" che il `Future` possa restituire un risultato. `Await` non è bloccante, semplicemente permette all'event loop di proseguire con gli eventi in coda e quando l'azione long-running è terminata allora l'event loop riprenderà l'esecuzione del `Future` che ha prodotto il risultato, permettendo così di non fare uso delle callbacks.

Asynchronous Programming in Flutter



Stream

Gli Stream in Dart rappresentano, assieme ai `Future`, un'importante astrazione per la programmazione asincrona. Essi sono sequenze di eventi asincroni, che restituiscono un evento quando questo è terminato. Il contenuto di uno Stream può essere elaborato mediante la parola chiave `await` o la callback `listen()`, la quale consente a un Listener di ascoltare lo Stream. Prima di utilizzare questo metodo, lo Stream è un oggetto inerme che contiene al suo interno degli eventi. Nel momento in cui si utilizza il metodo `listen()` viene restituito un oggetto di tipo `StreamSubscription` che è lo stream attivo che produce eventi ed è possibile iniziare ad estrarre gli eventi contenuti al suo interno.

Gli Stream si suddividono in due categorie:

- single subscription stream, in cui gli eventi vengono restituiti tutti e in ordine. È la tipologia utilizzata quando ad esempio si legge un file o si ricevono dati in seguito ad una chiamata Web. Tale stream però può essere ascoltato da un solo Listener un'unica volta, poiché è possibile che un ascolto successivo comporti la perdita degli eventi iniziali;
- broadcast stream, tipologia destinata ai singoli messaggi che possono essere gestiti uno alla volta, come ad esempio gli eventi generati dal mouse. Più Listener possono mettersi in ascolto sul medesimo Stream di questa categoria e ci possono essere ascolti successivi senza questo comporti la perdita di eventi.

Quando viene attivato un evento, i Listener collegati riceveranno istantaneamente l'evento. Se un Listener viene aggiunto a uno Stream Broadcast mentre viene

attivato un evento, quel Listener non riceverà l'evento attualmente in corso di esecuzione mentre se un Listener viene cancellato, interrompe immediatamente la ricezione degli eventi.

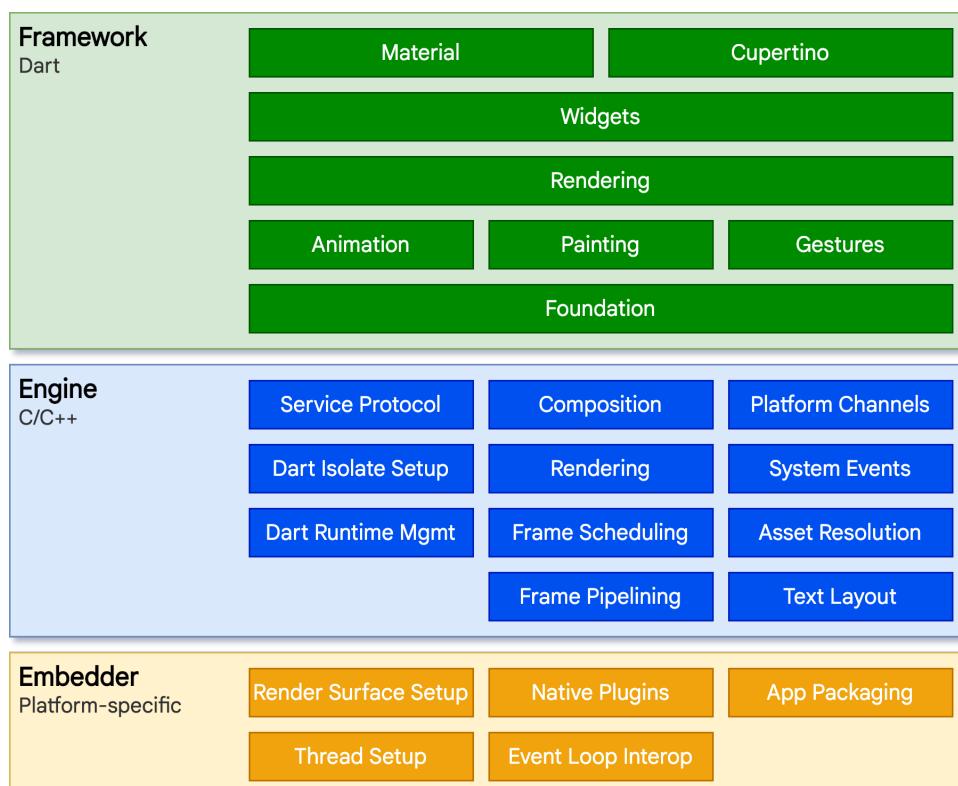
Flutter

Oggi per essere efficace lo stesso programma deve poter funzionare su smartphone Android, iOS e, idealmente, anche su browser.

A questo scopo, in passato, il codice di programmazione doveva essere riadattato e ricompilato per ogni piattaforma. In molti casi bisognava intervenire anche in misura importante per tenere conto, nella propria app, delle particolarità del rispettivo sistema operativo, nonché dell'estetica e della funzionalità delle interfacce utente a cui i clienti erano abituati. Flutter è subentrato a semplificare lo sviluppo multipiattaforma poiché richiede un unico codebase per tutte le piattaforme.

Flutter è un framework per lo sviluppo di app per diverse piattaforme, ideato da Google e pubblicato per la prima volta alla fine del 2018 come progetto open source. Flutter offre una vasta serie di librerie di elementi di interfaccia utente (UI) standard, di Android e iOS, ma è adatto anche per lo sviluppo di applicazioni web o desktop tradizionali.

Le applicazioni sviluppate con Flutter hanno l'aspetto delle app tipiche del sistema a cui sono destinate, comportandosi come queste ultime, senza che il programmatore debba prestare attenzione alle peculiarità del sistema stesso.



Il framework

Flutter si basa su un uso massiccio della composizione, in quanto tutte le interfacce che vengono create sono composte da un alto numero di widget. Il carico di lavoro viene quindi gestito tramite algoritmi sub lineari per costruire i widget e strutture che rendano efficiente la creazione dell'albero in modo da avere un'ottimizzazione costante.

Widget

In Flutter tutto è un widget, ovvero un elemento grafico che può adattarsi dinamicamente in base alle combinazioni ed alle logiche scelte dal programmatore. Per questo motivo, le applicazioni create in Flutter vengono chiamate alberi di widget.

I widget possono essere di due tipi:

- Stateless widget, non contengono informazioni sul loro stato.
- Stateful widget, contengono informazioni sul loro stato.

Stateless widget

I widget di tipo Stateless creano una parte dell'interfaccia grafica e contengono al loro interno altri widget a cascata, che descrivono più nei dettagli la grafica. Vengono utilizzati principalmente per quelle porzioni di interfaccia che dipendono esclusivamente dalle informazioni di configurazione fornite all'oggetto stesso.

Il metodo alla base dei widget è il build(), che permette di definire il contenuto del widget stesso e la sua logica. Viene richiamato ogni volta che una dipendenza del widget cambia, e quindi il suo aspetto deve essere modificato.

Stateful widget

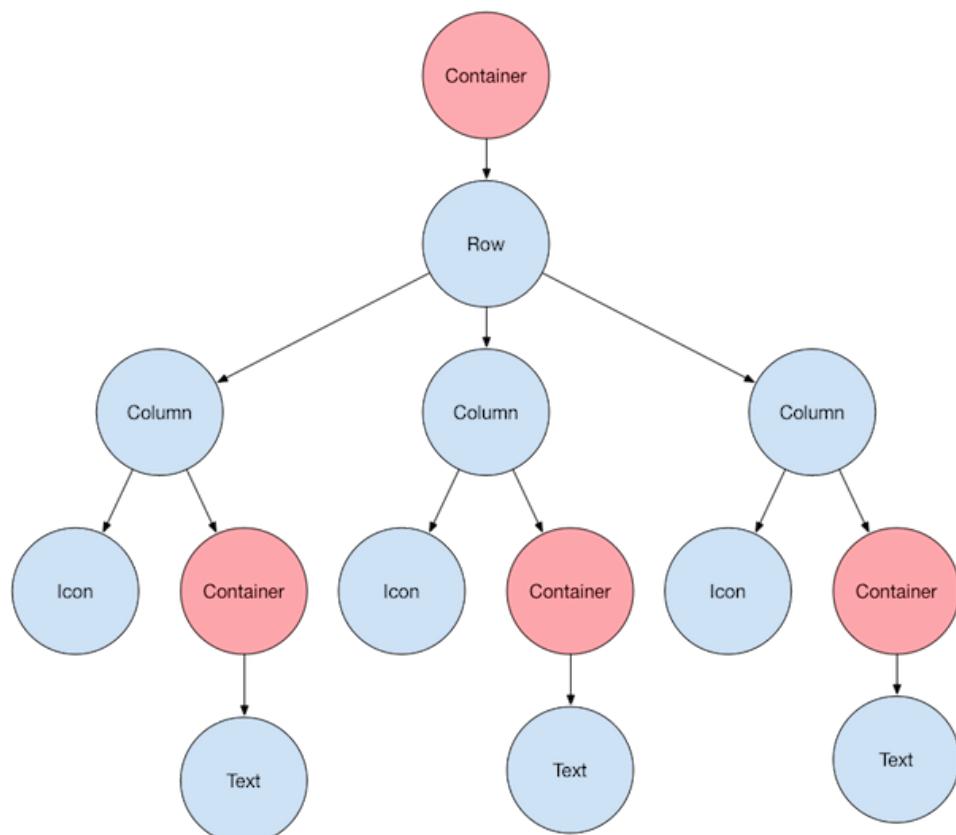
Gli Stateful widget posseggono uno stato mutabile. L'informazione legata allo stato può essere letta in maniera sincrona quando il widget viene creato, tramite la build come per gli stateless widget, ma può anche cambiare durante il ciclo di vita del widget stesso. Per assicurarsi che questa modifica venga visualizzata anche a livello grafico, quest'ultima deve essere segnalata tramite il metodo setState(), fondamentale per i widget di questo tipo. Il metodo contiene la logica e forza Flutter a ricostruire l'albero dei widget secondo le modifiche attuate.

Librerie Cupertino e Material

A partire dal 2014, Google ha adottato un nuovo design chiamato Material, che segue delle regole di progettazione sull'uso di layout a griglia insieme ad animazioni, transizioni ed effetti di profondità basati su luce ed ombre. Questo design è supportato nativamente solo a partire da Android 5.0 ma, attraverso l'utilizzo di Flutter, è possibile ottenerlo anche su dispositivi con versioni precedenti, in quanto i widget implementano la libreria Material che segue lo stile di casa Google. Per mantenere uno stile familiare anche per le applicazioni destinate ad iOS, Flutter include anche la libreria Cupertino, che contiene gli stessi widget della libreria Material ma modificati secondo il design di Apple.

Composizione di widget

La compatibilità aggressiva è uno degli aspetti caratteristici di Flutter. Tutti i widget infatti sono composti da tanti widget single-purpose più piccoli, che vengono combinati tra di loro per ottenere il miglior effetto possibile. Uno degli esempi più comuni è il Container, che permette di posizionare nello spazio, dare forma, colore e stile ad un altro widget. Tutto questo avviene tramite LimitedBox, Padding, DecoratedBox ed altri widget che lo compongono, i quali valori possono essere modificati per ottenere qualsiasi combinazione. A loro volta questi widget sono composti da widget ancora più semplici, a volte ridotti a singole proprietà, che ne descrivono le caratteristiche più specifiche.



Firebase

Firebase è una piattaforma Mobile backend as a service (MBaaS) che consente di interfacciare applicazioni mobili e web app ad un cloud backend, fornendo allo sviluppatore servizi utili per la gestione degli utenti, storage, notifiche push ed altri strumenti di analisi e sviluppo.

Il modello su cui si basa la piattaforma è relativamente recente poiché appoggiandosi al cloud computing, fornisce un servizio globale e uniforme per connettere client differenti offrendo una sincronizzazione dei dati in tempo reale.

Lo sviluppo di Firebase iniziò dall'omonima azienda che nel 2011 sviluppò la piattaforma con l'idea di fornire un servizio in grado di sincronizzare dati in tempo reale, successivamente ricevette grande interesse da parte di Google che nel 2014 acquistò Firebase e altre startup simili, integrandole con i suoi servizi Google Cloud Platform.

Servizi che offre firebase

Firebase offre principalmente un NoSQL realtime database ed una serie di API e librerie client per interagire con lo stesso. In quanto realtime database la grande potenzialità di questo strumento è quella di mantenere aggiornate le informazioni in tempo reale, previa presenza di connettività, in tutti i dispositivi che ospitano l'app connessa al db.

 Cloud Firestore Store and sync app data at global scale	 Crashlytics Prioritize and fix issues with powerful, realtime crash reporting	 In-App Messaging <small>BETA</small> Engage active app users with contextual messages
 ML Kit <small>BETA</small> Machine learning for mobile developers	 Performance Monitoring Gain insight into your app's performance	 Google Analytics Get free and unlimited app analytics
 Cloud Functions Run mobile backend code without managing servers	 Test Lab Test your app on devices hosted by Google	 Predictions Smart user segmentation based on predicted behavior
 Authentication Authenticate users simply and securely	 App Distribution <small>BETA</small> Distribute pre-release versions of your app to your trusted testers	 A/B Testing <small>BETA</small> Optimize your app experience through experimentation
 Hosting Deliver web app assets with speed and security		 Cloud Messaging Send targeted messages and notifications
 Cloud Storage Store and serve files at Google scale		 Remote Config Modify your app without deploying a new version
 Realtime Database Store and sync app data in milliseconds		 Dynamic Links Drive growth by using deep links with attribution

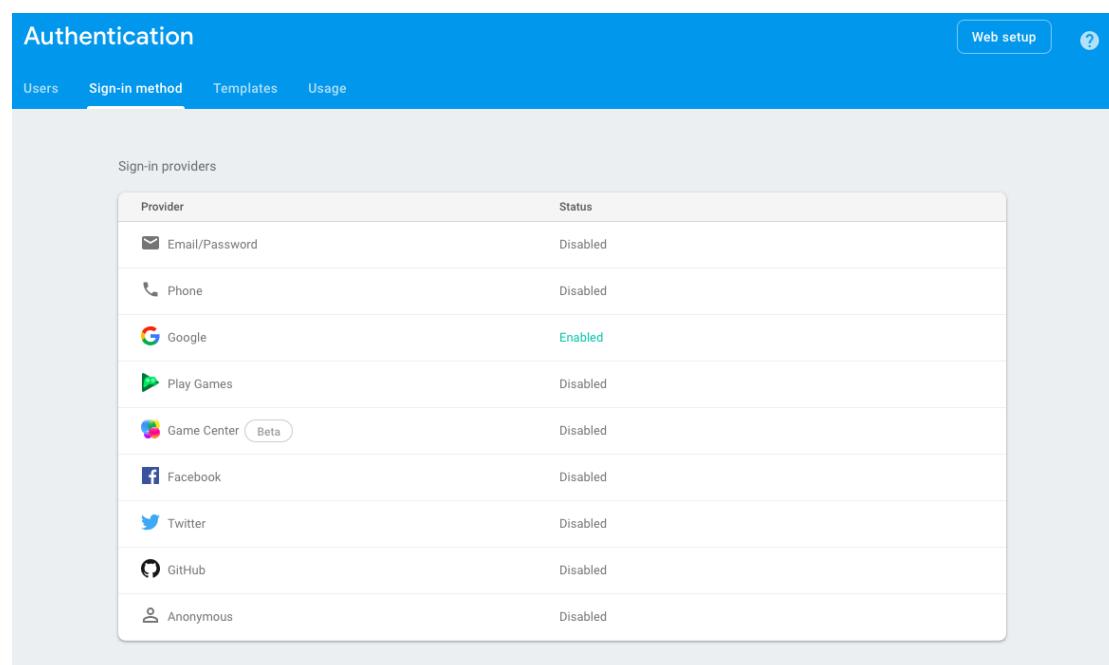
FirebaseAuthentication

Firebase Authentication permette di implementare funzioni di autenticazione degli utenti nell'applicazione attraverso l'uso di passwords, numeri di telefono, o identity providers come Google, Facebook, Twitter e altri. Utilizzando la SDK di Firebase Authentication è possibile integrare manualmente uno o più metodi di sign-in nella propria app.

Nel nostro caso di studio si è scelto di basare l'autenticazione sull'email/password o account google dell'utente.

