

**Progetto di Griglie Computazionali e Sistemi di Elaborazione Ubiqui**



A.A. 2024 -2025

Docente: Studentessa:

**Prof. Domenico Talia Dott.ssa Antonella Rizzo**

**matr. 122086**

Sommario

[1. INTRODUZIONE 3](#_Toc187083646)

[2. DESCRIZIONE DEL PROGETTO 5](#_Toc187083647)

[3. ANDROID 6](#_Toc187083648)

[SPRING BOOT 10](#_Toc187083649)

[4. BOOT DASHBOARD 12](#_Toc187083650)

[5. MAVEN 14](#_Toc187083651)

[6. PostgreSQL 17](#_Toc187083652)

[7. POSTMAN 18](#_Toc187083653)

[8. EVALUATE PERFORMANCE APP 21](#_Toc187083654)

[9. EVALUATE PERFORMANCE APP ANDROID 22](#_Toc187083655)

[10. CONCLUSIONI 23](#_Toc187083656)

# INTRODUZIONE

L'idea nasce dall’esigenza di rispondere alla domanda:"Volendo spendere tra gli x€ e gli y€, quale dispositivo mi converrebbe acquistare? Qual è il più performante?"

E’ stata sviluppata un’applicazione android che confronta, per fascia di prezzo, le performance del dispositivo su cui è installata, con altri dispositivi nella rete che hanno eseguito il medesimo test.

L’applicazione una volta installata consentire all’utente di valutare le performance del proprio dispositivo fornendo un punteggio complessivo in base ai seguenti parametri,

\* memoria (RAM), in termini di utilizzo della memoria;

\* tempo (CPU), in termini di velocità di esecuzione;

\* batteria, in termini di consumo di batteria.

Dopo aver eseguito il test, è possibile confrontare il punteggio del proprio dispositivo con quello di altri dispositivi presenti nel database, che include anche la suddivisione per fasce di prezzo.

Una volta eseguito il test di performance, l’utente avrà la possibilità di inviare i relativi risultati ad un web server che ne effettua il salvataggio su un db in remoto. L’operazione di invio è corredata dall’informazione sulla fascia di prezzo (selezionata tra le n disponibili), il kpi di performnce e il modello del dispositivo.

L’utente ha inoltre la possibilità di visualizzare la classifica generale dei dispositivi in base ad uno o più dei parametri di valutazione.

Riassumendo, l’applicazione espone i seguenti servizi:

\* valutazione performance dispositivo (servizio in locale). Il servizio è caratterizzato da un unico parametro obbligatorio (selezionato dall’utente tra gli n disponibili):

* identificativo kpi (ossia algoritmo di valutazione).

\* salvataggio su db remoto dei risultati di valutazione (invocazione web server). La chiamata a servizio è caratterizzata dai seguenti parametri obbligatori:

* array di risultati;
* fascia di prezzo;
* modello dispositivo;
* identificativo kpi.

Nel caso in cui venissero effettuati due o più invii dei risultati per il medesimo dispositivo, verrebbero persistiti i risultati più recenti.

\* visualizzazione classifica generale dei risultati, per fascia di prezzo, per parametro di valutazione e identificativo test. (invocazione web server). La chiamata a servizio è caratterizzata dai seguenti parametri:

* fascia di prezzo (parametro obbligatorio);
* identificativo test (parametro obbligatorio).

# DESCRIZIONE DEL PROGETTO

Si consideri un sistema costituito da un insieme di dispositivi mobili. L’insieme dei dispositivi forma una rete nella quale i nodi comunicano tra di loro attraverso la rete internet.

Ogni dispositivo funge da ***client*** edesegue algoritmi di valutazione performance in modo locale o remoto. L’esecuzione remota consiste nell’invocazione di un servizio pubblicato su un dispositivo server. Nel caso di esecuzione locale eventuali dataset sono presenti sul dispositivo client.

Il sistema operativo installato sui dispositivi mobili è ***Android***.

L’applicazione *EvaluatePerformance* è sviluppata come progetto ***Maven***.

Sui dispositivi deve essere installata l’applicazione android ***DataMiningAndroid*** che consente all’utente di scegliere l’algoritmo che deve essere eseguito per la valutazione della performance desiderata. La scelta dell’algoritmo determina se la modalità di esecuzione sarà da *locale* o da *remoto* su un server predefinito.

# ANDROID

Android è un ambiente software open source che comprende sistema operativo, middleware e applicazioni chiavi. È stato sviluppato da Google per dispositivi mobili. L'SDK di Android fornisce gli strumenti e le API necessarie per iniziare a sviluppare applicazioni per la piattaforma Android utilizzando il linguaggio di programmazione Java. Il codice Java compilato - insieme con tutti i dati e le risorse richieste dall'applicazione - è impacchettato in un archivio di file contrassegnati dal suffisso ***.apk***. Questo file è il veicolo per distribuire l'applicazione e installarlo sui dispositivi mobili: è il file di download degli utenti ai loro dispositivi. Tutto il codice in un unico file .apk è considerato una sola applicazione.

**Architettura di Android**

L’architettura di Android è composta da diversi livelli, ognuno con un ruolo specifico nel funzionamento del sistema operativo.

Il diagramma seguente mostra i componenti principali. Ogni sezione è descritta in dettaglio più avanti.

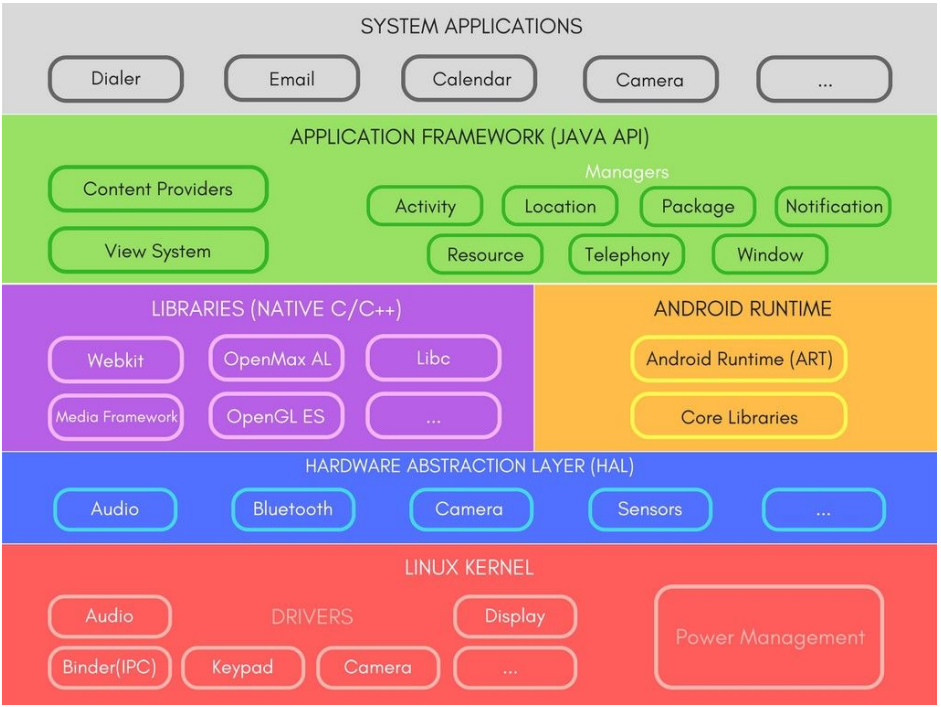


Figura 1: *Architettura Android*

***Kernel Linux***Il kernel Linux è il nucleo del sistema operativo Android. Gestisce tutte le interazioni con l’hardware del dispositivo, inclusi la gestione della memoria, il controllo dei processi, la gestione dei driver e la comunicazione con i dispositivi periferici. Il kernel fornisce anche funzionalità di sicurezza e gestione dell’energia.

Con l’introduzione del progetto Generic Kernel Image (GKI), il kernel di Android è diventato più modulare. Questo significa che il kernel principale è separato dai moduli specifici del fornitore, facilitando gli aggiornamenti e migliorando la compatibilità tra diversi dispositivi.

Essendo open source, il kernel di Linux può essere modificato e personalizzato per adattarsi alle esigenze specifiche dei produttori di dispositivi e degli sviluppatori. Questo ha portato a una vasta gamma di varianti del kernel, ciascuna ottimizzata per specifici hardware e usi.

