Sommario

Il progetto di tirocinio presentato in questa tesi consiste nello sviluppo di un'applicazione mobile multipiattaforma tramite Flutter, un framework moderno e altamente performante per la creazione di interfacce utente native.

L'obiettivo è la realizzazione di una piattaforma SaaS (Software as a Service) per il monitoraggio di ambienti di produzione digitali, in particolare servizi backend e API. Ogni utente ha la possibilità di configurare dei "monitor", associati a uno o più progetti, per controllare lo stato e l'affidabilità delle risorse attraverso le risposte HTTP (codici 2xx, 4xx, 5xx, ecc.).

L'applicazione permette di rilevare tempestivamente anomalie o interruzioni nei servizi, notificando in tempo reale eventuali errori e consentendo così una rapida risposta. Questo sistema si rivela particolarmente utile per agenzie e aziende che gestiscono infrastrutture digitali come siti web, database o microservizi.

Dal punto di vista commerciale, la piattaforma può essere monetizzata tramite un sistema di abbonamento mensile, con funzionalità scalabili in base al numero di progetti o monitor attivi. Questo approccio garantisce flessibilità sia per piccoli team sia per aziende strutturate, offrendo un valore concreto in termini di affidabilità e qualità del servizio.

Indice

1 Introduzione

1.1 Contesto e motivazione

Negli ultimi anni, il crescente utilizzo di servizi digitali, API e applicazioni web ha reso fondamentale per aziende e sviluppatori disporre di strumenti affidabili per il monitoraggio e la gestione degli ambienti di produzione.
Errori non rilevati, interruzioni di servizio o cali di prestazioni possono avere un impatto significativo sull'esperienza utente e sulla reputazione dei fornitori di servizi digitali.

Il progetto sviluppato durante il tirocinio nasce da questa esigenza concreta: realizzare una piattaforma SaaS (Software as a Service) in grado di monitorare in tempo reale lo stato di API e servizi web, notificando eventuali anomalie o malfunzionamenti.
Il sistema consente agli utenti di configurare e gestire autonomamente i propri monitor, associati a specifici progetti, e di visualizzare tramite un'interfaccia mobile intuitiva lo stato delle proprie risorse.

1.2 Obiettivo

L'obiettivo della tesi è lo sviluppo del frontend di un'applicazione mobile multipiattaforma in Flutter, seguendo i principi di scalabilità, manutenibilità e modularità tipici della Clean Architecture.
Il progetto prevede l'adozione delle best practice in ambito di progettazione software, con separazione netta tra logica e presentazione, uso di pattern architetturali, gestione dello stato reattiva e una particolare attenzione all'esperienza utente.
L'intero sistema è pensato per essere facilmente estendibile e in grado di rispettare sia i requisiti funzionali definiti dal committente, sia requisiti non funzionali quali performance, responsività e sicurezza.

1.3 Stato dell'arte

Lo sviluppo di applicazioni accessibili su dispositivi con diversi sistemi operativi è oggi uno dei principali obiettivi del mondo IT.
La necessità di raggiungere un'utenza sempre più diversificata, che utilizza dispositivi Android, iOS, Web o Desktop, ha spinto l'evoluzione di tecnologie in grado di unificare lo sviluppo software.

1.3.1 Framework multipiattaforma

Negli ultimi anni, sono emersi framework multipiattaforma che permettono di scrivere una singola codebase ed eseguirla su più piattaforme. I più diffusi sono **React Native**, sviluppato da Facebook, e **Flutter**, sviluppato da Google.

- React Native è basato su JavaScript e utilizza componenti nativi sotto il cofano. Permette di integrare facilmente codice nativo (Java/Swift) dove necessario e ha il vantaggio di una community molto ampia. Tuttavia, richiede spesso l'uso di bridge per comunicare tra il codice JavaScript e i componenti nativi, il che può causare rallentamenti o complessità.
- Flutter, invece, è basato su Dart e non si appoggia a componenti nativi, ma disegna ogni elemento della UI tramite il proprio motore grafico (Skia). Questo approccio garantisce maggiore coerenza visiva e controllo sull'interfaccia, oltre a performance competitive, ma richiede l'apprendimento di un linguaggio meno diffuso (Dart) e un ecosistema leggermente più giovane.

Entrambe le soluzioni rappresentano validi strumenti per la creazione di applicazioni moderne, ma Flutter viene spesso preferito per progetti in cui l'esperienza utente, la reattività e il controllo sul design sono prioritari.

1.3.2 Vantaggi e svantaggi

VANTAGGI:

- **Accessibilità:** una singola base di codice consente di distribuire l'app su più sistemi operativi (Android, iOS, Web, Desktop).
- **Riduzione dei costi:** sviluppare un'unica app invece di due native (iOS e Android) comporta un notevole risparmio economico.
- **Coerenza dell'interfaccia:** lo sviluppo centralizzato permette di mantenere un design uniforme su tutti i dispositivi.
- **Manutenzione semplificata:** aggiornamenti e fix vengono applicati una volta sola, evitando disallineamenti tra versioni.
- **Time-to-market ridotto:** si riduce il tempo complessivo necessario per ideare, sviluppare e distribuire l'app.
- **Community e strumenti:** framework come Flutter e React Native dispongono di plugin, librerie e comunità attive a supporto dello sviluppo.

SVANTAGGI:

- **Performance inferiori rispetto al nativo:** in app che richiedono uso intensivo di animazioni, grafica 3D o accesso a sensori avanzati, il nativo rimane ancora superiore.
- **Limitazioni nell'accesso alle API native:** anche se superabili con plugin o codice personalizzato, alcune funzionalità avanzate non sono sempre immediatamente disponibili.
- **Dipendenze da plugin esterni:** alcune funzionalità avanzate richiedono plugin, che possono essere non sempre aggiornati o ben mantenuti.