Per prima cosa quindi verranno inserite le credenziali (in questo caso email e password), che verranno passate alla SDK di Firebase Authentication. I servizi backend di Firebase verificheranno tali credenziali. Dopodiché sarà possibile accedere alle informazioni di base dello user e controllare il suo accesso ai dati immagazzinati in altri strumenti Firebase, come per esempio il Firebase Database. Inoltre attraverso la modifica delle regole di sicurezza di Firebase Realtime Database e Cloud Storage sarà possibile limitare le autorizzazioni a tale utente, che di default ha i permessi di lettura e scrittura.

L'utente in Firebase rappresenta l'account di un utente registrato.



The screenshot shows the Firebase Authentication console interface. At the top, there's a blue header bar with the title "Authentication". Below the header, there are four tabs: "Users", "Sign-in method" (which is currently selected), "Templates", and "Usage". On the right side of the header, there are two buttons: "Web setup" and a help icon. The main content area is titled "Sign-in providers" and contains a table with the following data:

Provider	Status
Email/Password	Disabled
Phone	Disabled
Google	Enabled
Play Games	Disabled
Game Center	Beta
Facebook	Disabled
Twitter	Disabled
GitHub	Disabled
Anonymous	Disabled

Ogni account presenta le seguenti caratteristiche: ID, un indirizzo email primario, un nome e un photo URL. Altre proprietà non possono essere aggiunte direttamente al set, è però possibile memorizzarle in altri servizi di storage di Firebase (come nel nostro caso il Firebase Realtime Database). Per esempio, la prima volta che l'utente si registra all'applicazione, i dati del profilo dell'utente vengono popolati con le

informazioni disponibili: nel nostro caso l'utente si è registrato con email e password, e solamente il campo indirizzo email verrà popolato. In ogni caso, ogni volta che verranno modificati i dati dell'utente, le sue informazioni verranno aggiornate automaticamente.

Nel momento in cui un utente si registra o effettua il login, diventa il “current user” dell’istanza Auth. L’istanza mantiene lo stato dell’utente in modo tale da non perdere le informazioni dell’utente ad ogni refresh di pagina o restart dell’applicazione.

Quando l’utente effettua il logout, l’istanza Auth cessa di mantenere un riferimento all’oggetto user, e non c’è nessun current user.

CloudFirestore

Cloud Firestore è un database flessibile e scalabile per lo sviluppo di dispositivi mobili, web e server da Firebase e Google Cloud. Come Firebase Realtime Database, mantiene i tuoi dati sincronizzati tra le app client tramite listener in tempo reale e offre supporto offline per dispositivi mobili e Web in modo da poter creare app reattive che funzionano indipendentemente dalla latenza di rete o dalla connettività Internet. Cloud Firestore offre anche una perfetta integrazione con altri prodotti Firebase e Google Cloud, incluse Cloud Functions.

The screenshot shows the Cloud Firestore console. On the left, there's a sidebar with a 'heroes' collection icon. The main area displays a document named 'K54K1vAAQrg001CUSn'. This document has a 'weapons' field, which is expanded to show three sub-document IDs: '-Lf30U0eEJE63825VN1j', '-Lf30V1EH4mPJCW04WGR', and '-Lf30V9xW8MeQhI13TQN'. Below the 'weapons' field, there are buttons for '+ Add field' and two fields with values: 'name: "Birdman"' and 'strength: 100'.

Cloud Firestore è un database NoSQL ospitato nel cloud a cui le tue app iOS, Android e Web possono accedere direttamente tramite SDK nativi. Cloud Firestore è disponibile anche negli SDK nativi Node.js, Java, Python, Unity, C++ e Go, oltre alle API REST e RPC.

Seguendo il modello di dati NoSQL di Cloud Firestore, archivi i dati in documenti che contengono campi mappati a valori. Questi documenti sono archiviati in raccolte, che sono contenitori per i tuoi documenti che puoi utilizzare per organizzare i dati e creare query. I documenti supportano molti tipi di dati diversi, da semplici stringhe e numeri a oggetti complessi e nidificati. Puoi anche creare

sottoraccolte all'interno dei documenti e creare strutture di dati gerarchiche che si ridimensionano man mano che il database cresce. Il modello di dati Cloud Firestore supporta qualsiasi struttura di dati che funziona meglio per la tua app.

Inoltre, l'esecuzione di query in Cloud Firestore è espressiva, efficiente e flessibile. Crea query superficiali per recuperare i dati a livello di documento senza dover recuperare l'intera raccolta o eventuali sottoraccolte nidificate. Aggiungi ordinamento, filtri e limiti alle tue query o ai cursori per impaginare i risultati. Per mantenere aggiornati i dati nelle tue app, senza dover recuperare l'intero database ogni volta che si verifica un aggiornamento, aggiungi listener in tempo reale. L'aggiunta di listener in tempo reale alla tua app ti avvisa con uno snapshot dei dati ogni volta che i dati che le tue app client stanno ascoltando cambiano, recuperando solo le nuove modifiche.

Proteggi l'accesso ai tuoi dati in Cloud Firestore con l'autenticazione Firebase e le regole di sicurezza Cloud Firestore per Android, iOS e JavaScript o Identity and Access Management (IAM) per le lingue lato server.

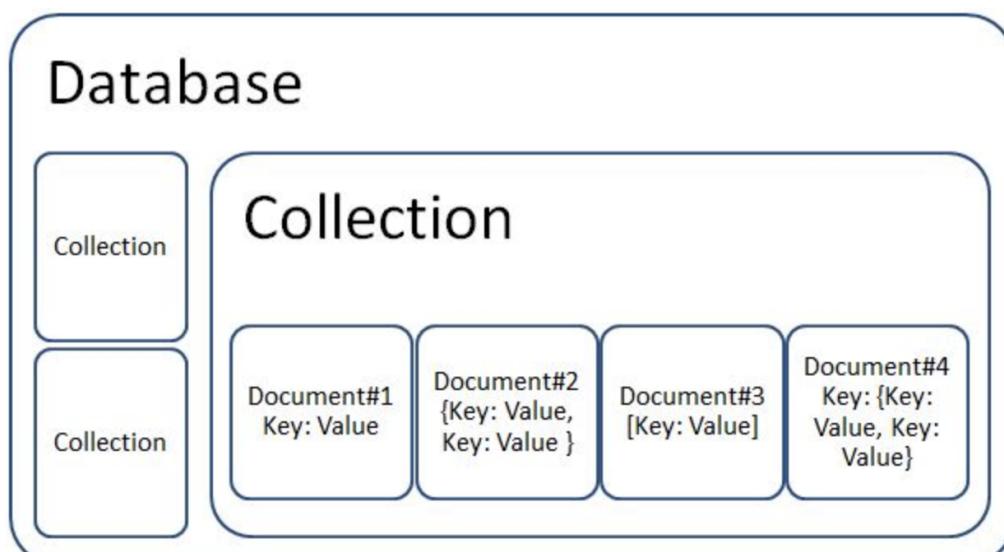
Il modello di memorizzazione dei dati è basato sui documenti che possono contenere stringhe e numeri, date, oggetti complessi e annidati.

Questi documenti sono archiviati in raccolte, chiamate collezioni che contengono i documenti, ma è anche possibile creare sub-collezioni all'interno dei documenti e creare strutture gerarchiche di dati che scalano man mano che il database cresce.

Gli unici limiti imposti da Firestore sono: la dimensione di un singolo documento che è di 1 MiB (1,048,576 bytes) e un massimo di 100 collezioni annidate.

L'SDK del database Firestore mantiene i dati aggiornati attraverso una buona gestione del caching, i client invece utilizzando appositi listener offrono una sincronizzazione dei dati in tempo reale.

L'aggiunta di listener oltre a informare il client su modifiche effettuate nel database, permette di memorizzare le richieste effettuate in precedenza e mantenere una copia delle risposte del server nella cache, offrendo quindi un supporto offline dei dati.



Documenti

Un documento è identificato da un nome univoco e rappresenta un record contenente una quantità variabile di chiavi associate a valori in una struttura JSON. Tutti i documenti possono contenere una quantità indefinita di Collezioni, permettendo in questo modo la creazione di creare un vero e proprio albero dei dati. Recuperare un documento ne carica solamente i dati diretti, lasciando dunque ad una query puntuale il compito di recuperare i dati contenuti in eventuali sotto collezioni del documento.

Collezioni

Le collezioni sono liste di Documenti ottimizzate per l'ordinamento, il recupero e la gestione di lunghe liste di documenti, ciò significa che per garantire una buona scalabilità dell'applicazione è necessario tenere bene a mente che è sempre preferibile avere collezioni ampie invece di grandi documenti.

Reference

Per accedere a documenti e collezioni è necessario creare Reference al quale poi vengono applicate le funzioni di lettura/scrittura che variano in base alla tipologia di oggetto referenziato, ad esempio è possibile ordinare e filtrare la lista dei documenti contenuti all'interno di una collezione da una CollectionReference mentre è possibile aggiornare il contenuto di uno specifico documento solo da una DocumentReference.

Ogni reference può essere concatenata seguendo un'unica semplice regola: Una CollectionReference può essere susseguita unicamente da una DocumentReference e viceversa, ciò significa che le due tipologie di reference devono necessariamente susseguirsi.

Operazioni di lettura

I dati referenziati da una reference possono essere letti staticamente, creando un singolo snapshot, oppure ascoltati specificando una callback da chiamare ogni volta che viene rilevato un cambiamento dei dati.

Il metodo get() permette di leggere direttamente creando uno snapshot del dato, onSnapshot() è utilizzato invece per creare un listener che chiamerà la callback passata come argomento.

Le CollectionReference espongono metodi che permettono di filtrare la lista dei documenti, tali filtri vengono applicati tramite i diversi metodi descritti in tabella e possono essere ordinati tramite il metodo orderBy().

Operazioni di scrittura

Tutti i metodi di scrittura permettono di aggiornare, modificare, creare e cancellare i dati referenziati, tutte le modifiche vengono riflettute in realtime sul database innescando l'aggiornamento di tutte le risorse aggiornate da tutti i client.

Snapshot

I dati ritornati da un'operazione di lettura vengono identificati come snapshot, ovvero un'immagine dei dati recuperati in quell'istante. Esistono snapshot relativi sia a collezioni che documenti, la sua gestione in entrambi i casi richiede un'attenzione particolare.

DocumentSnapshot

Questo tipo di snapshot contiene il riferimento ad uno specifico documento, da questo è possibile recuperare tutti i dati contenuti all'interno del documento (`.data()`) o recuperare uno specifico `fieldPath`.

QuerySnapshot

Uno snapshot di questo tipo permette di gestire un riferimento ad una collezione, da questa è possibile effettuare tutte le operazioni di lettura sui dati, come l'array dei documenti, i metadati, le modifiche dall'ultimo snapshot, un iteratore sui documenti e tutto ciò che è descritto nella tabella.

Database Rules

Firebase offre per i suoi due database la possibilità di inserire delle restrizioni e regole di sicurezza per l'accesso al Database, chiamate Database Rules. Le Database Rules determinano chi ha accesso in lettura e scrittura al database o a collezioni di dati all'interno del database, queste regole sono gestite utilizzando il pannello di controllo di Firebase, una volta scritte le regole queste vengono applicate automaticamente ad ogni modifica. Ogni richiesta di lettura e scrittura di dati nel database sarà completata solo se le regole lo consentono.

Entrambi i database: Real time e Firestore supportano le Database Rules, le differenze fra i due servizi riguardano il metodo con cui vengono scritte le Introduzione regole e il tipo di controlli che è possibile effettuare.

Firebase permette di scrivere le regole utilizzando un file in formato JSON dove vengono definite le regole in base alla collezione in cui si trovano i dati, alla validazione dei dati o in base all'utente registrato su Firebase Auth.

The screenshot shows the Firebase Rules Editor interface. At the top, there are buttons for "Modifica regole" (Edit rules), "Monitora regole" (Monitor rules), and "Sviluppa e testa" (Develop and test). On the left, a sidebar lists three rule versions with timestamps: "mar 18, 2021 · 2:30 PM", "mar 18, 2021 · 12:13 PM", and "mag 22, 2020 · 6:35 PM". The main area displays the current rule code:

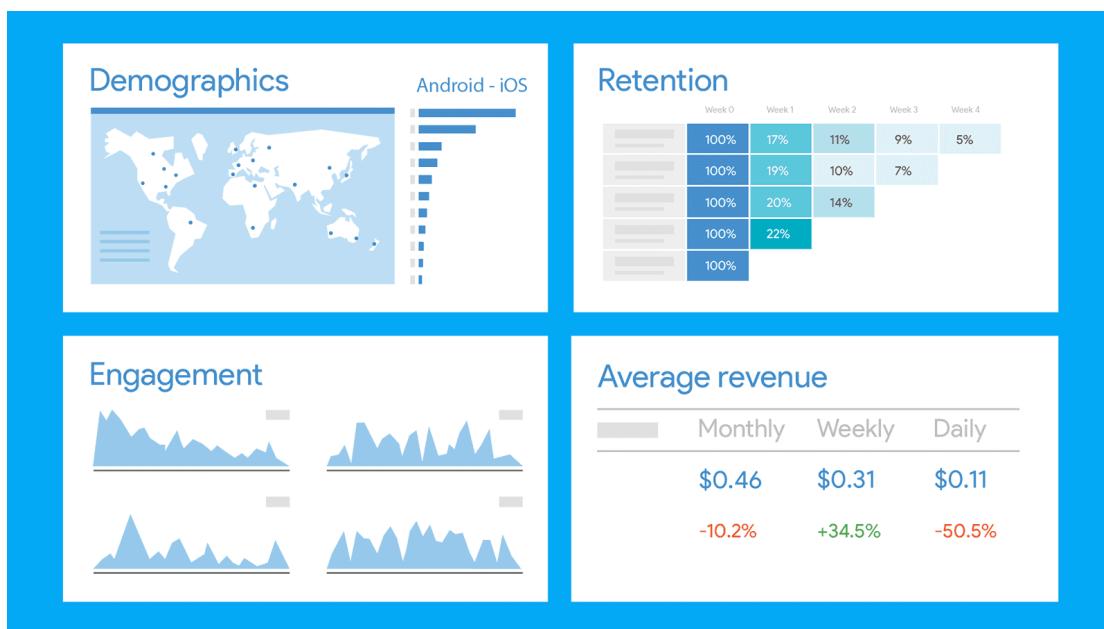
```

1  rules_version = '2';
2  service cloud.firestore {
3      match /databases/{database}/documents {
4          match /{document=**} {
5              allow read, write: if request.auth != null;
6          }
7      }
8  }

```

Analytics

Analytics si integra con le funzionalità di Firebase e ti fornisce rapporti illimitati per un massimo di 500 eventi distinti che puoi definire utilizzando l'SDK di Firebase. I rapporti di analisi ti aiutano a capire chiaramente come si comportano i tuoi utenti, il che ti consente di prendere decisioni informate in merito al marketing delle app e all'ottimizzazione delle prestazioni.



Google Analytics ti aiuta a capire come le persone utilizzano la tua app web, iOS o Android. L'SDK acquisisce automaticamente una serie di eventi e proprietà utente e ti consente anche di definire i tuoi eventi personalizzati per misurare le cose che contano in modo univoco per la tua attività. Una volta acquisiti, i dati sono disponibili in una dashboard tramite la console Firebase. Questa dashboard fornisce informazioni dettagliate sui tuoi dati, da dati di riepilogo come utenti attivi e dati demografici a dati più dettagliati come l'identificazione degli articoli più acquistati.

Analytics si integra anche con una serie di altre funzionalità di Firebase. Ad esempio, registra automaticamente gli eventi che corrispondono ai messaggi di notifica inviati tramite il compositore di notifiche e fornisce report sull'impatto di ciascuna campagna.

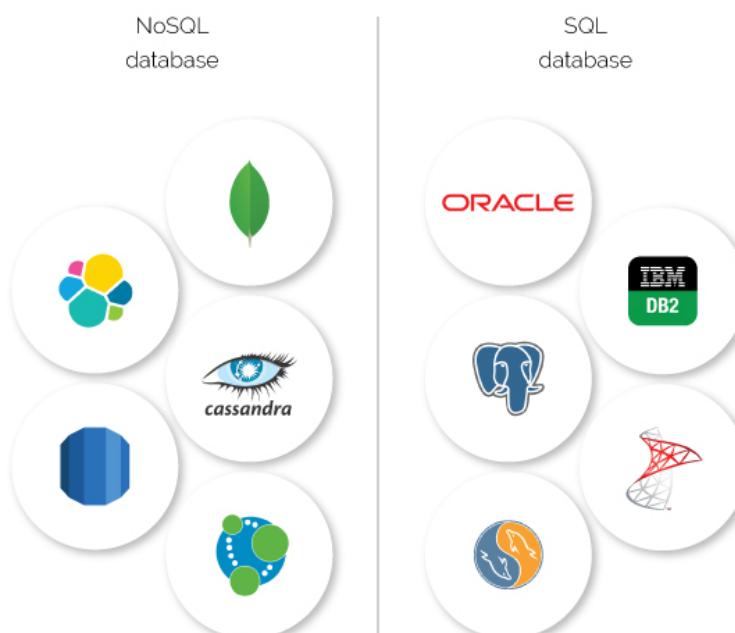
Analytics ti aiuta a capire come si comportano i tuoi utenti, così puoi prendere decisioni informate su come commercializzare la tua app. Guarda il rendimento delle tue campagne sui canali organici e a pagamento per capire quali metodi sono più efficaci per attirare utenti di alto valore. Se devi eseguire analisi personalizzate o unire i tuoi dati ad altre origini, puoi collegare i tuoi dati di Analytics a BigQuery, che consente analisi più complesse come l'esecuzione di query su grandi set di dati e l'unione di più origini dati.