**Librerie Native**  
Questo livello contiene una serie di librerie scritte in C/C++ che forniscono funzionalità di base per il sistema operativo. Tra queste librerie ci sono:

* *Surface Manager*: Gestisce l’accesso al display e la composizione delle superfici grafiche.
* *Media Framework*: Supporta la riproduzione e la registrazione di vari formati audio e video.
* *SQLite*: Un database leggero utilizzato per l’archiviazione dei dati delle applicazioni.
* *OpenGL ES*: Una libreria grafica per il rendering 2D e 3D.

**Android Runtime (ART)**  
ART è il runtime che esegue le applicazioni Android. Sostituisce il precedente Dalvik Virtual Machine e offre miglioramenti in termini di prestazioni e gestione della memoria. ART utilizza la compilazione ahead-of-time (AOT) per convertire il bytecode delle applicazioni in codice macchina nativo, migliorando così l’efficienza.

ART utilizza il formato Dalvik Executable (.dex) ottimizzato per una bassa occupazione di memoria e gestisce le classi compilate da Java. ART è progettato per migliorare le performance e ridurre i tempi di avvio delle applicazioni rispetto a Dalvik.

Alcuni pacchetti supportati da ART sono:

* java.lang: Supporto per linguaggio ed eccezioni.
* java.io: Input/Output di file e stream.
* java.math: Numeri grandi e precisione.
* java.net: Network I/O, URL, socket.
* java.nio: File e I/O di canali.
* java.security: Autorizzazione, certificati, chiavi pubbliche.
* java.sql: Interfacce per database.
* javax.crypto: Cifratura e chiavi pubbliche.
* javax.net: Socket factories, SSL.
* javax.xml.parsers: Parsing XML.
* org.w3c.dom e org.xml.sax: API per XML.

In aggiunta, l'SDK di Android include librerie di terze parti come:

* org.apache.commons.httpclient: Autenticazione HTTP, cookies, metodi e protocolli.
* org.apache.commons.codec: Utility per codifica e decodifica.
* org.bluez: Supporto per Bluetooth.
* org.json: JavaScript Object Notation.

**Application Framework**  
fornisce una serie di servizi di alto livello che le applicazioni possono utilizzare per interagire con il sistema operativo. Questo livello è fondamentale per lo sviluppo delle applicazioni Android, poiché offre le API necessarie per accedere alle funzionalità di sistema. Alcuni dei componenti principali includono:

* *Activity Manager*  
  Gestisce il ciclo di vita delle applicazioni e la navigazione tra le attività. Assicura che le applicazioni possano passare da uno stato all’altro senza problemi, gestendo la memoria e le risorse in modo efficiente
* *Window Manager*Gestisce le finestre e le loro interazioni con l’utente. Questo componente è responsabile della disposizione delle finestre sullo schermo e della gestione delle transizioni tra le diverse finestre.
* *Content Providers*  
  Permettono alle applicazioni di condividere dati con altre applicazioni. Forniscono un’interfaccia standard per accedere e modificare i dati, come i contatti o i file multimediali.
* *View System*  
  Un insieme di componenti UI che permettono agli sviluppatori di creare interfacce utente complesse. Include vari widget come pulsanti, campi di testo, liste e layout.
* *Package Manager*  
  Fornisce informazioni sulle applicazioni installate sul dispositivo. Permette di installare, aggiornare e rimuovere applicazioni, oltre a gestire i permessi delle applicazioni.
* *Telephony Manager*  
  Gestisce tutte le operazioni legate alla telefonia, come le chiamate, i messaggi SMS e le informazioni sulla rete.
* *Resource Manager*  
  Fornisce accesso alle risorse non codificate, come stringhe, layout e grafica. Questo componente permette di gestire le risorse in modo efficiente, supportando diverse configurazioni di dispositivo e localizzazioni.
* *Notification Manager*  
  Gestisce la visualizzazione delle notifiche. Permette alle applicazioni di mostrare messaggi all’utente, che possono essere visualizzati nella barra delle notifiche.
* *Location Manager*Fornisce servizi di localizzazione, permettendo alle applicazioni di ottenere la posizione geografica del dispositivo. Supporta diverse fonti di localizzazione, come GPS e reti mobili.
* *Sensor Manager*  
  Gestisce l’accesso ai sensori del dispositivo, come accelerometri, giroscopi e sensori di prossimità. Permette alle applicazioni di rilevare e rispondere ai cambiamenti nell’ambiente fisico del dispositivo.

Questi componenti lavorano insieme per fornire un ambiente di sviluppo robusto e flessibile, permettendo agli sviluppatori di creare applicazioni ricche di funzionalità e altamente interattive.

**Applications**  
Il livello Applications è il più alto nell’architettura di Android e rappresenta l’interfaccia con cui gli utenti interagiscono direttamente. Questo livello include tutte le applicazioni preinstallate e quelle scaricate dagli utenti tramite il Google Play Store. I componenti principali di questo livello sono:

* Le applicazioni preinstallate. Android viene fornito con una serie di applicazioni preinstallate che forniscono funzionalità di base. Queste includono:
  + *Dialer,* l’applicazione per effettuare e ricevere chiamate telefoniche.
  + *Contatti*, gestisce la rubrica dei contatti dell’utente.
  + *Browser,* un’applicazione per la navigazione web.
  + *Messaggi*, per inviare e ricevere SMS e MMS.
  + *Fotocamera*, per scattare foto e registrare video.
* Le applicazioni di terze parti. Gli utenti possono scaricare e installare applicazioni di terze parti dal Google Play Store o da altre fonti. Queste applicazioni possono variare ampiamente in termini di funzionalità, dalle app di social media ai giochi, alle applicazioni di produttività.
* Le componenti delle applicazioni. Le applicazioni Android sono costituite da diversi componenti principali, ciascuno con un ruolo specifico:
  + *Activity* che rappresenta una singola schermata con un’interfaccia utente. Ogni applicazione può avere una o più attività.
  + *Service* che è un componente che esegue operazioni in background senza un’interfaccia utente.
  + *Broadcast* *Receiver* che gestisce e risponde agli eventi di sistema o alle trasmissioni di altre applicazioni.
  + *Content Provider* che fornisce un meccanismo per condividere dati tra diverse applicazioni.

Le applicazioni interagiscono con il sistema operativo Android attraverso il framework delle applicazioni. Questo framework fornisce API che permettono alle applicazioni di accedere ai servizi di sistema, come la gestione delle risorse, la gestione delle attività e la gestione delle notifiche.

Quando un utente avvia un’applicazione, il sistema operativo Android crea un processo separato per quell’applicazione e un’istanza della ART. Questo isolamento garantisce che le applicazioni non interferiscano tra loro e migliorano la sicurezza e la stabilità del sistema.

Questa struttura a livelli consente ad Android di essere flessibile, modulare e facilmente aggiornabile, contribuendo alla sua diffusione e popolarità globale.

# SPRING BOOT

Spring Boot è un framework open source che semplifica e accelera il processo di creazione di applicazioni Java basate su Spring. È un’estensione del popolare framework Spring e mira a ridurre la complessità e il tempo necessario per configurare e avviare nuove applicazioni. Spring Boot consente agli sviluppatori di avviare rapidamente applicazioni stand-alone, basate su microservizi o basate su architettura a più livelli, con una configurazione minima.

Spring Boot è progettato per rendere lo sviluppo di applicazioni Spring più semplice ed efficiente. I suoi obiettivi principali includono:

* *Configurazione Automatica*: Spring Boot fornisce configurazioni predefinite e automatiche per molteplici aspetti dell’applicazione, riducendo la necessità di configurazioni manuali.
* *Semplicità di Avvio*: Permette di avviare un’applicazione Spring con pochi comandi, grazie all’inclusione di server embedded come Tomcat, Jetty o Undertow.
* *Indipendenza dalle Dipendenze:* Non richiede la presenza di un server di applicazioni esterno; le applicazioni Spring Boot possono essere eseguite come applicazioni stand-alone.
* *Struttura di Progetto Convenzionale:* Offre una struttura di progetto predefinita che semplifica l’organizzazione del codice e dei file di configurazione.