- **Dimensioni dell'app maggiori:** il motore di rendering integrato o le dipendenze multiple aumentano il peso complessivo dell'app.
- **Aggiornamenti asincroni:** cambiamenti nei sistemi operativi o SDK possono introdurre incompatibilità che richiedono interventi urgenti.
- **Necessità di download esplicito:** a differenza delle web app, le app multipiattaforma devono essere installate tramite uno store.

1.4 Descrizione UpApi

UpApi è una piattaforma SaaS (Software as a Service) progettata per il monitoraggio continuo e affidabile di ambienti di produzione digitali. Il sistema consente agli utenti di registrare un account personale, attivato tramite una verifica email, e di configurare autonomamente i propri ambienti da monitorare. Ogni utente può creare uno o più progetti, ciascuno dei quali rappresenta un ambiente digitale distinto (ad esempio, un sito web come amazon.it). All'interno di ogni progetto è possibile definire e gestire molteplici monitor, ciascuno dedicato al controllo di una funzionalità o risorsa specifica. I monitor eseguono controlli periodici, con cadenza personalizzabile, e restituiscono codici di stato HTTP (2xx, 3xx, 4xx, 5xx) per indicare lo stato del servizio monitorato. Oltre allo stato attuale, la piattaforma conserva uno storico completo degli esiti dei controlli effettuati, consentendo una rapida analisi delle prestazioni nel tempo. È inoltre prevista l'integrazione con un sistema di notifiche, che avvisa tempestivamente l'utente in caso di anomalie o interruzioni. Le notifiche possono essere personalizzate in base al tipo di errore o al monitor specifico. La piattaforma include anche funzionalità standard per la gestione del profilo utente, tra cui il recupero della password, la modifica dei dati personali e la gestione delle credenziali.

1.5 Scelte tecnologiche

Le tecnologie adottate sono state selezionate per costruire una soluzione moderna, stabile e performante, con particolare attenzione alla scalabilità, al supporto multi-piattaforma e alla qualità del codice.
L'obiettivo era garantire uno sviluppo agile e sostenibile nel tempo, utilizzando strumenti collaudati e tecnologie in continua evoluzione.

Di seguito le principali tecnologie adottate:

- **Flutter (Dart):** Il framework principale scelto per lo sviluppo frontend è Flutter, open-source e sviluppato da Google. Consente di creare applicazioni native Android e iOS da un'unica base di codice, riducendo tempi e costi di sviluppo. Il linguaggio utilizzato è Dart, noto per la sintassi pulita e le buone performance lato mobile.
- **Flutter Material:** È il pacchetto integrato in Flutter che implementa le linee guida di Material Design, offrendo componenti UI predefiniti come pulsanti, schede, app bar e

layout responsivi. L'uso di Flutter Material ha facilitato lo sviluppo rapido di un'interfaccia moderna, coerente e accessibile, riducendo la necessità di progettazione personalizzata da zero e mantenendo uno standard visivo professionale e riconoscibile su tutte le piattaforme.

- **Android Studio:** Come ambiente di sviluppo (IDE) è stato utilizzato Android Studio, che offre un'integrazione nativa con Flutter e strumenti avanzati per debugging, emulazione e profilazione delle performance.
- **Cubit (Flutter Bloc):** Per la gestione dello stato si è optato per Cubit, una soluzione più snella rispetto al pattern Bloc completo. Permette di separare la logica di business dalla UI, migliorando la manutenibilità e scalabilità del codice. Cubit ci permette di rispettare la clean architetture, che è considerata standard in progetti di qualità per flutter.
- **Node.js:** Il backend dell'applicazione (gestito dal team) è stato sviluppato in Node.js, utilizzato per esporre API RESTful con performance elevate e gestione asincrona delle richieste. È stato integrato tramite chiamate HTTP dal client Flutter.
- **Git + Bitbucket:** Il versionamento del codice è stato gestito tramite Git, con repository ospitata su Bitbucket. Questo ha garantito tracciabilità, organizzazione del lavoro in branch e possibilità di revisione collaborativa.
- **Dio:** Libreria per la gestione delle chiamate HTTP in Flutter. Offre strumenti avanzati come interceptor, configurazione di timeout, gestione centralizzata degli errori e supporto alla serializzazione dei dati.
- **GoRouter:** Utilizzata per la gestione della navigazione in Flutter. Consente routing dichiarativo, gestione avanzata di deep linking, redirect condizionati e passaggio di parametri tra schermate.
- **Flutter Localization:** Sistema integrato per la gestione della localizzazione e delle traduzioni. Ha permesso di predisporre l'applicazione per il supporto multilingua, rendendola pronta per la distribuzione in contesti internazionali.
- **Postman:** Strumento utilizzato per il testing delle API. Ha facilitato la verifica del corretto funzionamento delle interfacce REST, permettendo di organizzare richieste e simulare scenari reali durante lo sviluppo.

Flutter si è rivelato particolarmente adatto per garantire coerenza e qualità su più dispositivi, riducendo la complessità dello sviluppo. L'utilizzo di Cubit ha permesso di mantenere una chiara separazione tra logica di business e presentazione, semplificando la manutenzione futura. Inoltre, l'adozione di strumenti come Dio, GoRouter e Flutter Localization ha permesso di costruire un'app completa, modulare e pronta per una distribuzione internazionale. L'approccio orientato alla modularità e alla pulizia del codice si riflette nell'intero ciclo di sviluppo, in linea con i principi dell'ingegneria del software.

2 Analisi preliminare

3 Progettazione

4 Descrizione implementazione/sviluppo delle varie parti

5 Problematiche

6 Risultati ottenuti

7 Conclusioni e sviluppi futuri