Differenza tra database relazionale e NoSQL

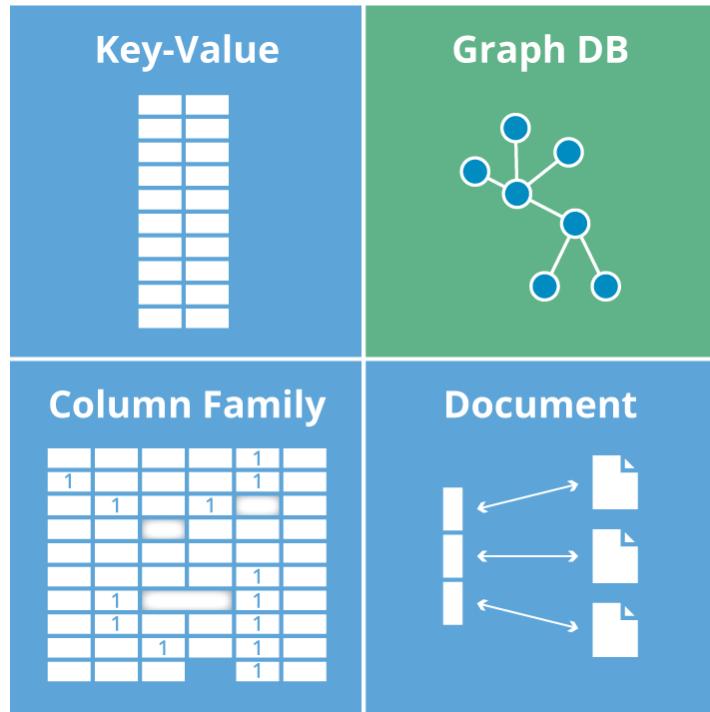
La strutturazione rigida dei contenuti, tipica dei database relazionali, è un elemento assente nei database NoSQL, e proprio tale assenza è uno degli aspetti che maggiormente ne hanno permesso il successo.

Le informazioni non sono più memorizzate in tabelle, ma in oggetti completamente diversi e non necessariamente strutturati, come ad esempio documenti archiviati in collezioni.

Un oggetto JSON ad esempio, come vedremo in seguito, rappresenta un documento da inserire nel database.



La chiave primaria del documento è un campo denominato '_id' che, se non viene fornito in fase di inserimento, verrà aggiunto automaticamente dal DBMS.



Nei DBMS NoSQL a documenti, è significativa l'assenza delle relazioni. I meccanismi con cui vengono collegate le informazioni sono infatti due:

- **Embedding:** significa annidare un oggetto JSON all'interno di un altro. Questa tecnica sostituisce molto spesso le relazioni 1-a-1 e 1-a-molti. È tuttavia sconsigliabile utilizzarla quando i documenti (quello annidato e quello che lo contiene) crescono di dimensione in maniera sproporzionata tra loro, oppure se la frequenza di accesso ad uno dei due è molto minore di quella dell'altro;
- **Referencing:** somiglia molto alle relazioni dei RDBMS, e consiste nel fare in modo che un documento contenga, tra i suoi dati, l'id di un altro documento. Molto utile per realizzare strutture complesse, relazioni molti-a-molti.

Perché usare un database NoSQL

Limiti dei database relazionali:

Sono state effettuate ricerche per valutare i pro e i contro di un database SQL/NoSQL ed è emerso, come già detto sopra, che non sempre i DBMS tradizionali si dimostrano in grado di gestire quantità di dati molto grandi, se non al prezzo di performance molto più limitate e costi piuttosto elevati.

Qui di seguito sono stati raccolti alcuni punti di debolezza ravvisati nell'uso dei DBMS relazionali:

- I Join: nonostante la loro efficacia, queste operazioni coinvolgono spesso più righe del necessario (soprattutto se le query non sono ottimizzate), limitando le performance delle interrogazioni eseguite;
- Struttura rigida delle tabelle, che si rivela utile finché si ha necessità di introdurre informazioni caratterizzate sempre dalle medesime proprietà, ma molto meno efficiente per informazioni di natura eterogenea;
- Conflitto di impedenza, che consiste nella differenza strutturale tra i record delle tabelle e gli oggetti comunemente utilizzati per gestire i dati nei software che interagiscono con le basi di dati. Questo problema ha prodotto – tra l'altro – la nascita di strumenti come gli O/RM, librerie in grado di convertire oggetti in record (e viceversa);
- Proprietà ACID: ogni transazione deve essere atomica, consistente, isolata e duratura. Implementare queste proprietà, però, comporta una serie di controlli e precauzioni che penalizzano le prestazioni del sistema. I database non relazionali sono meno stringenti, offrendo una maggiore elasticità.

Quando il NoSQL conviene:

Sempre dopo aver effettuato svariate ricerche, sono stati raccolte informazioni sul perché sia conveniente utilizzare un database NoSQL piuttosto che relazionale. In generale, conviene scegliere l'approccio NoSQL quando:

- La struttura dei dati non è definibile a priori. Pensiamo ad esempio alla gestione dei contenuti che impongono oggi i social network o gli applicativi web per la gestione dei contenuti;
- I dati disposti nei vari oggetti sono molto collegati tra loro e, situazione aggravata da una loro gran quantità, il Join rischia di non essere lo strumento ottimale: sarebbe da prediligere la navigazione tra oggetti sfruttando i riferimenti tra i vari nodi di informazione;
- È necessario interagire molto frequentemente con il database, volendo evitare conversioni onerose tra record e oggetti, né tanto meno ricorrere all'utilizzo di O/RM o altre librerie specifiche;
- Servono prestazioni più elevate, potendo considerare troppo stringenti le proprietà ACID per certi campi di applicazione.

Il plugin FlutterFire

FlutterFire è un insieme di plug-in Flutter che consentono alle app Flutter di utilizzare i servizi Firebase. Infatti Flutter supporta l'utilizzo di pacchetti condivisi forniti da altri sviluppatori agli ecosistemi Flutter e Dart. Ciò consente di creare rapidamente un'app senza dover sviluppare tutto da zero.

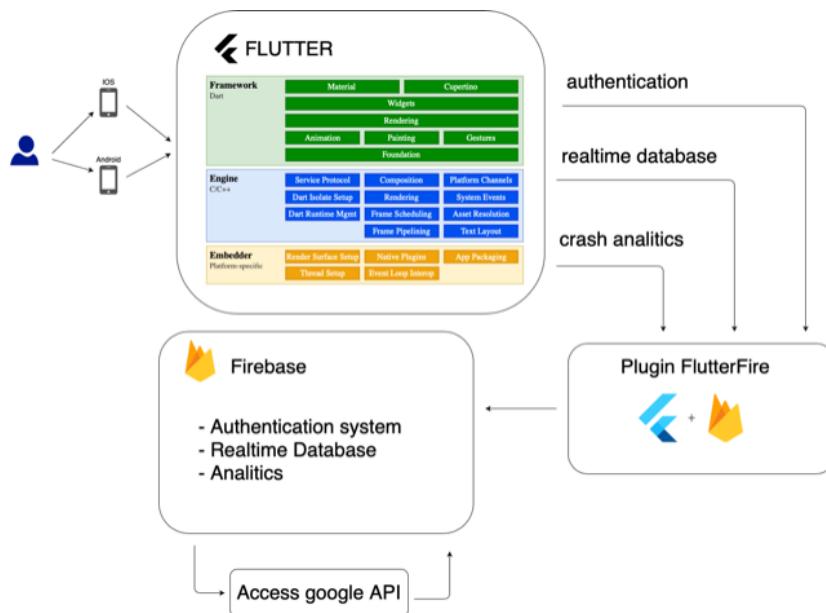
Qual è la differenza tra un pacchetto e un plugin? Un plug-in è un tipo di pacchetto: la designazione completa è pacchetto plug-in, generalmente abbreviato in plug-in.

Packages

Come minimo, un pacchetto Dart è una directory contenente un file pubspec. Inoltre, un pacchetto può contenere dipendenze (elencate nella pubspec), librerie Dart, app, risorse, test, immagini ed esempi. Il sito pub.dev elenca molti pacchetti, sviluppati da ingegneri di Google e generosi membri della community di Flutter e Dart, che puoi utilizzare nella tua app.

Plugin

Un pacchetto plug-in è un tipo speciale di pacchetto che rende disponibili le funzionalità della piattaforma all'app. I pacchetti plug-in possono essere scritti per Android (usando Kotlin o Java), iOS (usando Swift o Objective-C), web, macOS, Windows, Linux o qualsiasi combinazione di questi. Ad esempio, un plug-in potrebbe fornire alle app Flutter la possibilità di utilizzare la fotocamera di un dispositivo.



Il plugin Firebase ci permette di far comunicare la nostra applicazione con firebase mediante semplici e intuitivi comandi. Oltre all'importanza che ha nel funzionamento, un grande altro punto a favore è la grande quantità di documentazione ed esempi che gli sviluppatori e gli utenti mettono a disposizione.

Progettazione e studio di fattibilità del progetto

Per prima cosa ho contattato il professore Paolo Burgio, mio referente durante il tirocinio, per proporgli e spiegargli la mia idea e chiedergli consigli utili per la realizzazione del mio progetto. Dopo una lunga telefonata avevo molti spunti su cui lavorare.

La prima difficoltà che ho incontrato è stata l'analisi del mercato, non avevo pensato di dovervi dedicare così tanto tempo e renderla molto approfondita. Mi sono messo al computer e ho iniziato a cercare se esistessero già applicazioni simili, che dessero allo studente una panoramica sulle attività legate alla vita universitaria e al cittadino comune un'idea generale sulle proposte culturali e sociali della città. Non ho trovato nulla.

Poi mi sono lanciato sulla quantità di eventi disponibili in rete, da poter condividere sulla mia app. Dopo averne trovati tantissimi mi sono accorto di un grande problema, non avevano dei tags che mi permettessero di assegnargli ad una categoria. Questo è uno dei problemi che tratterò in seguito.

Il terzo step invece è stata la ricerca e lo studio di tecnologie che potessero aiutarmi e formare la mia piattaforma. Per questa ricerca ho impiegato molte settimane, sono partito da una applicazione Django con un componente REST per poter condividere gli eventi anche con le applicazioni mobile. Ho studiato a fondo questa tecnologia perché mi sembrava la più fattibile, a seguito di un software web e di un database relazionale integrato del framework.

Dunque ho iniziato a guardare le varie tipologie di tecnologia presenti per capire e studiare quale potesse essere la più utile e performante per la mia necessità. Per questo studio ho impiegato un mese poiché la gamma disponibile al giorno d'oggi è davvero ampia e caratterizzata da tante piccole differenze. Ho iniziato a progettare dall'inizio una applicazione web in Django, ma mi sono accorto ben presto che non era la via giusta. Mi sono poi lanciato nel mondo delle applicazioni, dove ho trovato due possibilità: React Native e Flutter.

REACT NATIVE

È un progetto creato e implementato da Facebook, sinonimo di robustezza e grande avanguardia tecnologica.

Per dirla in modo semplice, questa tecnologia offre una funzione di "traduzione" che esegue il codice React Native in Objective-C per iOS e in Java per Android. L'esperienza dell'utente è quindi identica a quella che si ottiene con una app nativa. In effetti, React Native non è un webview: tutto ciò che l'utente esegue sul proprio dispositivo è assolutamente nativo.

La comunità di sviluppo di questa piattaforma è in costante crescita e i programmati di Facebook continuano ad aggiungere funzionalità native per ampliarne le potenzialità. Un ulteriore vantaggio di React Native è la possibilità, se si rendesse necessario (per introdurre una funzione o un servizio di terzi), di integrare nei progetti un codice nativo: scegliere React Native non significa quindi rinunciare al nativo.

React è un framework open-source JavaScript sviluppato e mantenuto da Facebook, che consente la creazione di interfacce utente suddividendo il sistema in componenti, ognuno con un proprio stato, proprietà e ciclo di vita.

Un componente dispone di un costruttore tramite il quale è possibile inizializzare le variabili di stato, disporre delle proprietà passate da un componente padre e aggiungere della logica da eseguire alla creazione.

E' possibile annidare i componenti uno dentro l'altro e passare delle variabili proprietà contenenti dei valori ai figli. Tramite questo meccanismo è possibile creare dei componenti multifunzione che andranno ad utilizzare le proprietà, con valori diversi in base alla situazione e quindi in base al componente padre, per implementare la propria logica in modo opportuno.

React utilizza un'astrazione del HTML DOM, chiamata Virtual-DOM, che consente di strutturare i contenuti di una pagina in una struttura ad albero, la quale viene aggiornata solamente nelle parti in cui è veramente cambiato qualcosa, andando quindi ad ottimizzare i tempi di elaborazione rispetto al DOM tradizionale.

React Native è un framework open-source JavaScript sviluppato e mantenuto da Facebook, che consente la creazione di applicazioni mobile multipiattaforma (Android e iOS) scritte in JavaScript.

A differenza di altri framework per la creazione di applicazioni web in ambiente mobile React Native non utilizza la WebView, ovvero uno strumento basato sul motore Webkit che permette di renderizzare correttamente pagine web formate da JavaScript, HTML e CSS. Infatti l'aspetto negativo di questo strumento è che ha bisogno di molte risorse, quindi le applicazioni risultano meno performanti rispetto a quelle sviluppate con una UI (User Interface) nativa. React Native mappa ogni suo componente attraverso l'interfaccia utente nativa della piattaforma iOS o Android. Di conseguenza è sempre presente dell'overhead per l'interpretazione del linguaggio JavaScript in quello nativo della piattaforma (Java per Android e Swift per iOS), ma le prestazioni sono migliori degli altri framework.

React Native risulta essere anche molto versatile, in quanto permette l'utilizzo del linguaggio nativo qualora si voglia sfruttare a pieno le potenzialità della piattaforma per cui si sta sviluppando.

FLUTTER

Flutter lo abbiamo già spiegato sopra, ma un ripasso non ci fa mai male. Si tratta di un framework di Google nato nel 2018 come beta e poi sviluppato come software stabile a partire dal 2019. Già oggi viene utilizzato da tantissime piattaforme tra cui AdSense, che all'interno del mondo Google è quella che porta più introiti a conferma di quanto l'azienda americana ci punti. La grande forza di Flutter è quella di poter creare delle applicazioni per Android ed per iOS, andando a scrivere una sola volta il codice sorgente, e con un livello di qualità assimilabile a quello del nativo puro. Riesce a svolgere questo compito in maniera molto performante e realizzando un'interfaccia grafica identica in entrambi i casi. Colossi come il New York Times, Alibaba e Baidu hanno già deciso di sviluppare la propria applicazione con Flutter: il fatto che realtà così grosse e importanti abbiano deciso di usare Flutter per la propria applicazione è la miglior riprova possibile della grande qualità e sicurezza che offre.

Come accennato in precedenza, il fatto di non avere la necessità di dover scrivere due volte il sorgente rappresenta il principale vantaggio di Flutter: il tempo di sviluppo è decisamente inferiore, i costi sono più bassi.. La collaborazione tra chi ha il compito di creare il codice gli sviluppatori appunto e i designer, che invece curano la parte grafica dell'applicazione, diventa più snella e veloce, perché per massimizzare il riuso, non è obbligatorio rifare il lavoro due volte. Questo non vuol dire solo essere più veloci nel lavoro, ma anche rendersi conto di quale sia il risultato finale praticamente in tempo reale, dato che Flutter offre anche questa possibilità. Tutto questo non è applicabile solo al mondo delle applicazioni, ma anche ai siti web: questo rende Flutter ancora più unico nel suo genere perché riduce ulteriormente i costi e i tempi di lavoro anche per quanto riguarda il sito internet.