Un progetto Spring Boot segue una struttura di base che consente una rapida configurazione e avvio. La struttura tipica include:

* src/main/java: Contiene il codice sorgente dell’applicazione.
* src/main/resources: Contiene file di configurazione e risorse dell’applicazione, come file *application.properties* o *application*.*yml* per la configurazione.
* src/test/java Contiene i test dell’applicazione.
* src/test/resources: Contiene risorse specifiche per i test.
* *pom.xml* o *build.gradle*: Il file di configurazione per la gestione delle dipendenze e la build, a seconda che si utilizzi Maven o Gradle.

Il file principale di configurazione di Spring Boot è *application.properties* o *application.yml*, dove si possono definire le configurazioni dell’applicazione come la connessione al database, le proprietà del server, e altre impostazioni specifiche dell’ambiente.

Spring Boot semplifica l'avvio di un'applicazione grazie alla classe principale con l'annotazione `@SpringBootApplication`. Questa annotazione combina tre annotazioni principali: `@Configuration`, `@EnableAutoConfiguration`, e `@ComponentScan`, e avvia il contesto Spring. Un’applicazione Spring Boot può essere avviata con il comando:

bash

mvn spring-boot:run

oppure, se si utilizza Gradle:

bash

./gradlew bootRun

Inoltre, Spring Boot crea un file JAR eseguibile che può essere avviato direttamente con:

bash

java -jar nome-del-file.jar

Uno degli aspetti più potenti di Spring Boot è la configurazione automatica, che riduce la necessità di configurazione manuale e codice boilerplate. Spring Boot analizza le dipendenze nel progetto e configura automaticamente le componenti necessarie, come la connessione ai database, i server web, e i framework di sicurezza, basandosi sulle librerie incluse e sui file di configurazione.

Spring Boot utilizza un concetto di "starter" per semplificare la gestione delle dipendenze. Gli starter sono pacchetti di dipendenze preconfigurati per specifiche funzionalità, come l'accesso al database, la sicurezza o il supporto per web. Ad esempio, lo starter spring-boot-starter-web include tutte le dipendenze necessarie per creare una web application.

Spring Boot si integra facilmente con strumenti di costruzione come Maven e Gradle, facilitando il processo di build e deployment. Le configurazioni per Maven e Gradle sono predefinite e richiedono pochissime modifiche per adattarsi alle esigenze del progetto.

Spring Boot è, inoltre, particolarmente adatto per la creazione di architetture a microservizi grazie alla sua leggerezza e facilità di configurazione. Permette di sviluppare e distribuire microservizi come applicazioni indipendenti e stand-alone, semplificando la gestione e la scalabilità di applicazioni complesse.

Spring Boot è quindi uno strumento potente e versatile per lo sviluppo di applicazioni Java. Offre una configurazione automatica, una gestione delle dipendenze semplificata e un’avvio rapido delle applicazioni. La sua compatibilità con strumenti di costruzione come Maven e Gradle e la sua capacità di supportare architetture a microservizi lo rendono una scelta ideale per molti sviluppatori e team di sviluppo. Utilizzando Spring Boot, è possibile ridurre il tempo di sviluppo e migliorare la qualità e la manutenibilità delle applicazioni Java.

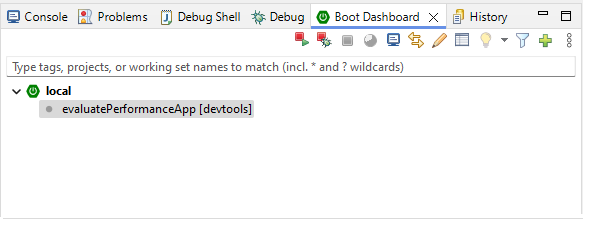
# BOOT DASHBOARD

Il Boot Dashboard è uno strumento essenziale per gli sviluppatori che lavorano con applicazioni Spring. Fornisce una vista unificata per gestire e monitorare le applicazioni Spring Boot. Questo capitolo guida attraverso l'utilizzo, l'avvio, l'arresto, la configurazione e il download di Boot Dashboard.

Per utilizzare Boot Dashboard, è necessario avere un IDE compatibile. Boot Dashboard, infatti, è integrato in ambienti di sviluppo come Spring Tool Suite (STS) ed Eclipse, facilitando lo sviluppo, il debug e la gestione di applicazioni Spring Boot. Offre una visione immediata dello stato delle applicazioni e permette operazioni come l'avvio, l'arresto e il monitoraggio.

Avviare un'applicazione Spring Boot tramite Boot Dashboard è molto semplice. E’ sufficiente seguire i seguenti passi:

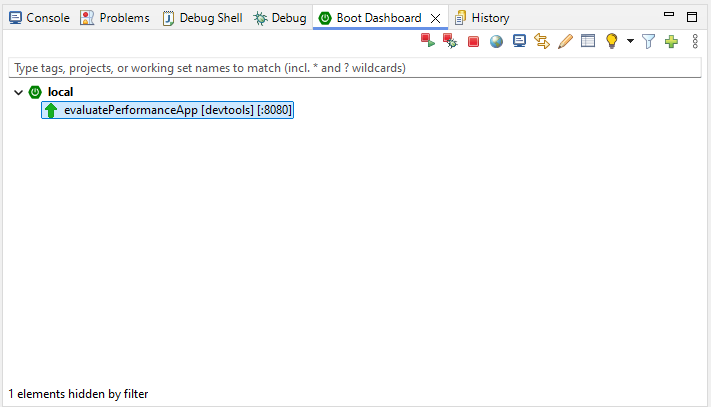
* Aprire Boot Dashboard nell'IDE.
* Selezionare l'applicazione desiderata dall'elenco.
* Cliccare sul pulsante di avvio



Ugualmente semplice è fermare un'applicazione attiva attraverso i seguenti passi:

- Selezionare l'applicazione in esecuzione su Boot Dashboard.

- Cliccare sul pulsante di arresto (solitamente rappresentato con un'icona rossa o un quadrato).



Boot Dashboard è uno strumento potente per la gestione di applicazioni Spring Boot, che semplifica notevolmente lo sviluppo e il monitoraggio. Attraverso il suo utilizzo gli sviluppatori possono migliorare l'efficienza nel lavorare con Spring Boot.

# MAVEN

Maven, principalmente, è uno strumento completo per la gestione di progetti software Java, in termini di compilazione del codice, distribuzione, documentazione e collaborazione del team di sviluppo. Secondo la definizione ufficiale, si tratta di un tentativo di applicare pattern ben collaudati all'infrastruttura del build dei progetti.

Si tratta di promuovere la comprensione e la produttività del team coinvolto nello sviluppo, fornendo un percorso chiaro all'utilizzo di best practice. Per questo motivo Maven è definito, sinteticamente, tool per la gestione e comprensione dei progetti. Maven è quindi contemporaneamente un insieme di standard, una struttura di repository e un'applicazione che servono alla gestione e la descrizione di progetti software. Esso definisce un ciclo di vita standard per il building, il test e il deployment di file di distribuzione Java.

Nel libro "Better Builds With Maven" è presente un'interessante definizione destinata principalmente ai manager: "Maven è uno strumento dichiarativo per la gestione dei progetti Java che permette di ridurre il tempo totale di sviluppo dei progetti (time-to-market) attraverso un efficace utilizzo delle sinergie disponibili. Maven permette di ridurre il numero delle risorse umane e contemporaneamente permette di ottenere elevate efficienze operative".

Le caratteristiche di Maven fanno sì che diverse persone, anche inizialmente estranee al progetto, possano lavorare insieme produttivamente senza dover trascorrere molto tempo per comprendere la struttura del progetto, il suo funzionamento, il processo di build, etc. Tutti coloro che si sono trovati, almeno una volta nella loro vita, a dover intervenire in progetti di grandi dimensioni in corso d'opera, sicuramente conoscono la frustrazione causata da dover capire rapidamente l'interazione delle varie parti del progetto, l'ordine e le dipendenze del processo di build, etc.

Le aree prese in considerazione da Maven sono: build, documentazione, reportistica, gestione delle dipendenze, SCMs (Software Configuration Management), rilascio e distribuzioni di nuove versioni. In effetti tutti i progetti, indipendentemente dal loro dominio, dalla loro estensione e dalla tecnologia impiegata, presentano una serie di necessità standard, quali ad esempio:

* la conversione dei sorgenti in codici "eseguibili" (build)
* la verifica (test)
* l'assemblaggio
* la documentazione
* eventualmente il "dispiegamento" e la relativa configurazione (deployment)

L’unità di lavoro fondamentale in Maven è il Project Object Model o ***POM***. È un file XML che contiene informazioni sul progetto e sui dettagli di configurazione usati da maven per fare il *build* del progetto. Contiene i valori di default per molti progetti, ad esempio la build directory, che è *target*; la source directory, che è *src/main/java*; la test source directory, che è *src/main/test*; e così via.