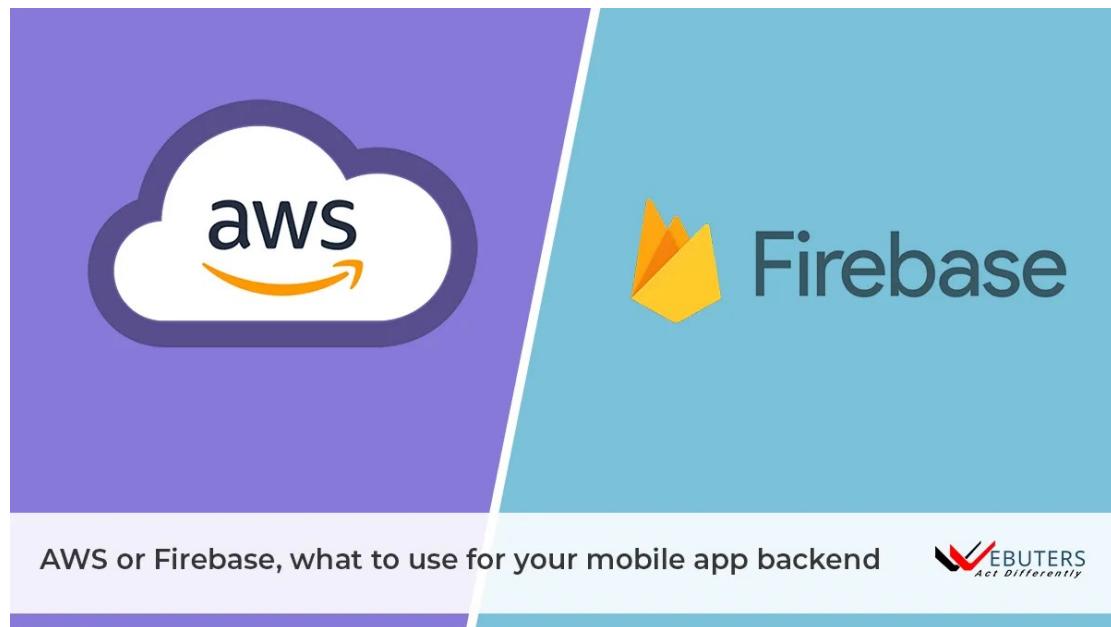
AWS vs Firebase

Come back-end avevo già utilizzato Firebase, quindi è stata una ovvia scelta prenderlo in considerazione. Le scelte erano due, o il grandissimo progetto di casa Google o Amazon AWS.

Anche qui c'è stata una grande ricerca e studio, ho letto tante pagine per poi andare a capire i vari punti di forza delle due piattaforme, Google Firebase vs AWS. Questa è stata una decisione critica che può significare la differenza tra il successo o il fallimento del progetto.

Quando si discute di progetti app, la maggior parte delle persone si concentrano sulle tecnologie di front-end.

Saranno valutati requisiti quali autenticazione, gestione database, funzioni serverless e archiviazione cloud. In conclusione, formuleremo una raccomandazione sulla piattaforma giusta per un progetto di sviluppo di app.



Autenticazione utente

L'autenticazione è un requisito comune per la maggior parte delle app. È necessario autenticare gli utenti e garantire che solo gli utenti autorizzati abbiano accesso ai dati e alle funzionalità.

Autenticazione Firebase

Metodi di accesso multipli

Firebase ha ampie opzioni di autenticazione. Dal nome utente e password standard all'autenticazione di terze parti da piattaforme come Google, Facebook, Apple, Twitter Microsoft e Github.

È anche possibile consentire agli utenti di accedere in una delle opzioni di cui sopra e saranno associati a un account Firebase. Ad esempio: In un caso, l'utente può accedere con il proprio account Facebook. La prossima volta che possono accedere con il proprio account Twitter. Entrambi gli accessi saranno associati a un account Firebase dove è possibile controllare i loro privilegi di accesso.

Prezzi di autenticazione Firebase

- Fino a 10k Autenticazioni SMS al mese -> GRATIS.
- Tutte le altre autenticazioni -> FREE senza restrizioni.

AWS Cognito

Il servizio AWS Cognito consente inoltre di utilizzare piattaforme social come Facebook, Google, Amazon o Apple.

Cognito può anche verificare i numeri di telefono e gli indirizzi e-mail automaticamente inviando i codici di verifica.

Una volta che l'utente riceve il codice di verifica, vengono autenticati con successo.

AWS Cognito Pricing

- Fino a 50k utenti attivi mensili -> GRATIS.
- 50,001-100,000 -> \$0.0055 per utente attivo mensile.

Database Service

La maggior parte delle applicazioni richiederà la possibilità di memorizzare e recuperare i dati nel cloud utilizzando un servizio di database. La tua app potrebbe anche aver bisogno di sincronizzare i dati tra diversi utenti e dispositivi.

Servizi di database Firebase

Firebase ha due servizi di database chiamati Cloud Firestore e Firebase Realtime Database. Entrambi sono database Nosql tuttavia ci sono applicazioni specifiche per ogni database.

In entrambi i servizi di database, l'applicazione può utilizzare un SDK per interagire con il database. Questo significa che non abbiamo bisogno di interagire con un livello di server dell'applicazione. Questo rende estremamente facile per ottenere la vostra applicazione e in esecuzione rapidamente.

Entrambi i database consentono anche la sincronizzazione in tempo reale da più dispositivi.

Alcuni motivi chiave per cui si dovrebbe scegliere un servizio rispetto a un altro che è riassunto di seguito.

Realtime DB vs Firestore

App Requirement	Realtime DB	Firestore
Synchronise small amount of data that changes frequently	Yes	No
Large volume of data with advanced querying, sorting and transactions	No	Yes
Only simple JSON structure is required	Yes	No
Documents stored in collections	No	Yes
App Scalability	200k concurrent users 1k writes per second	1 million concurrent users 10k writes per second

Prezzi in tempo reale DB Firebase (livello FREE)

- Fino a 100 connessioni simultanee
- 1 GB di memoria
- 10 GB Download

Prezzi Firestore (livello FREE)

- 1 Gib Storage
- 10 Gib/mese uscita dalla rete
- 20k/day Document Scrivi
- Il documento 50k/day legge
- 20k/day Cancella documento

AWS Database Services

AWS fornisce una moltitudine di servizi di database dal Nosql Amazon Dynamodb al database relazionale Amazon Aurora Serverless. Entrambi questi server di database sono servizi on demand che si autogestiscono e scalano automaticamente.

Inoltre, AWS offre anche servizi di database basati su istanze che ospitano un'ampia varietà di tipi di database.

Se hai un forte desiderio per la tua app di utilizzare un database relazionale, allora AWS è un chiaro vincitore.

Per questo articolo, confronteremo il database di Amazon Dynamodb Nosql con il servizio Google Firestore che è anche un database Nosql.

Dynamodb è accessibile tramite l'interfaccia API Graphql.

Prima di accedere al database, è necessario creare una mappatura dello schema. Questo dice al codice come le query del database dovranno essere risolte nel database Dynamodb.

AWS offre anche la possibilità di sincronizzare automaticamente i dati sul dispositivo locale e l'archiviazione remota. Come Firestore, questo permette la possibilità di lavorare offline e quindi sincronizzare automaticamente i dati quando la connessione è tornata.

Questo può essere fatto tramite AWS Appsync.

Prezzi della dinamo DB (livello FREE)

- 25 GB di archiviazione dati
- 2,5 milioni di richieste di lettura al mese
- 1 GB di trasferimento dati

Funzioni senza server

Indipendentemente dal tipo di applicazione che si sta sviluppando, probabilmente sarà necessario avere funzionalità lato server.

Questo può venire in varie forme, come la logica aziendale complessa o semplice post elaborazione di contenuti generati dagli utenti, come foto e video.

Prima delle funzioni serverless, avremmo una logica lato server ospitato in entrambi i server basati su cloud come Amazon EC2 o hardware fisico.

La gestione di questa infrastruttura ha richiesto uno sforzo significativo che impedisce di fornire valore ai clienti. Ora possiamo implementare il nostro codice in funzioni senza serverless che sono autogestite. Possono scalare automaticamente per soddisfare i requisiti di carico.

Questo ci permette di concentrarci sullo sviluppo di funzionalità piuttosto che sulla manutenzione dell'infrastruttura.

Funzioni cloud di Firebase

Firebase supporta il codice lato server attraverso le Funzioni Cloud che sono funzioni serverless autogestite.

Possono essere sviluppati solo in Javascript o Typescript all'interno di un ambiente Node.js versione 10.

Dopo aver sviluppato e testato la tua funzione, puoi spingere la funzione verso l'ambiente live con un'interfaccia a linea di comando.

Le funzioni cloud possono essere invocate utilizzando un trigger Firestore da vari servizi Firebase come Firestore. Questo è utile nelle situazioni in cui si desidera invocare una funzione cloud in base a un determinato evento (ad es. quando viene creato un nuovo account Firebase si desidera inviare una email di benvenuto).

Possono anche essere attivati dall'app client utilizzando l'SDK Firebase o dall'app che invoca un endpoint HTTP pubblicato dalla funzione Cloud.

Prezzi delle funzioni cloud Firebase (livello FREE)

- 2 milioni di invocazioni al mese
- 400k di GB secondi e 200k Ghz al mese di tempo di calcolo
- 5 GB di traffico Internet in uscita al mese

Nota: Anche per questo livello gratuito, è necessario avere un account di fatturazione attivo.

L'utilizzo al di sopra del livello gratuito di cui sopra verrà automaticamente addebitato sul conto di fatturazione attivo.

AWS Lambda Funzioni

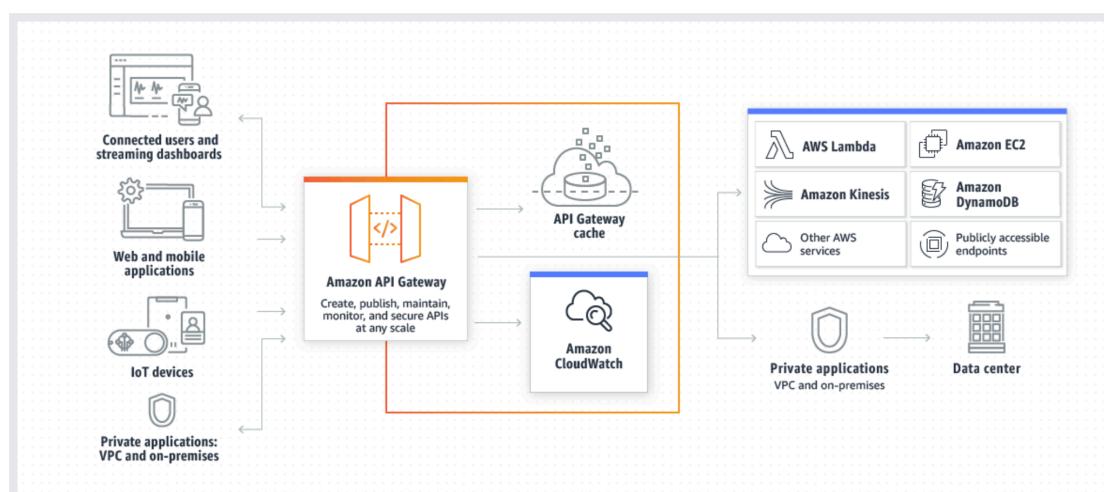
Le funzioni senza serverless in AWS sono denominate funzioni di Lambda di AWS . Consentono di sviluppare funzionalità lato server senza doversi preoccupare della gestione dell'infrastruttura. Saranno anche autoscale per soddisfare le vostre esigenze di carico.

Uno dei principali vantaggi rispetto alle funzioni di Google Cloud è il supporto per molti linguaggi di programmazione.

Le funzioni Lambda supportano nativamente Java, Go, Powershell, Node.js, C#, Python e Ruby. C'è anche un'API di runtime che fornisce supporto per altre lingue. Il supporto linguistico per le funzioni Lambda non può essere abbinato da Firebase. Le funzioni possono anche essere attivate da altri servizi AWS come un trigger da Dynamodb o AWS S3 storage di file. Ciò consente di trasformare i dati man mano che progrediscono attraverso l'infrastruttura AWS.

Per attivare le funzioni Lambda, è necessario implementare Amazon API Gateway. È quindi possibile configurare il gateway per invocare la funzione Lambda.

Ecco l'architettura della soluzione AWS:



Prezzi delle API di Amazon

- 1 milione di chiamate API entro i primi 12 mesi -> GRATIS

- Dopo 12 mesi o se si supera il milione di chiamate API entro i primi 12 mesi -> 1,29 dollari per milione di chiamate fino a 300 milioni di chiamate
- Lambda Funzioni Prezzi
- 1 milione di richieste e 400k GB di secondi di tempo di calcolo al mese -> GRATIS

Cloud Storage

La maggior parte delle applicazioni di successo necessita di memorizzare i file sul cloud. Questo potrebbe essere sotto forma di contenuti generati dagli utenti che devono essere condivisi come immagini o video. È importante notare che questo non è lo stesso di un servizio di database.

Archiviazione cloud di Firebase

Firebase Cloud Storage ha queste caratteristiche:

Le operazioni robuste garantiscono che i caricamenti e i download siano resilienti nelle connessioni fluttuanti.

Integrazione con Firebase Authentication garantendo che solo gli utenti autorizzati possano accedere ai file.

Infrastruttura che fornisce scalabilità Exabyte che potenzia Google Photos e Spotify.

Come per la maggior parte dei servizi Firebase, è possibile sfruttarli facilmente utilizzando Firebase SDK per le diverse piattaforme mobili.

Prezzi Cloud Storage FREE Tier:

- memoria da 5 GB
- 1 GB di download al giorno
- 20k operazioni di upload al giorno
- Operazioni di download di 50k al giorno

Oltre il livello LIBERO:

- Archiviazione: \$0.026/GB al mese
- Download: \$0.12/GB
- Upload operazioni: \$0.05 per 10k operazioni
- Operazioni di download: \$0.004 per operazioni 10k

AWS S3 Storage

AWS ha una piattaforma di storage standard aziendale chiamata S3.

I file sono memorizzati in strutture non gerarchiche, note come buckets. Ogni file memorizzato può aggiungere fino a 10 coppie di valori chiave per descrivere quel file. Questi valori possono anche essere mantenuti per tutta la vita dell'oggetto memorizzato. Sono disponibili rapporti di inventario che aiutano a tenere traccia dei file memorizzati in S3.

Il versioning permette anche il rollback degli oggetti in caso di cancellazione o modifica accidentale. Le cancellazioni accidentali possono anche essere

minimizzate mediante multifactor authentication delete. Ciò richiede più forme di autenticazione quando si tenta di eliminare un oggetto.

La replica è disponibile nelle stesse e diverse zone AWS. Questo migliora la latenza e il disaster recovery per i dati memorizzati.

AWS ha anche monitoraggio e controllo avanzati di S3. Questo ti assicura di tenere traccia non solo di come vengono utilizzate le tue risorse, ma anche di controllare la fatturazione dei tuoi account AWS.

I costi possono essere ulteriormente controllati con classi di stoccaggio diverse. I dati più frequentemente accessibili possono essere memorizzati nello standard S3 con dati meno frequentemente accessibili memorizzati in livelli di archiviazione a basso costo.

Sicurezza e funzionalità di gestione degli accessi sono eccellenti. La sicurezza può essere gestita utilizzando AWS Identity and Access Management (IAM). La sicurezza a grana fine può essere controllata con liste di controllo degli accessi (ACL) e politiche bucket.

La replica è disponibile nelle stesse e diverse zone AWS. Questo migliora la latenza e il disaster recovery per i dati memorizzati.

AWS ha anche monitoraggio e controllo avanzati di S3. Questo ti assicura di tenere traccia non solo di come vengono utilizzate le tue risorse, ma anche di controllare la fatturazione dei tuoi account AWS.

I costi possono essere ulteriormente controllati con classi di stoccaggio diverse. I dati più frequentemente accessibili possono essere memorizzati nello standard S3 con dati meno frequentemente accessibili memorizzati in livelli di archiviazione a basso costo.

Sicurezza e funzionalità di gestione degli accessi sono eccellenti. La sicurezza può essere gestita utilizzando AWS Identity and Access Management (IAM). La sicurezza a grana fine può essere controllata con liste di controllo degli accessi (ACL) e politiche bucket.

AWS S3 Pricing

- Storage costs for first 50 TB / Month: \$0.025 per GB

In conclusione la scelta è ricaduta sulla piattaforma Firebase, complice anche la stessa casa madre di Flutter, framework scelto per la realizzazione delle applicazioni Android e IOS. Firebase ha una grande community dietro con siti dove è possibile reperire tanti esempi o domande a cui è già stata data una risposta.