Il POM contiene le informazioni di configurazione necessarie ed esegue gli obiettivi (*goal*). Alcune delle informazioni di configurazione che possono essere specificate nel POM sono le dipendenze del progetto, i plug-in o i goals che devono essere eseguiti e così via. Altre informazioni che possono essere specificate sono la versione del progetto, la descrizione, gli sviluppatori, le mailing list.

La struttura standard di un *Maven Project* è la seguente:

my-app

|-- pom.xml

`-- src

|-- main

| |-- java

| | `-- com

| | `-- mycompany

| | `-- app

| |

| `-- webapp

| `-- index.html

| `-- WEB-INF

| `-- web.xml

|

`-- test

`-- java

`-- com

`-- mycompany

`-- app

`-- AppTest.java

La directory *src/main/java* contiene il codice java del progetto, la directory *src/main/webapp* contiene le informazioni per le applicazioni web, la directory *src/test/java* contiene il codice per I test dell’applicazione, e il file *pom.xml* è il Project Object Model del progetto. [3]

Maven gestisce il ciclo di vita del progetto attraverso una serie di fasi predefinite:

* *validate*: Verifica se il progetto è valido e pronto per la costruzione.
* *compile*: Compila il codice sorgente del progetto.
* *test*: Esegue i test unitari.
* *package*: Imballa il codice compilato in un formato distribuitivo, come un JAR o WAR.
* *verify*: Esegue ulteriori verifiche sul pacchetto creato, come test di integrazione.
* *install*: Installa il pacchetto nel repository locale, rendendolo disponibile per altri progetti.
* *deploy*: Distribuisce il pacchetto nel repository remoto per la condivisione con altri sviluppatori.

Maven semplifica anche la gestione delle dipendenze utilizzando i repository centrali e locali:

* *Repository Locale*: Una cache locale di artefatti sul computer dell'utente. Di solito, si trova in `~/.m2/repository`.
* *Repository Remoto*: Un repository online che contiene librerie e artefatti condivisi tra progetti. Il repository centrale di Maven è il repository predefinito, ma è possibile configurare repository privati o aggiuntivi.

*Artifactory* è un esempio di repository manager che si integra con Maven per gestire in modo efficace le dipendenze e i pacchetti. Artifactory fornisce una soluzione scalabile e robusta per la gestione degli artefatti, offrendo funzionalità come caching di artefatti, gestione dei permessi e integrazione con strumenti di CI/CD. Utilizzando Artifactory, le organizzazioni possono mantenere un repository interno per le proprie dipendenze e artefatti, migliorando la sicurezza e il controllo sulle librerie utilizzate nei progetti.

Maven utilizza i plugin per estendere ulteriormente le sue capacità. I plugin sono responsabili per le attività specifiche del ciclo di vita del progetto, come compilazione, test e packaging. Ogni plugin può avere uno o più obiettivi, che sono le unità di lavoro che il plugin esegue. Alcuni dei plugin più comuni includono:

* *maven-compiler-plugin*: Compila il codice sorgente Java.
* *maven-surefire-plugin:* Esegue i test unitari.
* *maven-jar-plugin:* Crea file JAR per il pacchetto dell'applicazione.
* *maven-deploy-plugin:* Gestisce la distribuzione degli artefatti.

Inoltre, Maven consente la definizione di profili di build per gestire configurazioni diverse per ambienti di sviluppo, test e produzione. I profili possono modificare le dipendenze, le proprietà e le configurazioni dei plugin in base alle esigenze specifiche di un ambiente. I profili sono definiti nel file pom.xml e possono essere attivati manualmente o automaticamente in base a condizioni specifiche.

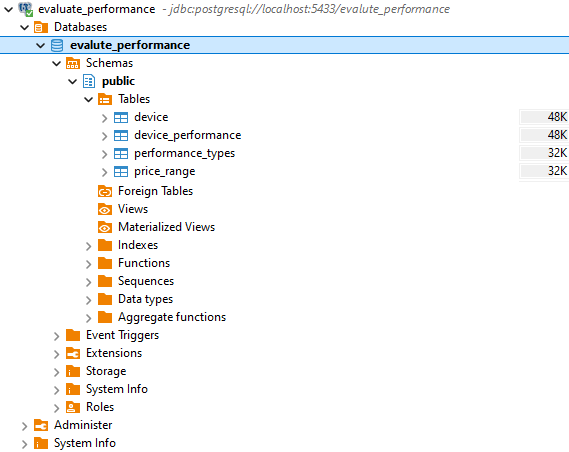
Maven è uno strumento potente e flessibile per la gestione dei progetti Java, che offre automazione, standardizzazione e facilità nella gestione delle dipendenze. La sua struttura basata su un ciclo di vita predefinito e una configurazione centralizzata tramite *pom.xml* lo rende uno strumento essenziale per molti sviluppatori e team di sviluppo. L'integrazione con strumenti come *Artifactory* migliora ulteriormente la gestione degli artefatti e delle dipendenze, offrendo soluzioni scalabili e sicure per le esigenze moderne di sviluppo software.

# PostgreSQL

PostgreSQL è un potente sistema di gestione di database relazionali open source ampiamente utilizzato. Offre funzionalità avanzate come transazioni ACID, stored procedure, trigger, viste materializzate e molto altro. È altamente scalabile e supporta una vasta gamma di tipi di dati. PostgreSQL è noto per la sua affidabilità, flessibilità e conformità agli standard SQL. È utilizzato in una vasta gamma di applicazioni, dalle piccole app ai sistemi aziendali complessi.

Per iniziare, è necessario installare PostgreSQL e seguire le istruzioni di installazione per il proprio sistema operativo.

Beaver è uno strumento di gestione e sviluppo per database universale e gratuito. Supporta tutti i principali database SQL, inclusi PostgreSQL, MySQL, SQLite, Oracle e molti altri. DBeaver offre funzionalità come la creazione e gestione di database, l’esecuzione di query SQL, la visualizzazione di tabelle e molto altro.

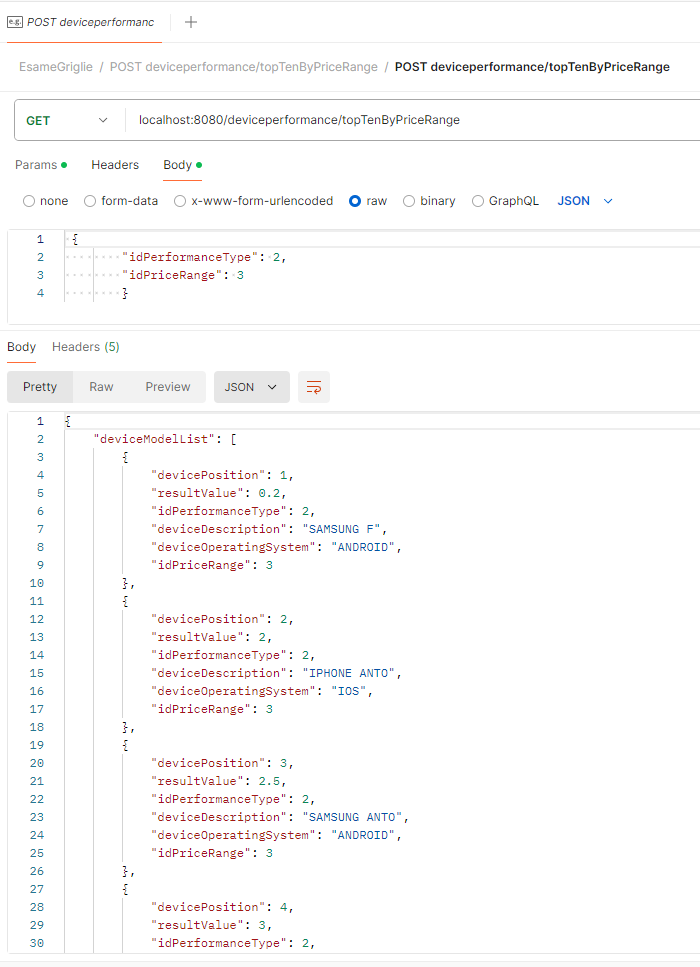


PostgreSQL e DBeaver sono strumenti potenti per la gestione dei database. Con PostgreSQL, hai a disposizione un sistema di gestione di database robusto e scalabile, mentre DBeaver offre un’interfaccia grafica intuitiva per interagire con i tuoi database.

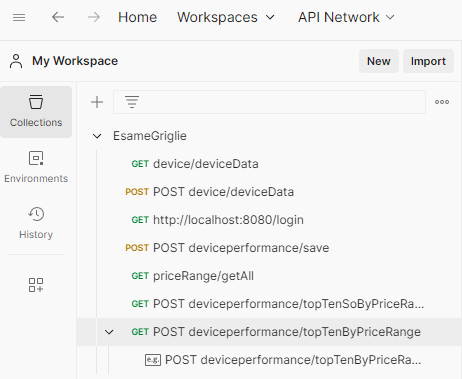
# POSTMAN

Postman è uno strumento versatile e potente utilizzato principalmente per il test delle API (Application Programming Interface).

Tra le sue principali funzionalità, Postman consente di inviare richieste HTTP come GET, POST, PUT e DELETE, permettendo agli sviluppatori di interagire facilmente con le API. Inoltre, supporta vari formati di risposta come JSON, XML, HTML e testo semplice.



Postman offre la possibilità di creare raccolte di richieste, che possono essere organizzate in cartelle e sottocartelle, facilitando la gestione e la condivisione dei test.



Un’altra caratteristica chiave è la capacità di eseguire script pre e post-richiesta per automatizzare i test e verificare le risposte.

Grazie alla sua integrazione con strumenti di CI/CD, Postman aiuta a mantenere pratiche di sviluppo continuo e a migliorare l’efficienza del flusso di lavoro.

# Flutter e Dart

Flutter e Dart sono tecnologie sviluppate da Google che hanno guadagnato grande popolarità negli ultimi anni. Flutter è un framework open-source per la creazione di applicazioni nativamente compilate per mobile, web e desktop da un singolo codice sorgente, mentre Dart è il linguaggio di programmazione utilizzato per sviluppare queste applicazioni.

Flutter è stato introdotto da Google nel 2017 come un framework open-source per lo sviluppo di applicazioni mobili. La sua prima versione stabile, Flutter 1.0, è stata rilasciata nel dicembre 2018. Flutter è stato progettato per affrontare le limitazioni degli strumenti di sviluppo esistenti, come React Native e Xamarin, offrendo una soluzione più performante e versatile.

Dal suo lancio, Flutter ha visto una rapida adozione e una crescita significativa. Google ha continuato a migliorare il framework con aggiornamenti regolari, introducendo nuove funzionalità e miglioramenti delle prestazioni. Nel 2019, Google ha annunciato Flutter for Web, estendendo le capacità di Flutter per supportare lo sviluppo di applicazioni web. Nel 2020, è stato rilasciato Flutter 2.0, che ha portato il supporto per le applicazioni desktop, rendendo Flutter un vero framework multi-piattaforma.

Flutter ha una comunità di sviluppatori molto attiva e un ecosistema in crescita di pacchetti e plugin. La comunità contribuisce regolarmente al miglioramento del framework e alla creazione di risorse educative. Flutter è supportato da un'ampia gamma di pacchetti disponibili su pub.dev, il repository ufficiale di pacchetti per Dart e Flutter.

Dart è stato introdotto da Google nel 2011 come un linguaggio di programmazione open-source. È stato progettato per essere semplice, veloce e sicuro, con un focus particolare sullo sviluppo web. Dart è stato concepito come un'alternativa a JavaScript, con l'obiettivo di migliorare la produttività degli sviluppatori e le prestazioni delle applicazioni.

Dart è un linguaggio orientato agli oggetti e tipizzato staticamente. Alcune delle sue caratteristiche principali includono:

- \*\*Sintassi Semplice\*\*: La sintassi di Dart è simile a quella di altri linguaggi popolari come Java, C# e JavaScript, rendendolo facile da apprendere per gli sviluppatori con esperienza in questi linguaggi.

- \*\*Tipizzazione Flessibile\*\*: Dart supporta sia la tipizzazione statica che dinamica, permettendo agli sviluppatori di scegliere l'approccio più adatto alle loro esigenze.

- \*\*Prestazioni Elevate\*\*: Dart è progettato per essere compilato in codice macchina nativo, offrendo prestazioni elevate per le applicazioni.

- \*\*Asincronia\*\*: Dart ha un supporto robusto per la programmazione asincrona tramite Future e Stream, facilitando la gestione delle operazioni asincrone.

Dart è utilizzato principalmente per lo sviluppo di applicazioni con Flutter. Tuttavia, può essere utilizzato anche per lo sviluppo web e server-side. Alcuni esempi di utilizzi includono:

- \*\*Applicazioni Mobile\*\*: Dart è il linguaggio principale per lo sviluppo di applicazioni Flutter per iOS e Android.

- \*\*Applicazioni Web\*\*: Con Dart, è possibile sviluppare applicazioni web performanti che possono essere eseguite su qualsiasi browser moderno.

- \*\*Applicazioni Desktop\*\*: Dart, insieme a Flutter, consente lo sviluppo di applicazioni desktop per Windows, macOS e Linux.

- \*\*Backend Server\*\*: Dart può essere utilizzato per creare server e API backend, offrendo una soluzione completa per lo sviluppo full-stack.

Flutter è ampiamente utilizzato per lo sviluppo di applicazioni mobile per iOS e Android. Grazie al suo motore di rendering nativo, Flutter offre prestazioni elevate e una user experience fluida. Gli sviluppatori possono creare interfacce utente personalizzate con facilità, utilizzando i widget di Flutter.

Con Flutter for Web, gli sviluppatori possono creare applicazioni web che offrono un'esperienza utente nativa. Flutter permette di condividere il codice tra le piattaforme mobile e web, riducendo il tempo di sviluppo e i costi.

Flutter supporta lo sviluppo di applicazioni desktop per Windows, macOS e Linux. Questo permette agli sviluppatori di creare applicazioni multi-piattaforma con un singolo codice sorgente, garantendo una user experience coerente su tutte le piattaforme.

Flutter è uno strumento eccellente per la prototipazione rapida e il design delle interfacce utente. La sua capacità di visualizzare i cambiamenti in tempo reale permette ai designer e agli sviluppatori di iterare rapidamente e testare diverse soluzioni.

Vantaggi e Svantaggi di Flutter e Dart

Vantaggi di Flutter

- \*\*Prestazioni Elevate\*\*: Grazie alla compilazione nativa, Flutter offre prestazioni paragonabili alle applicazioni native.

- \*\*Sviluppo Rapido\*\*: La funzionalità di hot reload di Flutter permette agli sviluppatori di vedere i cambiamenti in tempo reale, accelerando il ciclo di sviluppo.

- \*\*UI Personalizzabile\*\*: Flutter offre una vasta gamma di widget personalizzabili che permettono di creare interfacce utente uniche.

- \*\*Multi-Piattaforma\*\*: Flutter supporta lo sviluppo per mobile, web e desktop con un singolo codice sorgente.

Svantaggi di Flutter

- \*\*Dimensione dell'App\*\*: Le applicazioni Flutter tendono ad avere dimensioni maggiori rispetto alle applicazioni native.

- \*\*Maturità\*\*: Anche se Flutter è in rapida crescita, alcuni sviluppatori potrebbero trovare che manca di maturità rispetto ad altri framework più affermati.

- \*\*Comunità e Risorse\*\*: Anche se in crescita, la comunità e le risorse disponibili per Flutter potrebbero essere meno estese rispetto ad altri framework.

Vantaggi di Dart

- \*\*Sintassi Semplice\*\*: Dart è facile da apprendere per gli sviluppatori con esperienza in altri linguaggi orientati agli oggetti.

- \*\*Prestazioni Elevate\*\*: Dart è progettato per offrire prestazioni elevate, sia per applicazioni client-side che server-side.

- \*\*Asincronia\*\*: Dart ha un supporto robusto per la programmazione asincrona, facilitando la gestione delle operazioni asincrone.

Svantaggi di Dart

- \*\*Adozione Limitata\*\*: Rispetto a linguaggi come JavaScript, Dart ha una base di utenti più limitata.