Alla fine del mio primo mese avevo scelto la tecnologia da usare nel mio progetto.

Firebase e Flutter.

Autenticazione

L'autenticazione è il processo attraverso il quale viene verificata l'identità di un utente che vuole accedere ad un computer o ad una rete. È il sistema che verifica, effettivamente, che un individuo è chi sostiene di essere. L'autenticazione è diversa dall'identificazione, ovvero la determinazione che un individuo sia conosciuto o meno dal sistema, e dall'autorizzazione, cioè il conferimento ad un utente del diritto ad accedere a specifiche risorse del sistema, sulla base della sua identità.

L'autenticazione è l'atto di confermare la verità di un attributo di una singola parte di dato o di una informazione sostenuta veramente da un'entità. In contrasto con l'identificazione, che si riferisce all'atto di confermare l'identità di una persona o qualcosa, l'autenticazione è il processo di confermare davvero l'identità. Per esempio confermando l'identità di una persona attraverso la convalida dei suoi documenti di identità, verificando l'autenticità di un sito web con un certificato digitale, determinando l'età di un manufatto dalla datazione al carbonio, o di garantire che un prodotto è quello indicato dall'etichetta d'imballaggio. In altre parole, l'autenticazione comporta spesso verificare la validità di almeno una forma di identificazione.

Si definisce in informatica il processo tramite il quale un sistema informatico, un computer, un software o un utente verifica la corretta, o almeno presunta, identità di un altro computer, software o utente che vuole comunicare attraverso una connessione, autorizzandolo ad usufruire dei relativi servizi associati. È il sistema che verifica, effettivamente, che un individuo è chi sostiene di essere. L'autenticazione è diversa dall'identificazione (la determinazione che un individuo sia conosciuto o meno dal sistema) e dall'autorizzazione (il conferimento ad un utente del diritto ad accedere a specifiche risorse del sistema, sulla base della sua identità).

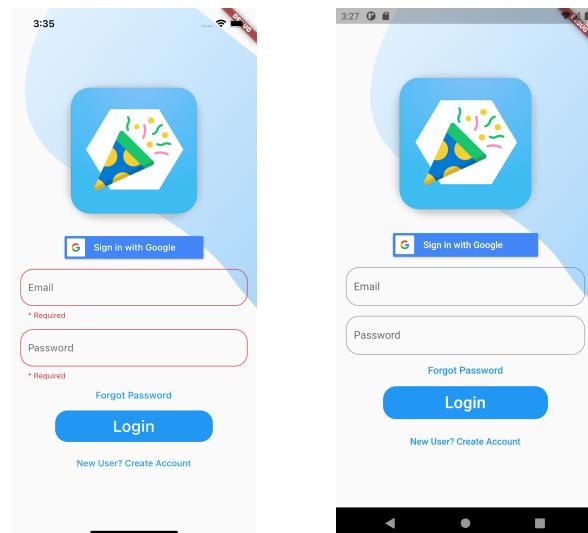
È un punto fondamentale per la riuscita di una applicazione, bisogna anche tenere conto di due aspetti fondamentali: la registrazione di un nuovo utente e il login di una persona già registrata.

Per il mio progetto ho notato che in fase di registrazione ho bisogno sostanzialmente di un nome utente, una email e una password. Per l'autenticazione mi appoggio al sistema di firebase, già completo e con un altissimo livello di sicurezza, vista la sua provenienza dalla casa madre Google.

Inizio il mio processo di autenticazione controllando se all'interno della memoria del telefono esista già un utente loggato, cosa fondamentale per la praticità della applicazione. Personalmente penso sia uno degli aspetti più importanti, questo lo riesco a fare grazie alla mia classe Authenticate che svolge molti compiti. Controlla se esiste un utente, e in base alla ciò che la sua email sia stata verificata, controllo molto importante per mantenere uno standard di sicurezza, in caso non esistesse ci indirizza al login.

```
class Authenticate extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        final firebaseUser = context.watch<User>();  
        if (firebaseUser != null) {  
            if (!firebaseUser.emailVerified) {  
                return ConfirmEmail();  
            } else {  
                return HomePage();  
            }  
        }  
        return SignInPage();  
    }  
}
```

Questo pezzo di codice è il primo ad essere eseguito dalla mia applicazione. Quando un utente apre la applicazione per la prima volta viene inevitabilmente indirizzato sulla pagina per il login e si ritrova questa schermata:



In questa schermata può inserire la sua email e la sua password verificando l'esistenza e la correttezza premendo il pulsante login.

La funzione che uso per verificare le due cose sopra elencate è questa:

Sign-in

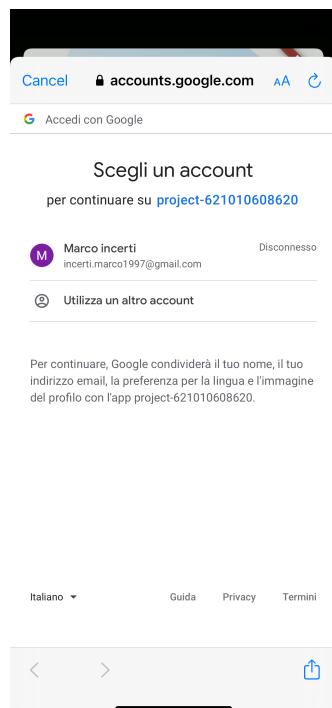
To sign-in to an existing account, call the `signInWithEmailAndPassword()` method:

```
try {
    UserCredential userCredential = await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: "barry.allen@example.com",
        password: "SuperSecretPassword!"
    );
} on FirebaseAuthException catch (e) {
    if (e.code == 'user-not-found') {
        print('No user found for that email.');
    } else if (e.code == 'wrong-password') {
        print('Wrong password provided for that user.');
    }
}
```

Dove, nel caso uno dei due campi sia sbagliato, manda un messaggio di errore.

Questa parte è stata molto complicata perché deve esserci un controllo su tutti i campi usati per la sicurezza dell'account, ad esempio bisogna verificare che la email sia un indirizzo valido o che la password sia sicura. Banalmente bisogna controllare che entrambi i campi siano compilati.

Poiché i pulsanti sono tanti andiamo ad analizzarli uni per uno:



Ho inserito la possibilità di loggarsi anche con un account google, grazie anche Firebase che permette di attivare questa opzione anche sulla console.

Questo punto è fondamentale per la buon riuscita della app affinché le persone non perdano più tempo a crearsi nuovi account, ma accedano più rapidamente tramite Google.

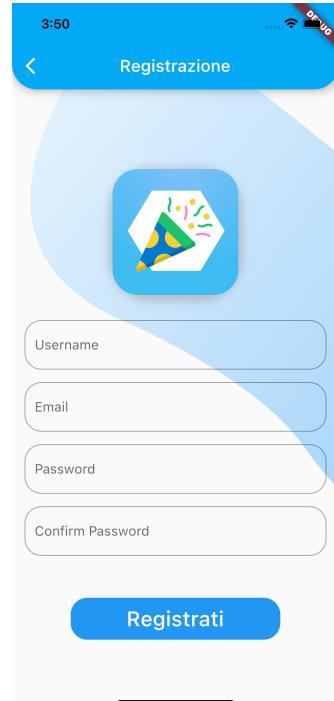
Inizialmente si va a creare la funzione che ci apre il provider di google dove andremo a selezionare il nostro account, la comodità di tutto questo è che tutti i controlli e la sicurezza sono stati implementati dalla azienda multinazionale, quindi è tutto a carico loro semplificandoci la vita.

In questa sezione invece c'è la vera e propria creazione dell'utente.

Un tratto fondamentale è il controllo di tutti i parametri che l'utente andrà ad inserire, corrispettivi dei messaggi di errore per indirizzarlo nel modo più semplice alla creazione del proprio account.

Pochi campi per una perdita di tempo relativamente piccola.

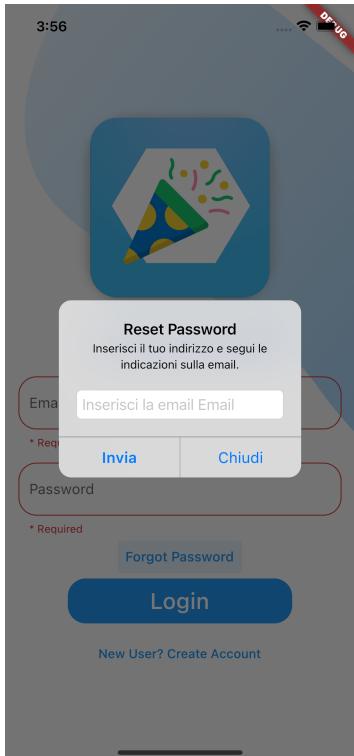
Una grande difficoltà qua è stato il fatto che Firebase non crea un documento per ogni user, quindi siamo dovuti ad andare a farlo noi. Prima della fine della creazione dell'utente abbiamo implementato una funzione che in automatico andava a creare un documento nella raccolta "users" con tutti i campi aggiunti dell'utente.



Se la creazione va a buon fine l'utente si ritrova nella home e, se chiude la applicazione e la riapre, si ritrova il suo utente già loggiato. Un'altra scomodità per l'utente difficile da notare e da implementare è il fatto che quando si seleziona una casella per scriverci dentro non si richiude la tastiera del telefono dopo, un problema risolto grazie alla possibilità di cliccare in una parte qualunque dello schermo per togliere il focus dal TextField.

```
Future<UserCredential> addUser(value) async {
    // Call the user's CollectionReference to add a new user
    final docSnapshot =
        await firestore.collection("users").doc(value.uid).get();

    if (!docSnapshot.exists) {
        await firestore.collection("users").doc(value.uid).set({
            "uid": value.uid,
            "email": value.email,
            "username": _nameController.text,
            "creazione": FieldValue.serverTimestamp(),
            "tags_interesse": [],
            "universita": "",
            "grafico": {},
            "attiva_filtrri": true,
        }).then((_) {
            print("success!");
        });
    }
}
```



E se l'utente non ricorda la sua password?

Punto fondamentale.

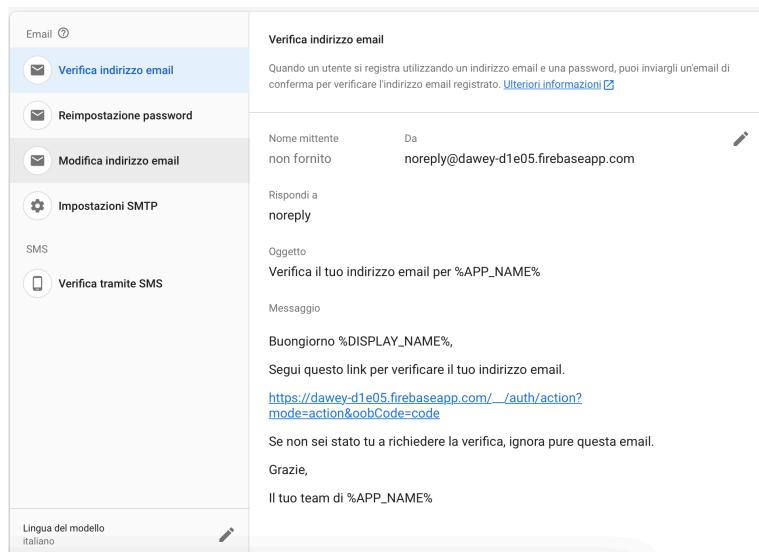
Se un utente non ricorda la sua password gli basta cliccare sul pulsante apposito, dove si vedrà apparire una sovrapposizione che gli chiede di inserire la sua email. Una volta inserita Firebase gli invia una email in automatico con un link composto da un token univoco per quel specifico utente, che lo reindirizzerà a una pagina web che gli darà la possibilità di inserire una nuova password.

Noi possiamo andare a cambiare il template che Firebase andrà ad inviare all'utente mediante la console del sito. Anche per queste funzioni ci appoggiamo al plugin su FlutterFire.

Una volta registrata con il classico metodo mediante la seconda schermata della applicazione e tutto il procedimento è andato a buon fine gli rimane il controllo a die fattori.

L'autenticazione a due fattori (2FA) è un metodo di autenticazione sicura per sistemi e piattaforme informatiche e consiste nell'utilizzo di due metodi invece che uno, ad esempio l'inserimento di una password e la scansione dell'impronta digitale. Spesso viene confusa con la verifica in due passaggi (2SV), ma non sono la stessa cosa. L'autenticazione a due fattori protegge efficacemente gli account perché aggiunge un livello di sicurezza in più, rendendo più difficile l'accesso ad hacker e utenti non autorizzati.

Quindi quando un utente si registra riceve in automatico una email con questo template.





Con un link univoco che dovrà cliccare per poter accedere al suo account nella app.

Ho implementato una funzione che invia il la email di verifica e un sistema che controlli che l'utente abbia validato la sua email prima di farlo accedere alla home del sito. Se la email non è stata ancora verificata l'utente sarà costretto ad aspettare bloccato su questa schermata, con la possibilità di rinviare la email in qualunque momento possibile.

Appena l'utente avrà verificato la sua email gli sarà possibile accedere all'intera applicazione.

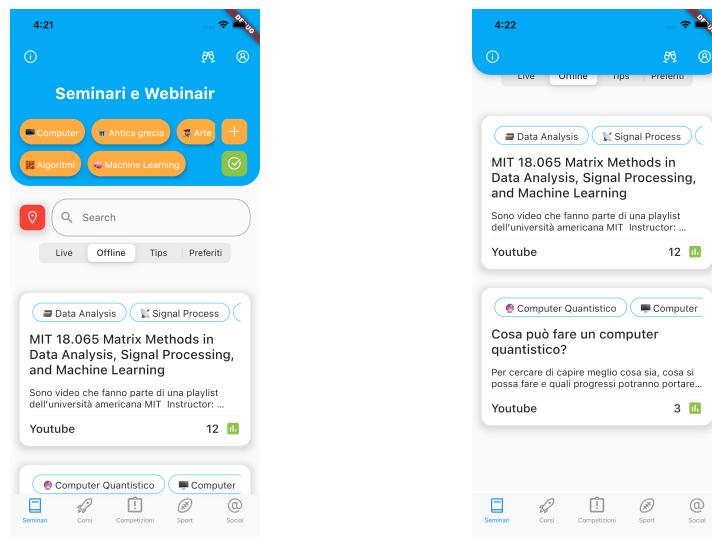
Homepage

La homepage è il fulcro della applicazione, dove si può entrare nel vivo della applicazione.

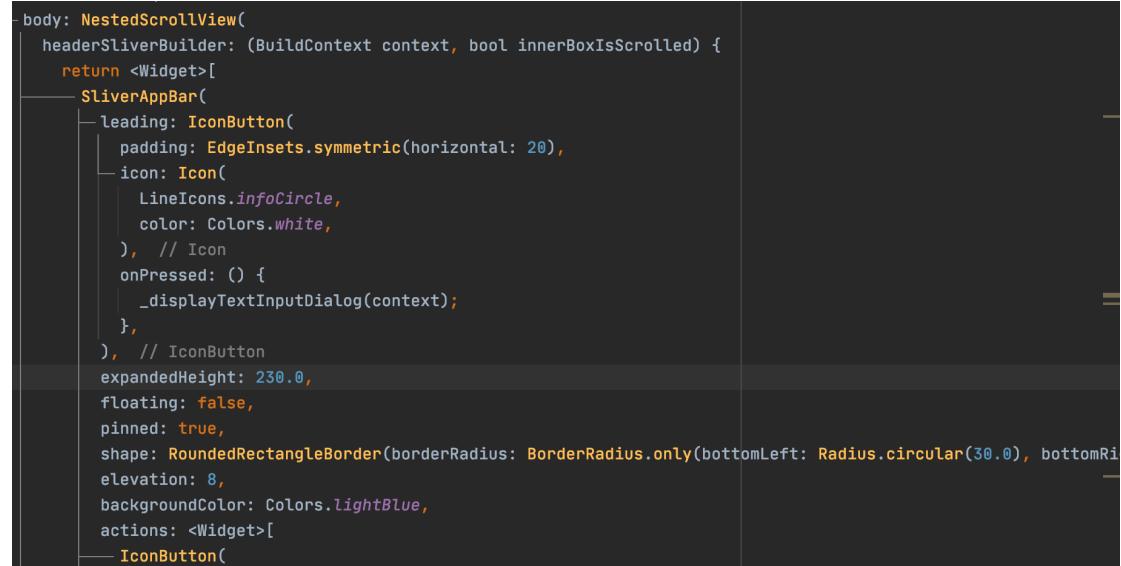
L'*home page* (letteralmente *pagina di casa*), anche chiamata inizio, pagina d'inizio, pagina iniziale o pagina principale è solitamente la prima pagina di un sito web o un'applicazione web o la prima pagina quando si apre un browser che si può anche richiamare con apposita opzione o tasto *home* del browser (se correttamente impostata) o di alcune tastiere multimediali. La homepage, come l'intero sito, deve essere accessibile e usabile.