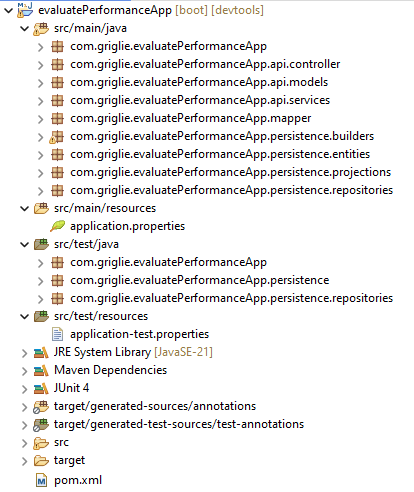
- \*\*Ecosistema\*\*: Anche se in crescita, l'ecosistema di pacchetti e librerie per Dart potrebbe essere meno esteso rispetto ad altri linguaggi più popolari.

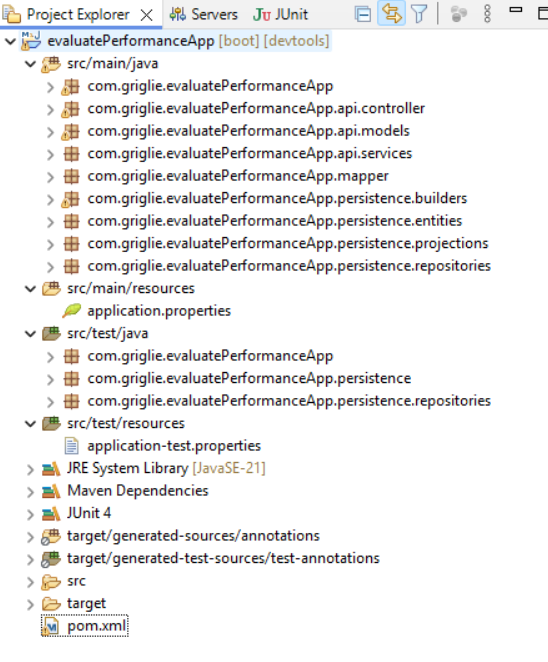
---

Flutter e Dart rappresentano una potente combinazione per lo sviluppo di applicazioni multi-piattaforma. Flutter offre un framework versatile e performante per la creazione di interfacce utente moderne e reattive, mentre Dart fornisce un linguaggio di programmazione robusto e semplice da apprendere. Nonostante alcuni svantaggi, come la dimensione dell'app e l'adozione limitata, Flutter e Dart continuano a guadagnare popolarità e supporto nella comunità degli sviluppatori. Con il continuo sviluppo e miglioramento, queste tecnologie sono destinate a giocare un ruolo sempre più importante nel futuro dello sviluppo software.

# EVALUATE PERFORMANCE APP

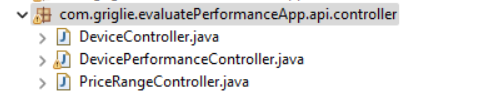
Il backend dell’applicazione è formata dai seguenti package. Di seguito l’illustrazione e la descrizione dettagliata di ciascuna di essi.





## **Package** com.griglie.evaluatePerformanceApp.api.controller

In uno schema MVC (model – view- controller) il controller riceve i comandi dell'utente attraverso il View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato del View.



La classe **DeviceController** nel progetto evaluatePerformanceApp è un controller Spring REST che gestisce le operazioni CRUD (Create, Read, Update, Delete) sui dispositivi (Device). Ecco una panoramica delle funzionalità della classe:

1. **Annotazioni e Dipendenze**:
   * @RestController: Indica che questa classe è un controller REST.
   * @RequestMapping("/device"): Mappa le richieste HTTP all'URL /device.
   * @Autowired: Inietta automaticamente il servizio DeviceService.
2. **Metodi**:
   * getDeviceData(): Gestisce le richieste GET a /device/deviceData e restituisce una lista di tutti i dispositivi.
   * getDeviceDataById(@PathVariable Long id): Gestisce le richieste GET a /device/deviceData/{id} e restituisce i dati del dispositivo con l'ID specificato.
   * addDeviceData(@RequestBody Device device): Gestisce le richieste POST a /device/deviceData per aggiungere un nuovo dispositivo. Restituisce una risposta HTTP con lo stato CREATED e il dispositivo appena aggiunto.

La classe utilizza il servizio DeviceService per eseguire le operazioni sui dispositivi e gestisce le risposte HTTP in modo appropriato.

La classe **DevicePerformanceController** nel progetto evaluatePerformanceApp è un controller Spring REST che gestisce le operazioni relative alle prestazioni dei dispositivi. Ecco una panoramica delle funzionalità della classe:

1. **Annotazioni e Dipendenze**:
   * @RestController: Indica che questa classe è un controller REST.
   * @RequestMapping("/deviceperformance"): Mappa le richieste HTTP all'URL /deviceperformance.
   * @Autowired: Inietta automaticamente i servizi DevicePerformanceService, DeviceService e il mapper DeviceMapper.
2. **Endpoint e Metodi**:
   * @PostMapping(value=SAVE\_PERFORMANCE): Gestisce le richieste POST a /deviceperformance/save per salvare o aggiornare i dati di prestazione di un dispositivo. Restituisce una risposta con lo stato OK e i dettagli della prestazione salvata.
   * @GetMapping(value=DEVICE\_POSITION): Gestisce le richieste GET a /deviceperformance/deviceposition per ottenere la posizione di un dispositivo in base ai KPI specificati nella richiesta.
   * @GetMapping(value=TOP\_TEN\_SO\_PRICE\_RANGE): Gestisce le richieste GET a /deviceperformance/topTenBySOPriceRange per ottenere i primi dieci dispositivi per sistema operativo e fascia di prezzo specificati.
   * @GetMapping(value=TOP\_TEN\_PRICE\_RANGE): Gestisce le richieste GET a /deviceperformance/topTenByPriceRange per ottenere i primi dieci dispositivi per fascia di prezzo specificata.
3. **Modelli e Servizi Utilizzati**:
   * Utilizza vari modelli come PerformanceSaveRequest, PerformanceSaveResponse, TopTenDeviceResponse, TopTenPriceRangeRequest, e TopTenSOPriceRangeRequest.
   * I servizi DevicePerformanceService e DeviceService sono utilizzati per eseguire le operazioni di business logic.
   * Il mapper DeviceMapper è utilizzato per convertire i dati tra i modelli e le entità.

La classe fornisce funzionalità per salvare i dati di prestazione dei dispositivi, ottenere la posizione di un dispositivo in base ai KPI, e recuperare i migliori dispositivi in base a criteri di sistema operativo e fascia di prezzo.

La classe **PriceRangeController** nel progetto evaluatePerformanceApp è un controller Spring REST che gestisce le operazioni relative alle fasce di prezzo. Ecco una panoramica delle funzionalità della classe:

1. **Annotazioni e Dipendenze**:

- `@RestController`: Indica che questa classe è un controller REST.

- `@RequestMapping("/priceRange")`: Mappa le richieste HTTP all'URL `/priceRange`.

- `@Autowired`: Inietta automaticamente il servizio `PriceRangeService`.

2. **Metodi**:

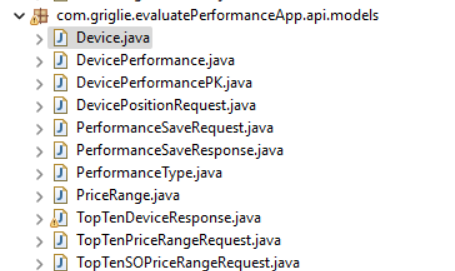
- `getAllPriceRange()`: Gestisce le richieste GET a `/priceRange/getAll` e restituisce una lista di tutte le fasce di prezzo.

La classe utilizza il servizio `PriceRangeService` per eseguire le operazioni necessarie e gestisce le risposte HTTP in modo appropriato.

## **Package** com.griglie.evaluatePerformanceApp.api.models

I models generalmente rappresentano le entità principali dell'applicazione e sono utilizzati per trasferire dati tra i livelli di servizio e di persistenza.

Le annotazioni Lombok utilizzate, semplificano la creazione e la gestione di queste classi eliminando la necessità di scrivere boilerplate code come getter, setter e costruttori.



La classe **Device** nel progetto evaluatePerformanceApp rappresenta un modello di dispositivo con vari attributi relativi alle sue prestazioni. Ecco una panoramica delle funzionalità della classe:

1. **Annotazioni Lombok**:
   * @Getter e @Setter: Generano automaticamente i metodi getter e setter per tutti i campi.
   * @Builder: Fornisce un pattern builder per creare istanze della classe.
   * @NoArgsConstructor: Genera un costruttore senza argomenti.
   * @AllArgsConstructor(staticName="of"): Genera un costruttore con tutti gli argomenti e un metodo statico of per creare istanze.
2. **Campi**:
   * private Long devicePosition: La posizione del dispositivo.
   * private BigDecimal resultValue: Il valore del risultato delle prestazioni del dispositivo.
   * private Long idPerformanceType: L'ID del tipo di prestazione.
   * private String deviceDescription: La descrizione del dispositivo.
   * private String deviceOperatingSystem: Il sistema operativo del dispositivo.
   * private Long idPriceRange: L'ID della fascia di prezzo.

La classe Device è utilizzata per rappresentare i dati relativi ai dispositivi, inclusi i loro risultati di prestazione, descrizione, sistema operativo, e fascia di prezzo.

La classe **DevicePerformance** nel progetto evaluatePerformanceApp rappresenta un modello per le prestazioni di un dispositivo. Ecco una panoramica delle funzionalità della classe:

1. **Annotazioni Lombok**:

- `@Getter` e `@Setter`: Generano automaticamente i metodi getter e setter per tutti i campi.

- `@Builder`: Fornisce un pattern builder per creare istanze della classe.

- `@NoArgsConstructor`: Genera un costruttore senza argomenti.

- `@AllArgsConstructor(staticName="of")`: Genera un costruttore con tutti gli argomenti e un metodo statico `of` per creare istanze.

2. **Campi**:

- `private Long idPerformanceType`: L'ID del tipo di prestazione.

- `private Long result`: Il risultato delle prestazioni del dispositivo.

- `private Instant creationDate`: La data di creazione della registrazione delle prestazioni.

La classe DevicePerformance è utilizzata per rappresentare i dati relativi alle prestazioni dei dispositivi, inclusi il tipo di prestazione, il risultato e la data di creazione.

La classe **DevicePerformancePK** nel progetto evaluatePerformanceApp è una chiave primaria composta per l'entità DevicePerformance. Ecco una panoramica delle funzionalità della classe:

1. **Annotazioni**:
   * @NoArgsConstructor e @AllArgsConstructor: Generano costruttori senza argomenti e con tutti gli argomenti, rispettivamente.
   * @Embeddable: Indica che questa classe può essere incorporata in altre entità.
2. **Campi**:
   * private Device device: Rappresenta il dispositivo.
   * private PerformanceType performanceType: Rappresenta il tipo di prestazione.
3. **Metodi**:
   * getDevice(), setDevice(Device device): Getter e setter per il campo device.
   * getPerformanceType(), setPerformanceType(PerformanceType performanceType): Getter e setter per il campo performanceType.
   * equals(Object o): Verifica l'uguaglianza tra due oggetti DevicePerformancePK.
   * hashCode(): Genera il codice hash per l'oggetto DevicePerformancePK.
4. **Annotazioni sulle relazioni**:
   * @ManyToOne(fetch = FetchType.LAZY) e @JoinColumn(name = "ID\_DEVICE"): Definiscono una relazione molti-a-uno con l'entità Device.
   * @ManyToOne(fetch = FetchType.LAZY) e @JoinColumn(name = "ID\_PERFORMANCE\_TYPE"): Definiscono una relazione molti-a-uno con l'entità PerformanceType.

Questa classe viene utilizzata come chiave primaria composta per l'entità DevicePerformance, combinando le informazioni del dispositivo e del tipo di prestazione.

La classe **DevicePositionRequest** nel progetto **evaluatePerformanceApp** rappresenta un modello di richiesta per la posizione dei dispositivi. Ecco una panoramica delle funzionalità della classe:

1. **Annotazioni Lombok**:

- `@Getter` e `@Setter`: Generano automaticamente i metodi getter e setter per tutti i campi.

- `@Builder`: Fornisce un pattern builder per creare istanze della classe.

- `@NoArgsConstructor`: Genera un costruttore senza argomenti.

- `@AllArgsConstructor`: Genera un costruttore con tutti gli argomenti.

2. **Campi**:

- `private Long idPerformanceType`: L'ID del tipo di prestazione.

- `private String deviceDescription`: La descrizione del dispositivo.

- `private String deviceOperatingSystem`: Il sistema operativo del dispositivo.

- `private Long idPriceRange`: L'ID della fascia di prezzo.

La classe DevicePositionRequest è utilizzata per rappresentare i dati necessari per una richiesta di posizione dei dispositivi, inclusi il tipo di prestazione, la descrizione del dispositivo, il sistema operativo e la fascia di prezzo.

Le seguenti classi rappresentano vari modelli di dati utilizzati per gestire le prestazioni dei dispositivi e le richieste relative a fasce di prezzo e sistemi operativi.

**PerformanceSaveResponse**:

Questa classe rappresenta la risposta di salvataggio delle prestazioni.

- **Campi**:

- `Long idPerformanceType`: L'ID del tipo di prestazione.

- `BigDecimal result`: Il risultato della prestazione.

- `Instant executionDate`: La data di esecuzione.

- `String deviceDescription`: La descrizione del dispositivo.

- `String deviceOperatingSystem`: Il sistema operativo del dispositivo.

- `Long idPriceRange`: L'ID della fascia di prezzo.

- `Long position`: La posizione del dispositivo.

- **Costruttore**: Un costruttore che accetta `DevicePerformance`, `Device` e `Long position` per inizializzare i campi.

‘java

public PerformanceSaveResponse(DevicePerformance dp, Device device, Long position) {

this.result = dp.getResult();

this.idPerformanceType = dp.getKey().getPerformanceType().getId();

this.executionDate = dp.getCreationDate().toInstant();

this.deviceDescription = device.getDescription();

this.deviceOperatingSystem = device.getOperatingSystem();

this.idPriceRange = device.getIdPriceRange();

this.position = position;

}

‘

**PerformanceType**:

Questa classe rappresenta un tipo di prestazione.

- **Campi**:

- `String id`: L'ID del tipo di prestazione.

- `String description`: La descrizione del tipo di prestazione.

`java

@Getter

@Setter

@Builder

@NoArgsConstructor

@AllArgsConstructor(staticName="of")

public class PerformanceType {

@NotNull

private String id;

@NotNull

private String description;

}

`

**PriceRange**:

- Questa classe rappresenta una fascia di prezzo.

- **Campi**:

- `Long id`: L'ID della fascia di prezzo.

- `String description`: La descrizione della fascia di prezzo.

- `String shortDescription`: Una breve descrizione della fascia di prezzo.

`java

@Getter

@Setter

@Builder

@NoArgsConstructor

@AllArgsConstructor(staticName="of")

public class PriceRange {

@NotNull

private Long id;

private String description;

private String shortDescription;

}

`

**TopTenDeviceResponse**:

Questa classe rappresenta la risposta per i primi dieci dispositivi.

- **Campi**:

- `List<Device> deviceModelList`: Una lista di modelli di dispositivi.

`java

@Getter

@Setter

@Builder

@NoArgsConstructor

@AllArgsConstructor

public class TopTenDeviceResponse {

private List<Device> deviceModelList;

}

`

**TopTenPriceRangeRequest**:

Questa classe rappresenta una richiesta per i primi dieci dispositivi in una certa fascia di prezzo.

-**Campi**:

- `Long idPerformanceType`: L'ID del tipo di prestazione.

- `Long idPriceRange`: L'ID della fascia di prezzo.

`java

@Getter

@Setter

@Builder

@NoArgsConstructor

@AllArgsConstructor

public class TopTenPriceRangeRequest {

private Long idPerformanceType;

private Long idPriceRange;

}

`

**TopTenSOPriceRangeRequest:**

Questa classe rappresenta una richiesta per i primi dieci dispositivi in una certa fascia di prezzo e sistema operativo.

- **Campi**:

- `Long idPerformanceType`: L'ID del tipo di prestazione.

- `String deviceOperatingSystem`: Il sistema operativo del dispositivo.

- `Long idPriceRange`: L'ID della fascia di prezzo.