L'accessibilità è la caratteristica di un dispositivo, di un servizio, di una risorsa o di un ambiente d'essere fruibile con facilità da una qualsiasi tipologia d'utente.

Il termine è comunemente associato alla possibilità anche per persone con ridotta o impedita capacità sensoriale, motoria, o psichica (ovvero affette da disabilità sia temporanea, sia stabile), di accedere e muoversi autonomamente in ambienti fisici (per cui si parla di accessibilità fisica), di fruire e accedere autonomamente a contenuti culturali (nel qual caso si parla di accessibilità culturale) o fruire dei sistemi informatici e delle risorse a disposizione tipicamente attraverso l'uso di tecnologie assistive o tramite il rispetto di requisiti di accessibilità dei prodotti. L'usabilità è definita dall'ISO (*International Organization for Standardization*), come *l'efficacia, l'efficienza e la soddisfazione con le quali determinati utenti raggiungono determinati obiettivi in determinati contesti*. In pratica definisce il grado di facilità e soddisfazione con cui si compie l'interazione tra l'uomo e uno strumento (console, leva del cambio, interfaccia grafica, ecc.). Il problema dell'usabilità si pone quando il modello del progettista (ovvero le idee di questi riguardo al funzionamento del prodotto, che trasferisce sul *design* del prodotto stesso) non coincide con il modello dell'utente finale (ovvero l'idea che l'utente concepisce del prodotto e del suo funzionamento). Il grado di usabilità si innalza proporzionalmente all'avvicinamento dei due modelli (modello del progettista, e modello dell'utente).



La mia hompage è strutturata a blocchi, innanzitutto possiamo notare che una buona parte dello schermo è occupata dalla nostra appBar, per la creazione di quest'ultima ho impegnato molto tempo e risorse.sull'aspetto del design ho praticamente ridisegnato e riprogettato la appBar molte volte fino a raggiungere questa finale.



```
body: NestedScrollView(
  headerSliverBuilder: (BuildContext context, bool innerBoxIsScrolled) {
    return <Widget>[
      SliverAppBar(
        leading: IconButton(
          padding: EdgeInsets.symmetric(horizontal: 20),
          icon: Icon(
            LineIcons.infoCircle,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            _displayTextInputDialog(context);
          },
        ), // IconButton
        expandedHeight: 230.0,
        floating: false,
        pinned: true,
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.only(bottomLeft: Radius.circular(30.0), bottomRight: Radius.circular(30.0))),
        elevation: 8,
        backgroundColor: Colors.lightBlue,
        actions: <Widget>[
          IconButton(

```

Praticamente è una NestedScrollView con all'interno una silverAppBar.

- **NestedScrollView:** Una vista a scorrimento all'interno della quale possono essere annidate altre viste a scorrimento, con le loro posizioni di scorrimento intrinsecamente collegate. Il caso d'uso più comune per questo widget è una vista scrollabile con una Sliverappbar flessibile contenente una Tabbar nell'intestazione (costruita da headerSliverBuilder, e con una Tabbarview nel corpo, in modo che il contenuto della vista scrollabile varia in base a quale scheda è visibile. In una normale vista a scorrimento, c'è un insieme di frammenti (i componenti della vista a scorrimento). Se uno di questi frammenti ospitava una Tabbarview che scorre nella direzione opposta (ad es. consentendo all'utente di scorrere orizzontalmente tra le pagine rappresentate dalle schede, mentre l'elenco scorre verticalmente), quindi qualsiasi elenco all'interno di tale Tabbarview non interagirebbe con la vista di scorrimento esterna. Per esempio, lanciare la lista interna per scorrere verso l'alto non causerebbe un collasso Sliverappbar nella vista Scroll esterna per espandersi. Nestedscrollview risolve questo problema fornendo controller Scroll personalizzati per la Scrollview esterna e le Scrollview interne (quelli all'interno della Tabbarview, agganciandoli insieme in modo che appaiano, all'utente, come una visualizzazione di scorrimento coerente).

- **SliverAppBar:** Una barra delle applicazioni è costituito da una barra degli strumenti e potenzialmente altri widget, come una barra delle schede e una barra spaziatrice Flexiblespacebar. Le barre delle app di solito espongono una o più azioni comuni con IconButton, eventualmente seguite da un pulsante Popupmenubutton per operazioni meno comuni. Le barre delle app Sliver sono tipicamente utilizzate come il primo figlio di una vista Customscroll, che consente alla barra delle applicazioni di integrarsi con la vista di scorrimento in modo che possa variare in altezza in base allo spostamento di scorrimento o fluttuare sopra l'altro contenuto nella vista di scorrimento. Per una barra app ad altezza fissa nella parte superiore dello schermo, vedere Appbar, che viene utilizzato nel Scaffold.appBar.

Questo uso combinato di widget mi permette di far scomparire piano piano la mia appBar mentre l'utente scrolla la schermata.

Subito c'è il titolo della pagina in cui siamo, con subito sotto i tags di interesse dell'utente.

```
StreamBuilder<DocumentSnapshot>(
    stream: FirebaseFirestore.instance.collection('users').doc(_auth.currentUser.uid).snapshots(),
    builder: (BuildContext context, AsyncSnapshot<DocumentSnapshot> snapshot) {
        if (snapshot.hasError) {
            return Text('Something went wrong');
        }
        if (snapshot.connectionState == ConnectionState.waiting) {
            return SizedBox();
        }

        Map<String, dynamic> data = snapshot.data.data();
        return List.from(data["tags_interesse"]).isNotEmpty
```

Grazie allo StreamBuilder vado a selezionare il documento nel mio DB che corrisponde all'utente corrente, una volta ricevuto vado a estrarre i tags che ha selezionato per poi andarli a esporre.

- **StreamBuilder:** Widget che si costruisce in base all'ultima istantanea di interazione con un flusso. La ricostruzione del widget è pianificata da ogni interazione, utilizzando State.setstate, ma è altrimenti disaccoppiata dai tempi del flusso. Il costruttore è chiamato a discrezione della pipeline di Flutter, e riceverà quindi una sub-sequenza delle istantanee dipendente dal tempo che rappresentano l'interazione con il flusso. I dati dell'istantanea iniziale possono essere controllati specificando initialData. Questo dovrebbe essere usato per garantire che il primo fotogramma abbia il valore atteso, dato che il costruttore sarà sempre chiamato prima che l'ascoltatore abbia la possibilità di essere processato.

Vado a creare una listView alla quale vado a dare i tags dell’utente dove, grazie alla funzione qua sotto, mi crea delle chips da mostrare.

```
Widget _buildChip(String label, Color color, int index) {
    if (index > 4) {
        return Container();
    } else {
        return Container(
            padding: EdgeInsets.only(top: 5, bottom: 5, left: 5, right: 5),
            child: Chip(
                labelPadding: EdgeInsets.all(1.0),
                label: Text(
                    label,
                    style: TextStyle(
                        color: Colors.white,
                    ), // TextStyle
                ), // Text
                backgroundColor: color,
                elevation: 6.0,
                shadowColor: Colors.grey[60],
                padding: EdgeInsets.all(8.0),
            ), // Chip
        ); // Container
    }
}
```

Tags

I tags sono un sistema per creare un raccomandation system molto basilare, dove l’utente può selezionare i suoi interessi per poi avere nella home solo gli eventi più interessanti per lui.

Il nostro punto di arrivo vuole essere però un sistema ibrido: I sistemi di maggior successo utilizzano approcci ibridi che combinano entrambi i metodi di filtraggio. Gli approcci ibridi possono essere implementati in diversi modi, facendo separatamente e poi combinando le previsioni basate sui contenuti e quelle basate sulla collaborazione. Diversi studi hanno confrontato le prestazioni dell’ibrido con i metodi puramente collaborativi o basati sui contenuti e hanno dimostrato che quelli ibridi possono fornire raccomandazioni più accurate rispetto agli approcci puri.

Questi metodi possono essere utilizzati per superare alcuni dei problemi comuni come “l’avviamento a freddo” e il “problema della scarsità”, che si riscontrano all’inizio dell’applicazione, quando i dati sono pochi, di qualità non certa e che quindi danno risultati poco affidabili.

Netflix è un buon esempio di sistema ibrido: formula raccomandazioni confrontando le abitudini di visione e di ricerca di clienti simili (quindi con il filtraggio collaborativo) e offrendo film che condividono caratteristiche con film che un cliente ha valutato positivamente (quindi con il filtraggio basato sui contenuti).

Sono molto importanti le spiegazioni e le note che vengono affiancate alle raccomandazioni. Queste spiegazioni hanno lo scopo sia di fornire informazioni sugli

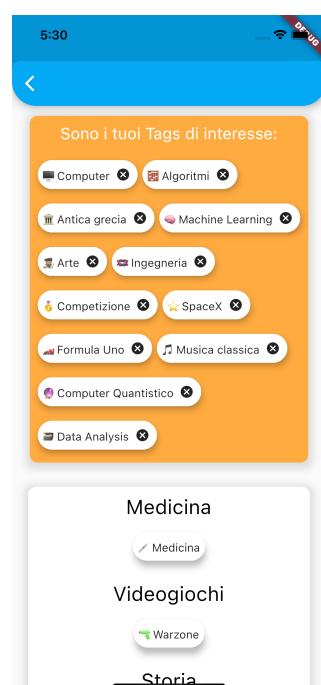
elementi raccomandati sia di convergere verso un maggiore coinvolgimento degli utenti, grazie ad una maggiore trasparenza delle raccomandazioni stesse.

Gli utenti rispondono alle spiegazioni in maniera differente e di conseguenza vi è la necessità di ottimizzare sia le voci che le spiegazioni. Gli stessi utenti possono avere reazioni differenti di fronte alle spiegazioni a seconda del contesto e dell'intenzione, ad esempio, le persone che utilizzano apparecchiature mobili, es. i cellulari, possono richiedere raccomandazioni più concise. Nel caso di YouTube il problema è leggermente diverso: quando gli utenti guardano la pagina, viene visualizzato un elenco di video consigliati che l'utente potrebbe gradire in un ordine diverso da quello proposto.

L'algoritmo si concentra su due obiettivi: di coinvolgimento (clic, tempo trascorso) e di soddisfazione (mi piace) ma occorre ridurre il “bias (distorsione) di selezione”.

Questa distorsione è introdotta dal sistema in quanto gli utenti, a causa della posizione sullo schermo, hanno maggiori probabilità di fare clic sulla prima raccomandazione, anche se i video in posizione più bassa potrebbero aumentare il coinvolgimento e la soddisfazione. Il modello genera una previsione per ciascuno degli obiettivi. Le caratteristiche del video corrente (contenuto, titolo, argomento, tempo di caricamento, ecc.) ed i dati dell'utente che sta guardando (tempo, profilo utente, ecc.) vengono utilizzati come input. Si utilizza inoltre un algoritmo che considera i video su cui è stato fatto clic precedentemente ed il dispositivo utilizzato per guardare i video stessi, i vari modelli vengono poi combinati per ottenere il risultato atteso.

In conclusione, i sistemi di raccomandazione comprendono un insieme di tecniche e algoritmi in grado di suggerire agli utenti elementi “rilevanti”, raggiungendo l'obiettivo.



Ma in due mesi il tempo a disposizione è poco quindi mi sono limitato a fornire dei tags, ognuno sotto una specifica categoria. Tutti i tags vengono sincronizzati in tempo reale con il database online in tempo reale.

L'utente può selezionarli in base alle sue preferenze e, una volta selezionati, vengono automaticamente aggiunti alla lista dei tags preferiti. Grazie a quest'ultimi gli eventi che conterranno almeno uno di questi tags verranno mostrati in ordine cronologico divisi per categoria.

I tags già presenti nei preferiti non vengono mostrati in quelli generali.

Le funzioni usate per la sincronizzazione sono:

Questa funzione viene chiamata appena la pagina si crea, e vado a creare i collegamenti ai miei documenti nel database online.

```
@override  
void initState() {  
    collectionStream = FirebaseFirestore.instance.collection('tags').snapshots();  
    tags_account = FirebaseFirestore.instance.collection('users').doc(_auth.currentUser.uid).snapshots();  
}
```

Una volta creati vengono chiamati nello StreamBuilder, dove analizza lo stato della chiamata mostrando una pagina di caricamento mentre viengono scaricati i dati. Una volta scaricati viene creata una lista.

```
StreamBuilder<DocumentSnapshot>(  
    stream: tags_account,  
    builder: (BuildContext context, AsyncSnapshot<DocumentSnapshot> snapshot) {  
        if (snapshot.hasError) {  
            return Text('Something went wrong');  
        }  
  
        if (snapshot.connectionState == ConnectionState.waiting) {  
            return Container(  
                child: Center(  
                    child: Loading(indicator: BallPulseIndicator(), size: 100.0, color: Colors.lightBlue),  
                ), // Center  
            ); // Container  
        }  
  
        user_doc = snapshot.data.data();  
  
        _dynamicChips = List.from(user_doc["tags_interesse"]);
```

Quando un utente clicca su un tags bisogna toglierlo da quelli generali ed aggiungerlo a quelli dell'utente.

Con questa funzione vado a inserire il tags nel documento personale dell'utente e grazie alla funzione setState() vado a "ricaricare" la pagina togliendo il tags dalla lista di quelli generali.

```
FirebaseFirestore.instance.collection("users").doc(_auth.currentUser.uid).update({  
    "tags_interesse": FieldValue.arrayUnion([lista[index]]),  
    "grafico.${document.data()['categoria']}": numero,  
}).then((value) => ScaffoldMessenger.of(context).showSnackBar(  
    SnackBar(  
        duration: const Duration(seconds: 1),  
        content: Text('Tag aggiunto!'),  
    ), // SnackBar  
>);  
setState(() {  
    lista.remove(lista[index]);  
});
```

È stata implementata anche una ricerca per riuscire a trovare i tags più velocemente.
È una ricerca molto basilare che va a mostrare solo i tags che matchano con la parola cercata.

In questo caso la `searchBox` è stata aggiunta dentro la `appBar` per cercare di avere un design della applicazione più avanguardista possibile.



Il grafico dei tags di interesse dell'utente

Nella schermata riservata all'utente è stato implementato un grafico che mostra le categorie più interessanti per l'utente. Questo è stato fatto per dare alla possibilità all'utente di capire quale siano i suoi interessi ed avere così una panoramica più completa degli eventi che gli verranno proposti.

E il secondo motivo è quello di raccogliere le categorie più cercate dagli utenti per poi farci degli studi e capire cosa vogliono gli utenti che usano la applicazione.

Tutte queste informazioni sono già calcolate mentre l'utente modifica la sua lista dei tags.



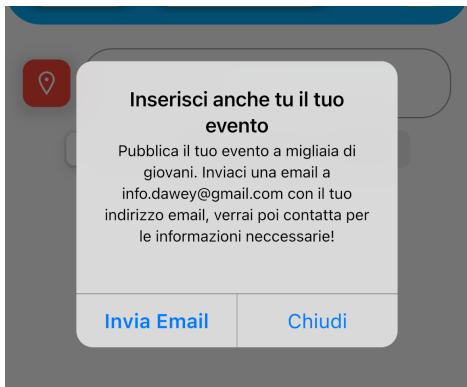
Con questa funzione creo il grafico dentro a un container.

In questo momento devo controllare il numero di elementi che contiene l'array che viene scaricato dal database.

Gli elementi possono andare da 0 a 4 e vengono presi solo le 4 categorie più grandi.

```
- Container(
  height: 150,
  width: 150,
  child: PieChart(
    PieChartData(
      pieTouchData: PieTouchData(touchCallback: (pieTouchResponse) {
        setState(() {
          final desiredTouch =
            pieTouchResponse.touchInput is! PointerExitEvent && pieTouchResponse.touchInput is! PointerUpEvent;
          if (desiredTouch && pieTouchResponse.touchedSection != null) {
            touchedIndex = pieTouchResponse.touchedSection.touchedSectionIndex;
          } else {
            touchedIndex = -1;
          }
        });
      }),
      // PieTouchData
      borderData: FlBorderData(
        show: false,
      ),
      // FlBorderData
      sectionsSpace: 0,
      centerSpaceRadius: 10,
      sections: showingSections(), // PieChartData
    ), // PieChart
  ), // Container
```

Richiesta di pubblicazione di un evento



Ho implementato un modal che permette a chiunque di inviare una email per segnalare o fare richiesta di inserimento di un evento. Lo studente premendo su invia email apre in automatico la propria app di default per la gestione delle email con l'oggetto e il destinatario già inseriti.

Questo punto da alla applicazione la vera fonte di guadago, permette alle aziende o alle persone di mettersi in contatto con noi per stabilire la correttezza e l'importanza dell'evento che propongono, per poi procedere al pagamento e l'inserimento dell'evento.

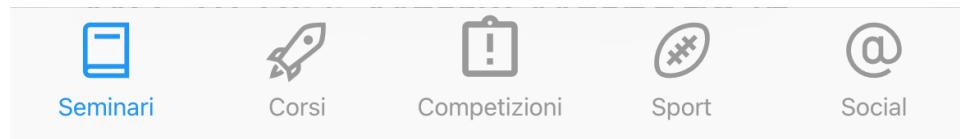
L'evento non viene aggiunto in automatico per non permettere alle persone di creare eventi che non esistono o con informazioni fasulle.

Si era pensato anche a un sistema per inserire gli eventi in automatico da i vari siti, ma non esiste ancora un metodo per inserire i tags con un errore fattibile.

La selezione dei tags è una procedura molto importante e delicata, perché sono l'unico metodo per differenziare gli eventi tra di loro e di mostrare all'utente solo quelli interessanti per lui.

I tipi di eventi

Gli eventi sono divisi in 5 categorie per permettere di avere una selezione più ampia e riuscire ad offrire un bacino di proposte più ampio. Anche se ci sono diverse tipologie il filo conduttore di tutti gli eventi è la cultura o l'università, cioè ognuno di essi punta a migliorare un aspetto della persona.



Le categorie sono:

- **Seminari:** i seminari comprendono tutti gli eventi a sfondo culturali, ad esempio anche gli webinars, cioè tutte quelle situazioni che porta etimologicamente con sé un concetto di formazione. I seminari possono comprendere anche video su YouTube o podcast, questa per me è stata una grande aggiunta per rendere ancora più inclusiva la proposta;
- **Corsi:** Per corsi si intendono delle lezioni o comunque un qualsiasi tipo di attività che comporta l'acquisizione di nozioni e strumenti utile per ottenere le qualifiche necessarie per svolgere una professione. I corsi sono perfetti per ottenere tutte le conoscenze che servono per lavorare in determinati settori. Questi corsi di dividono poi in varie categorie, a seconda degli argomenti trattati;
- **Competizioni:** Per competizioni intendiamo tutte le gare interne ed esterne all'università che possono trattare dalle competizioni di codice fino ai tornei di scacchi. Molte aziende usano queste competizioni per trovare nuovi talenti da poi reclutare all'interno della propria azienda;
- **Sport:** questa categoria è riservata agli sport, cioè tutte le attività fisiche collegate alla università principalmente, ad esempio tutte le manifestazioni organizzate dall'organo predisposto per lo sport. Questa categoria punta molto sulle relazioni interpersonali;
- **Social:** Tutti sappiamo che i momenti di svago sono importanti quanto quelli di apprendimento, quindi abbiamo pensato di riservare questa categoria a tutti gli eventi che si terranno sui social, che possono andare dalle dirette dei lanci spaziali a una live di un influencer.

TabBar

Un widget di progettazione materiale che visualizza una fila orizzontale di schede.
Tipicamente creato come Appbar.bottom parte di una Appbar e in combinazione con una Tabbarview.

Se un Tabcontroller non è fornito, allora deve essere fornito un antenato Defaulttabcontroller. Il controller della scheda Tabcontroller.length deve essere uguale alla lunghezza della lista delle schede e alla lunghezza della lista Tabbarview.children.

Richiede che uno dei suoi antenati sia un widget Materiale.

```
— bottomNavigationBar: CupertinoTabBar(
  items: const <BottomNavigationBarItem>[
    BottomNavigationBarItem(
      icon: Icon(LineIcons.book),
      label: 'Seminari',
    ), // BottomNavigationBarItem
    BottomNavigationBarItem(
      icon: Icon(LineIcons.rocket),
      label: 'Corsi',
    ), // BottomNavigationBarItem
    BottomNavigationBarItem(
      icon: Icon(Icons.assignment_late_outlined),
      label: 'Competizioni',
    ), // BottomNavigationBarItem
    BottomNavigationBarItem(
      icon: Icon(LineIcons.footballBall),
      label: 'Sport',
    ), // BottomNavigationBarItem
    BottomNavigationBarItem(
      icon: Icon(LineIcons.at),
      label: 'Social',
    ), // BottomNavigationBarItem
  ], // <BottomNavigationBarItem>[]
  currentIndex: _selectedIndex,
  onTap: _onItemTapped,
), // CupertinoTabBar
```

```
void _onItemTapped(int index) {
    setState(() {
        _selectedIndex = index;
        switch (index) {
            case 0:
                titolo = "Seminari e Webinair";
                break;
            case 1:
                titolo = "Corsi";
                break;
            case 2:
                titolo = "Competizioni e concorsi";
                break;
            case 3:
                titolo = "Eventi sportivi";
                break;
            case 4:
                titolo = "Eventi social";
                break;
        }
    });
}
```

Questo widget mi permette di creare una TabBar con 5 elementi BottomNavigationBar, ognuno composto da una icona e un testo. Ci permette anche di decidere quale Tab mostrare mediante l'uso di un indice, che viene modificato dalla funzione `_onItemTapped()` quando l'utente decide di cambiare pagina. Questa è la funzione che cambia il contenuto della pagina e anche il titolo contenuto nella app Bar.

Invece questa è la lista delle pagine che verrano aperte al cambiare dell'indice.

Ognuna per una categoria diversa.

```
List<Widget> _widgetOptions = <Widget>[
    SeminariPage(auth: _auth),
    CorsiPage(auth: _auth),
    CompetizioniPage(auth: _auth),
    SportiviPage(auth: _auth),
    SocialPage(auth: _auth),
]; // <Widget>[]
```

Eventi offline e online

Ho fatto ulteriore distinzione tra gli eventi aggiungendo un filtro che mostra gli eventi con una data e un orario specifico e quelli che invece sono accessibili sempre e ovunque.

Per creare questo filtro ho usato CupertinoSlidingSegmentedControl, visualizza gli oggetti forniti nella mappa dei children in un elenco orizzontale. Consente all'utente di selezionare tra una serie di opzioni reciprocamente esclusive, toccando o trascinando all'interno del controllo segmentato. Un controllo segmentato può caratterizzare qualsiasi Widget come uno dei valori nella sua Mappa dei children. Il tipo T è il tipo di tasti Mappa utilizzati per identificare ogni widget e determinare quale widget è selezionato. Come richiesto dalla classe Mappa, le chiavi devono essere di tipo coerente e comparabili. L'argomento figli deve essere una Mappa ordinata come una LinkedHashMap, l'ordine delle chiavi determinerà l'ordine dei widget nel controllo segmentato. Il widget chiama il callback onValueChanged quando un gesto utente valido completa su un segmento non selezionato. La chiave mappa associata al widget appena selezionato viene restituita nel callback onValueChanged.

In genere, i widget che usano un controllo segmentato ascolteranno la callback onValueChanged e ricostruiranno il controllo segmentato con un nuovo gruppo Value per aggiornare quale opzione è attualmente selezionata. I children saranno visualizzati nell'ordine dei tasti nella mappa, lungo la direzione di testo corrente. Ogni widget Child avrà la stessa dimensione. L'altezza del controllo segmentato è determinata dall'altezza del widget Child più alto. La larghezza di ogni Child sarà la larghezza intrinseca del Child più largo, o lo spazio orizzontale disponibile diviso per il numero di Children, che è sempre più piccolo.

```
CupertinoSlidingSegmentedControl(  
    groupValue: segmentedControlGroupValue,  
    children: myTabs,  
    onValueChanged: (i) {  
        setState(  
            () {  
                segmentedControlGroupValue = i;  
            },  
        );  
    },  
, // CupertinoSlidingSegmentedControl
```

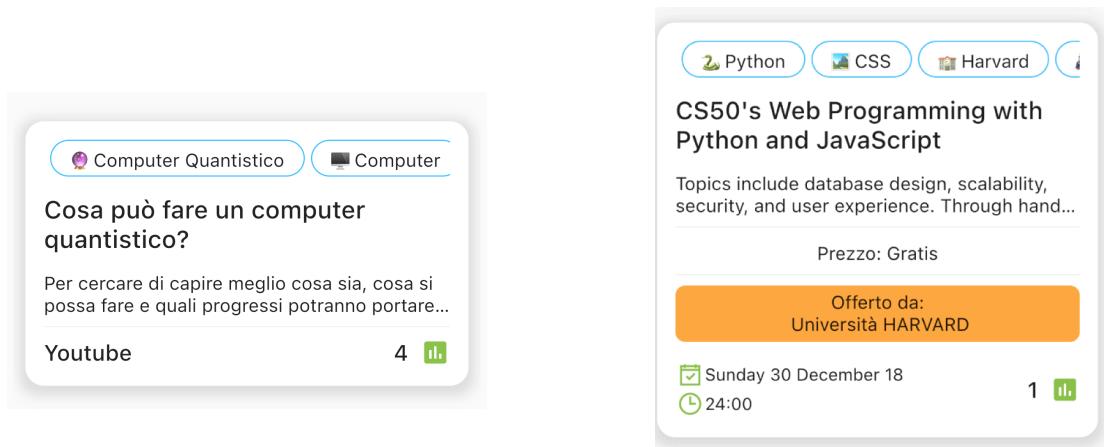
Nella Slider possiamo notare oltre ai live e offline, tips e preferiti.

```
final Map<int, Widget> myTabs = const <int, Widget>{
  0: Text(" Live "),
  1: Text(" Offline "),
  2: Text(" Tips "),
  3: Text("Preferiti"),
};
```

- **Tips:** I tips sono gli eventi più aperti dagli utenti, questa implementazione viene fatta tramite una chiamata al documento che incrementa il dato relativo al conteggio. Questo permette di creare statistiche sulle preferenze degli utenti;
- **Preferiti:** i preferiti sono tutti gli eventi che l'utente si salva per poi riguardarli in futuro.

Visualizzazione degli eventi nella Listview

Per mostrare gli eventi utilizzo una Listview dove per ogni elemento vado a creare un container contenente le informazioni principali.



- **ListView:** Un elenco scorrevole di widget disposti linearmente. Listview è il widget di scorrimento più comunemente usato. Visualizza i suoi figli uno dopo l'altro nella direzione di scorrimento. Nell'asse trasversale, i children sono tenuti a riempire il Listview. Il costruttore Listview.builder prende un Indexedwidgetbuilder, che costruisce i children su richiesta. Questo costruttore è adatto per le visualizzazioni

di liste con un numero grande (o infinito) di children perché il costruttore è chiamato solo per quei children che sono effettivamente visibili.

La funzione crea un Row contenente tutti i tags che può essere scrollata orizzontalmente, subito sotto abbiamo il titolo e una breve presentazione. Può esserci anche la presenza di altri elementi che richiamano subito a particolari informazioni da sapere.

Infine ci può essere il sito o provider dove si svolge l'evento, la data e l'orario e il numero di quante persone sono state interessante a quel particolare evento.

```
Widget DataConvertitore(Timestamp data) {
  DateTime d = data.toDate();
  String formatDate = DateFormat('EEEE dd MMMM yy').format(d);
  String formatOra = DateFormat('kk:mm').format(d);
  return Column(
    mainAxisAlignment: MainAxisAlignment.start,
    children: [
      Row(children: <Widget>[
        Icon(
          LineIcons.calendarCheck,
          color: Colors.lightGreen,
        ), // Icon
        Text(formatDate)
      ], // <Widget>[], Row
      Row(
        children: <Widget>[
          Icon(
            LineIcons.clock,
            color: Colors.lightGreen,
          ), // Icon
          Text(formatOra)
        ], // <Widget>[]
      ) // Row
    ],
  );
}
```

Per la gestione della data e ora sono ricorso a delle funzioni che mi hanno permesso di modificarne il formato e adattarlo così a una visione più semplice. Questa funzione in più va a creare il layout inserendo le icone con le date in due differenti righe.

Sincronizzazione dei dati con il database

Per sincronizzare i dati nella applicazione con gli eventi presenti nel database uso uno stream, che permette di leggere i dati in Realtime. Una grande comodità di usare il plugin FlutterFire è che gli ultimi dati scaricati sono disponibili anche se il dispositivo va in offline. Firestore fornisce il supporto per le funzionalità offline. Durante la lettura e la scrittura dei dati, Firestore utilizza un database locale che si sincronizza automaticamente con il server. La funzionalità Cloud Firestore continua quando gli utenti sono offline e gestisce automaticamente la migrazione dei dati quando riacquistano la connettività.

Innanzitutto uso una funziona che si basa sulla sezione in cui l'utente si trova:

Innanzitutto scarico i tags preferiti dell'utente e solo dopo inizio a creare la query per scaricare gli eventi dal database.

```

Stream getData2() {
    List<String> tags = List.from(user["tags_interesse"]);
    switch (segmentedControlGroupValue) {
        case 0:
            if (tags.length <= 10 && user["attiva_filtrri"] && tags.isNotEmpty) {
                return FirebaseFirestore.instance
                    .collection('seminari')
                    .where("live", isEqualTo: true)
                    .where("data_inizio", isGreaterThanOrEqualTo: new DateTime.now())
                    .where('tags', arrayContainsAny: tags)
                    .orderBy("data_inizio")
                    //.limit(20)
                    .snapshots();
            } else {
                return FirebaseFirestore.instance
                    .collection('seminari')
                    .where("live", isEqualTo: true)
                    .where("data_inizio", isGreaterThanOrEqualTo: new DateTime.now())
                    .orderBy("data_inizio")
                    //.limit(70)
                    .snapshots();
            }
            break;
        case 1:
            if (tags.length <= 10 && user["attiva_filtrri"] && tags.isNotEmpty) {
                tags.shuffle();
                return FirebaseFirestore.instance
                    .collection('seminari')
                    .where("Live", isEqualTo: false)
                    .where('tags', arrayContainsAny: tags)
                    //.limit(20)
                    .snapshots();
            }
    }
}

```

I dati vengono scaricati dalle collezioni Firestore. Cloud Firestore è un database flessibile e scalabile per lo sviluppo di dispositivi mobili, web e server da Firebase e Google Cloud. Come Firebase Realtime Database, mantiene i tuoi dati sincronizzati tra le app client tramite listener in tempo reale e offre supporto offline per dispositivi mobili e Web in modo da poter creare app reattive che funzionano indipendentemente dalla latenza di rete o dalla connettività Internet. Cloud Firestore offre anche una perfetta integrazione con altri prodotti Firebase e Google Cloud, incluse Cloud Functions.

Come funziona?

Cloud Firestore è un database NoSQL ospitato nel cloud a cui le tue app iOS, Android e Web possono accedere direttamente tramite SDK nativi. Cloud Firestore è disponibile anche negli SDK nativi Node.js, Java, Python, Unity, C++ e Go, oltre alle API REST e RPC.

Seguendo il modello di dati NoSQL di Cloud Firestore, archivi i dati in documenti che contengono campi mappati a valori. Questi documenti sono archiviati in

raccolte, che sono contenitori per i tuoi documenti che puoi utilizzare per organizzare i dati e creare query. I documenti supportano molti tipi di dati diversi, da semplici stringhe e numeri a oggetti complessi e nidificati. Puoi anche creare sottoraccolte all'interno dei documenti e costruire strutture di dati gerarchiche che si ridimensionano man mano che il database cresce. Il modello di dati Cloud Firestore supporta qualsiasi struttura di dati che funzioni meglio per la tua app.



Inoltre, l'esecuzione di query in Cloud Firestore è espressiva, efficiente e flessibile. Crea query superficiali per recuperare i dati a livello di documento senza dover recuperare l'intera raccolta o eventuali sottoraccolte nidificate. Aggiungi ordinamento, filtri e limiti alle tue query o ai cursori per impaginare i risultati. Per mantenere aggiornati i dati nelle tue app, senza dover recuperare l'intero database ogni volta che si verifica un aggiornamento, aggiungi listener in tempo reale. L'aggiunta di listener in tempo reale alla tua app ti informa con un'istantanea dei dati ogni volta che i dati che le tue app client stanno ascoltando cambiano, recuperando solo le nuove modifiche.

Proteggi l'accesso ai tuoi dati in Cloud Firestore con l'autenticazione Firebase e le regole di sicurezza Cloud Firestore per Android, iOS e JavaScript o Identity and Access Management (IAM) per le lingue lato server.

Controllo di scaricare solo gli eventi con una data valida, cioè che non siano già passati, e che contengano almeno un tag di quelli che piacciono all'utente, infine li ordino da quello più recente.

Per rendere ancora più veloce la ricerca della query sono stati creati degli indici, indicizzare una collezione consente di ridurre drasticamente il numero di documenti esaminati dal DB durante una query, con una conseguente riduzione dei tempi di esecuzione. D'altra parte, la decisione di indicizzare uno o più campi deve essere ponderata, perché l'indicizzazione ha comunque l'effetto di occupare spazio su disco e rallenta l'inserimento e la modifica dei documenti.

Un indice composito memorizza una mappatura ordinata di tutti i documenti di una collezione, basata su una lista ordinata di campi da indicizzare. Firestore utilizza indici composti per supportare query non ancora supportate da indici a campo singolo. Firestore non crea automaticamente indici composti come per gli indici a campo singolo a causa del gran numero di possibili combinazioni di campo.

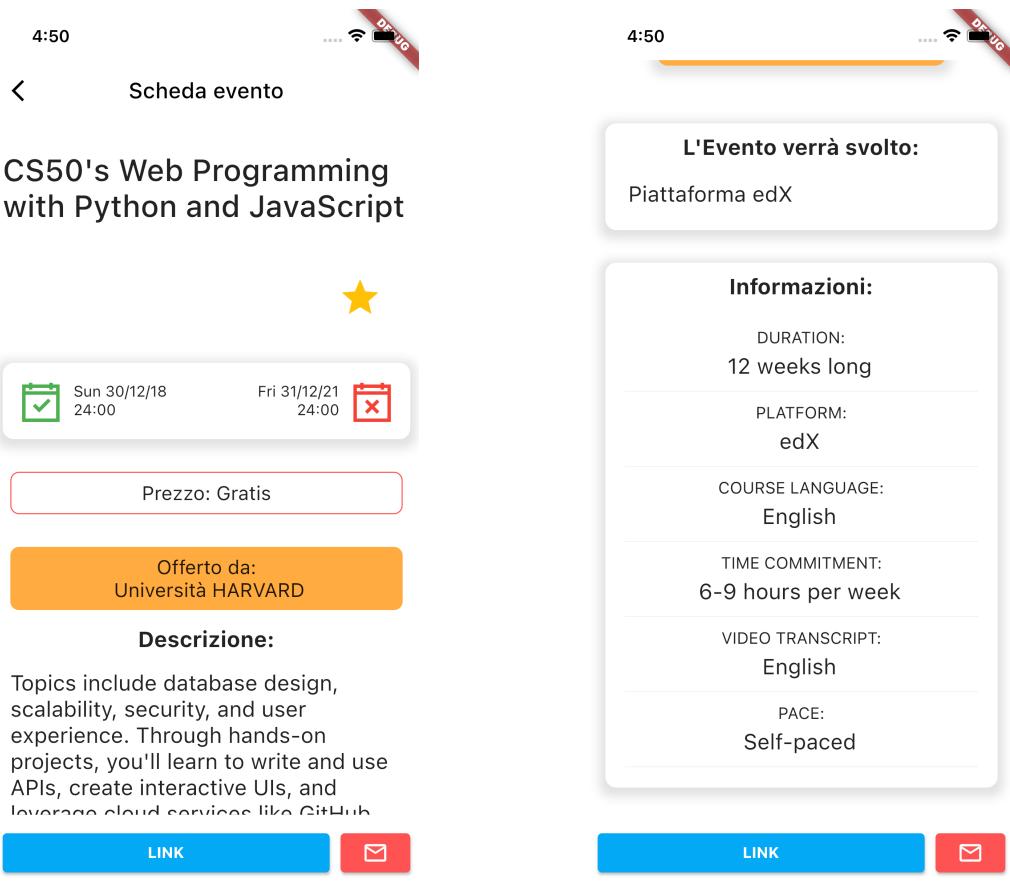
I miei indici composti creati sono:

Indici composti [+ CREA INDICE](#)

Filtro Filter

ID raccolta	Campi	Query scope
competizioni	live: ASC data_fine: ASC	Collection
competizioni	tags: ARRAY live: ASC data_fine: ASC	Collection
corsi	live: ASC data_inizio: ASC	Collection
corsi	tags: ARRAY live: ASC data_inizio: ASC	Collection
seminari	live: ASC data_inizio: ASC	Collection
seminari	tags: ARRAY live: ASC data_inizio: ASC	Collection
social	live: ASC data_inizio: ASC	Collection
social	tags: ARRAY live: ASC data_inizio: ASC	Collection
sportivi	live: ASC data_inizio: ASC	Collection
sportivi	tags: ARRAY live: ASC data_inizio: ASC	Collection

Pagina dell'evento



La pagina singola della applicazione è costruita in modo da mostrare le giuste informazioni anche se due eventi appartengono a due categorie diverse.

Innanzitutto viene mostrato il titolo con sotto il pulsante che aggiunge l'evento ai preferiti dell'utente, se è già presente la stella sarà gialla, in caso contrario rimarrà grigia.

Poi seguono una serie di informazioni che devono subito risultare alla persona interessata, come il prezzo o l'ente proprietario dell'evento.

È presente anche un pulsante “link” che ci porta direttamente alla pagina web completa dell'evento. Anche questa parte è stata realizzata con le funzioni che ci mette a disposizione il pacchetto “url_launcher”.

Abbiamo deciso di tenerla più pulita possibile per essere il più semplice possibile, avere tutte le informazioni a portata di mano e decidere se è evento giusto a cui partecipare.

Test della applicazione

Nello sviluppo di un'applicazione software il testing è un'attività tanto complessa quanto imprescindibile. In genere si compone di più fasi: si inizia con la progettazione dei casi di test, scegliendo particolari parametri di input del sistema e stimando quali saranno i relativi output nel caso in cui il sistema operi secondo le sue specifiche; infine si termina con la valutazione dei risultati ottenuti. Il test avrà successo se il suo obiettivo sarà stato raggiunto: rivelare la presenza di malfunzionamenti nei sistemi. Ma, comunque, nulla garantirà dalla loro assenza.

Non esiste un momento temporale preciso nel progetto di un'applicazione in cui ha inizio il test: l'approccio migliore suggerirebbe di testare il software ad ogni fase dello sviluppo, per collaudare passo dopo passo il comportamento del sistema. Esistono varie strategie di testing: ciascuna di esse agisce su un particolare aspetto.

Tra gli aspetti di un sistema più consistenti ci sono:

1. L'analisi delle singole componenti e della loro integrazione: si realizza mediante test di unità e test di interfaccia;
2. La conoscenza delle funzionalità interne: si realizza mediante test "white box" oppure "black box";
3. La valutazione delle prestazioni: mediante test prestazionale e test di stress;

Strategie per le attività di test

In generale, in un sistema molto complesso sarà difficile (se non impossibile) testare tutte le sue componenti contemporaneamente. Se anche fosse fattibile, probabilmente si manifesterebbero parecchie difficoltà nell'individuare della sorgente dell'errore. Per questo nasce il test più semplice, il test delle unità, che prevede l'analisi di una singola unità del programma per volta. Un'unità può rappresentare un isolato programma, una funzione, una procedura, o anche un modulo. Quando il sistema è composto da diversi oggetti interagenti, si accede alle loro funzionalità attraverso la definizione di un'interfaccia. Effettuare un test su tale interfaccia è fondamentale per verificare il corretto funzionamento dell'intero sistema: gli errori riscontrabili nell'uso di un'interfaccia potrebbero essere molteplici poiché legati alla collaborazione fra elementi diversi, ecc. Dunque, un caso particolare di test delle unità è il test delle interfacce.

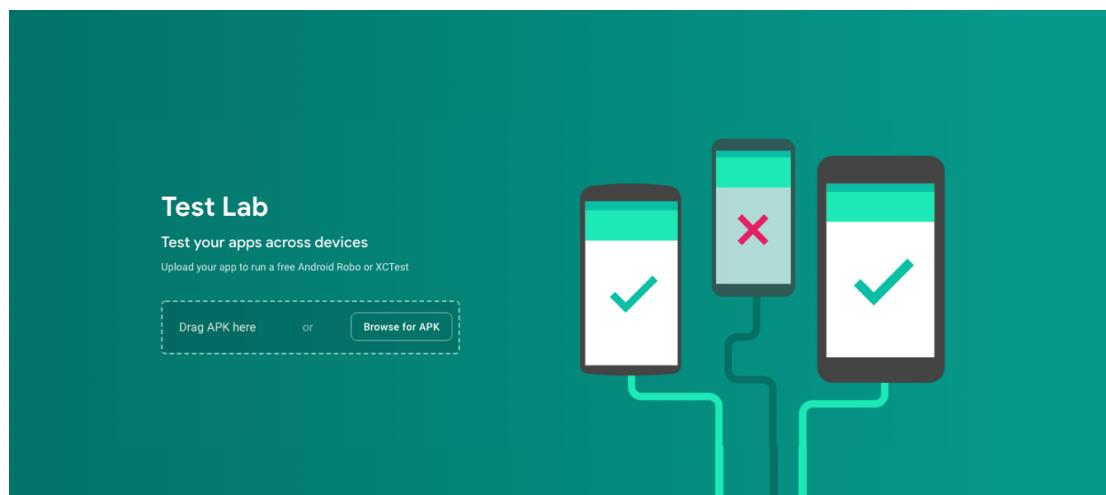
Per test strutturale o "white box" si intendono quei casi in cui è nota la struttura interna del programma (moduli, funzioni, procedure, ecc.) e quindi si cerca di scoprire gli errori logici commessi dal programmatore.

Nel test funzionale o "black box" non si ha la conoscenza di come il sistema sia strutturato; esso è visto come una scatola nera. Il test verrà progettato avendo presenti soltanto le specifiche del sistema.

Il test prestazionale si occupa di verificare che il sistema possa supportare una ingente quantità di carico di lavoro. In questo tipo di test, si tende ad aumentare via via il carico, fino a che le prestazioni del sistema diventano intollerabili.

Un'ulteriore strategia di test è il test di regressione, che ha senso effettuare nel momento in cui si apportano modifiche al progetto iniziale. Il termine “regressione” spiega bene il concetto base di questo approccio: verificare se, a seguito di una qualche modifica nel programma, la qualità del software sia regredita, cioè se siano stati introdotti dei difetti non presenti nella versione precedente alla modifica. La strategia consiste nel riapplicare al software modificato i test progettati per la sua versione originale e confrontare i risultati.

Per la mia sezione di testing ho utilizzato un metodo messo a disposizione da Firebase, che permette di rilasciare la applicazione direttamente ai tester. Firebase Test Lab è un'infrastruttura di test delle app basata su cloud che ti consente di testare la tua app su una vasta gamma di dispositivi e configurazioni, così puoi avere un'idea migliore di come funzionerà nelle mani degli utenti live. Test Lab utilizza dispositivi di produzione reali in esecuzione in un data center di Google per testare la tua app. I dispositivi sono dotati di API aggiornate e hanno impostazioni locali personalizzabili, che ti consentono di testare su strada la tua app sull'hardware e sulle configurazioni che incontrerà nell'uso reale.



Questo mi ha permesso di condividere le varie release della applicazione in un modo estremamente facile e, al qual tempo, una raccolta veloce dei dati per poter poi migliorare la applicazione.

Team e traguardi raggiunti

Per buona parte di questa tesi ho usato il plurale e lo ho fatto perché non c'ero solo io dentro a questo progetto, infatti a questa startup ci lavoriamo principalmente io e un mio amico (io ho sviluppato un primo MVP e Vittorio Mazzacani si è occupato di tutta la parte riguardante l'organizzazione). Di questo team ha fatto parte anche Davide Carletti che non ha lavorato direttamente alla applicazione ma ogni tanto ci incontravamo per discuterne e trovare buone soluzioni.

Perché è fondamentale avere un buon team?

La composizione del team è di vitale importanza per una startup, poichè rappresenta una delle componenti che ne condiziona il percorso di crescita. In questo senso, uno studio condotto nel 2020 da CB Insights spiega come nel 23% dei casi la causa del fallimento sia legata a un management inadatto, poco coeso e altrettanto poco motivato. Mentre l'esperienza aiuta le persone a identificare opportunità e permette una visione ad ampio raggio, un team ha anche bisogno di competenze trasversali e di una condivisione dei piani futuri per il progresso di un'impresa. Si può così affermare come la crescita di una startup sia strettamente correlata alle scelte e strategie adottate dai membri che la gestiscono. Le caratteristiche e le abilità dei componenti del team, tra l'altro, vengono particolarmente soppesate dai finanziatori, i quali tengono a considerare il valore del gruppo uno degli elementi cruciali nella valutazione del progetto.

Un punto fondamentale per il progetto è stato quello di trovare università disposte a permetterci di condividere i loro eventi nella applicazione e condividerle con molti più studenti. Per farlo abbiamo mandato questa email:

“Salve,

Mi chiamo Marco Incerti, sono uno studente all'ultimo anno di Informatica presso UNIMORE e sto svolgendo la mia tesi assieme al professor Paolo Burgio.

Il mio progetto consiste in una piattaforma sulla quale raccogliere tutti gli eventi legati al mondo universitario (quali seminari, corsi extra-curricolari e competizioni sportive e non) e renderli accessibili agli studenti, in modo da raggiungere un maggior numero di interessati.

Vorrei, dunque, chiedere il vostro permesso al fine di pubblicare eventi riguardanti anche il vostro ateneo, in modo da sostenere lo sviluppo della mia tesi.

Ovviamente prima di contattarvi, io e il professor Burgio abbiamo chiesto le stesse autorizzazioni ad UNIMORE, che ha da subito aderito per una collaborazione: tuttavia, al fine di raggiungere un numero sempre crescente di studenti, l'obiettivo della piattaforma è quello di rendere accessibili gli eventi di tutte le università del

territorio (per l'organizzatore sarà poi possibile limitare gli eventi ai soli studenti di determinati corsi o atenei).

In caso di vostra risposta affermativa, cominceremo al più presto ad aggiungere i vostri corsi e seminari assieme a quelli di UNIMORE e degli altri atenei cooperanti.”

Oltre alla email abbiamo allegato una presentazione corta e concisa per spiegare al meglio la nostra idea.

I destinatari sono state 15 prestigiose università italiane con ben 14 risposte affermative, tra cui l'UNIMORE.

Vedere una partecipazione da parte di enti così grandi è stato un gran passo e non vediamo l'ora di iniziare a pubblicare tutti gli eventi sperando che gli studenti apprezzino la possibilità di poter accrescere i loro interessi.

In cantiere abbiamo anche in mente di collaborare con alcune pagine Instagram che hanno già dato il loro consenso. Il punto è che non stiamo parlando di semplici pagine ma pagine con migliaia e migliaia di followers.

Conclusioni

Per me è stata una esperienza ricca di spunti, dove per la prima volta sono partito da una semplice idea fino ad arrivare a un prodotto funzionante curando anche il contorno del progetto.

Ho iniziato a parlarne con il prof Paolo Burgio, che ringrazio tantissimo, quando era ancora solo una cosa dentro la mia testa, quindi ho dovuto trovare e studiare tutte le possibili soluzioni che sono disponibili al giorno d'oggi. Per la prima volta ho capito quanto sia fondamentale il design in una app perché gli utenti decidono in 20 secondi se continuare ad usare quella applicazione basandosi solo sul primo approccio.

Neanche i colori sono a caso in questa applicazione, infatti ho scelto dei colori complementari perché ci sono degli studi che dicono che rendano le applicazioni più belle.

Ho dovuto creare un sistema per gestire gli eventi, un metodo per decidere quali mostrare ai vari utenti, un soluzione per monitorare la applicazione e crearla!

Penso di averla progettata non so quante volte e di averla modificata in corso d'opera ancora più spesso, cambiando la AppBar o la forma delle celle che vengono mostrate nella homepage.

Con il mio team siamo riusciti ad avere l'approvazione da 14 prestigiose università potendo inserire già molti eventi, siamo contenti di questo risultato ma puntiamo ancora più in alto.

Ad Agosto metterò a punto tutte le piccolezze che ci sono rimaste per perfezionare la applicazione e, se tutto va bene a inizio settembre uscirà. Oltre a pubblicare ufficialmente la applicazione in questo periodo si unirà al Team altre due persone.

Ringrazio di nuovo il professore Paolo Burgio e confido in me e nel mio team per la buon riuscita di questo progetto e per rendere (per ora) questa Italia un paese con uno strumento in più per poter coltivare le proprie passioni.

Questo progetto lo dedico a mia nonna che una volta mi disse: **IMPARA L'ARTE E METTILA PARTE.**