`java

@Getter

@Setter

@Builder

@NoArgsConstructor

@AllArgsConstructor

public class TopTenSOPriceRangeRequest {

private Long idPerformanceType;

private String deviceOperatingSystem;

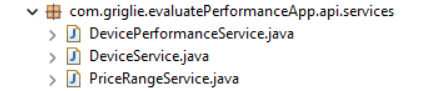
private Long idPriceRange;

}

`

## **Package** com.griglie.evaluatePerformanceApp.api.services

Le classi che compongono il package sono utilizzate per fornire servizi legati alla gestione dei dispositivi, delle loro prestazioni e delle fasce di prezzo, facilitando le operazioni CRUD (Create, Read, Update, Delete) e la logica di business associata.



**DevicePerformanceService**

Questa classe fornisce metodi per gestire i dati delle prestazioni dei dispositivi.

**Metodi principali**:

- `saveDevicePerformanceData(DevicePerformance devicePerformance)`: Salva i dati delle prestazioni di un dispositivo.

- `saveOrUpdateDevicePerformanceData(Device device, Long idPerformanceType, BigDecimal result)`: Salva o aggiorna i dati delle prestazioni di un dispositivo.

- `getDevicePositionByKpi(String deviceDescription, String deviceOperatingSystem, Long idPerformanceType)`: Ottiene la posizione del dispositivo in base al KPI.

- `getDeviceOperatingSystemTopTen(String operatingSystem, Long idPriceRange, Long idPerformanceType, Pageable pageable)`: Ottiene i primi dieci dispositivi per sistema operativo e fascia di prezzo.

- `getDevicePriceRangeTopTen(Long idPriceRange, Long idPerformanceType, Pageable pageable)`: Ottiene i primi dieci dispositivi per fascia di prezzo.

E’ utilizzata da altri componenti del sistema che necessitano di gestire i dati delle prestazioni dei dispositivi.

Serve a gestire e recuperare i dati delle prestazioni dei dispositivi, incluse le operazioni di salvataggio e aggiornamento.

**DeviceService**

Questa classe fornisce metodi per gestire i dati dei dispositivi.

**Metodi principali:**

- `getAllDeviceData()`: Recupera tutti i dati dei dispositivi.

- `getDeviceById(Long id)`: Recupera i dati di un dispositivo per ID.

- `addDeviceData(Device device)`: Aggiunge i dati di un dispositivo.

- `saveOrUpdateDevice(String deviceDescription, String deviceOperatingSystem, Long idPriceRange)`: Salva o aggiorna i dati di un dispositivo.

E’ utilizzata da altri componenti del sistema che necessitano di gestire i dati dei dispositivi.

Serve a gestire e recuperare i dati dei dispositivi, incluse le operazioni di salvataggio e aggiornamento.

**PriceRangeService**

Questa classe fornisce metodi per gestire le fasce di prezzo.

**Metodi principali:**

- `getAllPriceRange()`: Recupera tutte le fasce di prezzo.

E’ utilizzata da altri componenti del sistema che necessitano di gestire le fasce di prezzo.

Serve a gestire e recuperare le fasce di prezzo disponibili.

## **Package** com.griglie.evaluatePerformanceApp.mapper



La classe `DeviceMapper` nel progetto `evaluatePerformanceApp` serve per mappare e trasformare le entità `DeviceProjection` in modelli di `Device` e creare risposte di tipo `TopTenDeviceResponse`. Ecco una panoramica delle sue funzionalità:

**Annotazioni:**

- `@Component`: Indica che questa classe è un componente Spring, quindi gestita dal contenitore Spring.

**Metodi:**

- `toTopTenDeviceResponse(List<DeviceProjection> deviceProjectionList)`:

- Converte una lista di `DeviceProjection` in un oggetto `TopTenDeviceResponse`.

- Crea una lista di `Device` a partire dalle proiezioni e la inserisce in un oggetto `TopTenDeviceResponse`.

- `toDeviceModel(DeviceProjection deviceProjection)`:

- Converte un singolo oggetto `DeviceProjection` in un oggetto `Device`.

- Imposta i campi del modello `Device` con i valori della proiezione.

Questa classe può essere utilizzata dai servizi che necessitano di trasformare i dati delle proiezioni dei dispositivi in modelli di dispositivo o in risposte per le API.

Serve a mappare i dati delle proiezioni dei dispositivi in modelli di dispositivo utilizzabili all'interno dell'applicazione e a creare risposte strutturate per le API.

Esempio di utilizzo del metodo `toTopTenDeviceResponse`:

```java

// Supponiamo di avere una lista di DeviceProjection

List<DeviceProjection> deviceProjections = devicePerformanceService.getDevicePriceRangeTopTen(idPriceRange, idPerformanceType, pageable);

// Creiamo un'istanza di DeviceMapper

DeviceMapper deviceMapper = new DeviceMapper();

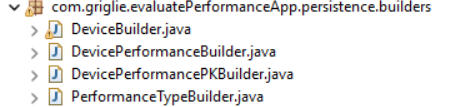
// Convertiamo la lista di DeviceProjection in una risposta TopTenDeviceResponse

TopTenDeviceResponse response = deviceMapper.toTopTenDeviceResponse(deviceProjections);

```

In questo esempio, `deviceMapper` viene utilizzato per trasformare una lista di proiezioni di dispositivi in una risposta strutturata `TopTenDeviceResponse`, che può essere poi restituita da un controller API.

## **Package** com.griglie.evaluatePerformanceApp.persistence.builders



Le classi sono dei costruttori (builder) utilizzati per creare e configurare entità nel progetto `evaluatePerformanceApp`.

Queste classi sono utilizzate per costruire entità con un approccio "builder", che rende il codice più leggibile e configurabile. Ogni costruttore consente di impostare vari campi e di creare istanze delle entità con i valori specificati.

**DeviceBuilder**

Questa classe fornisce un costruttore per creare e configurare oggetti `Device`.

**Metodi principali**:

- `withId(Long id)`: Imposta l'ID del dispositivo.

- `withDescription(String description)`: Imposta la descrizione del dispositivo.

- `withOperatingSystem(String operatingSystem)`: Imposta il sistema operativo del dispositivo.

- `withIdPriceRange(Long idPriceRange)`: Imposta l'ID della fascia di prezzo del dispositivo.

- `build()`: Costruisce e restituisce un'istanza di `Device` con i valori impostati.

Serve a creare oggetti `Device` con valori specificati in modo fluido e leggibile.

`Java

public final class DeviceBuilder {

private Long id;

private String description;

private String operatingSystem;

private Long idPriceRange;

public static DeviceBuilder aDevice() {

return new DeviceBuilder();

}

public DeviceBuilder withId(Long id) {

this.id = id;

return this;

}

public DeviceBuilder withDescription(String description) {

this.description = description;

return this;

}

public DeviceBuilder withOperatingSystem(String operatingSystem) {

this.operatingSystem = operatingSystem;

return this;

}

public DeviceBuilder withIdPriceRange(Long idPriceRange) {

this.idPriceRange = idPriceRange;

return this;

}

public Device build() {

Device device = new Device();

device.setId(id);

device.setDescription(description);

device.setOperatingSystem(operatingSystem);

device.setIdPriceRange(idPriceRange);

device.setLastModifiedDate(Timestamp.from(Instant.now()));

device.setCreationDate(Timestamp.from(Instant.now()));

return device;

}

}

`

**DevicePerformanceBuilder**

Questa classe fornisce un costruttore per creare e configurare oggetti `DevicePerformance`.

**Metodi principali:**

- `withKey(DevicePerformancePK key)`: Imposta la chiave primaria dell'entità `DevicePerformance`.

- `withResult(BigDecimal result)`: Imposta il risultato delle prestazioni del dispositivo.

- `build()`: Costruisce e restituisce un'istanza di `DevicePerformance` con i valori impostati.

Serve a creare oggetti `DevicePerformance` con valori specificati in modo fluido e leggibile.

`java

public final class DevicePerformanceBuilder {

private DevicePerformancePK key;

private BigDecimal result;

public static DevicePerformanceBuilder aDevicePerformance() {

return new DevicePerformanceBuilder();

}

public DevicePerformanceBuilder withKey(DevicePerformancePK key) {

this.key = key;

return this;

}

public DevicePerformanceBuilder withResult(BigDecimal result) {

this.result = result;

return this;

}

public DevicePerformance build() {

DevicePerformance devicePerformance = new DevicePerformance();

devicePerformance.setKey(key);

devicePerformance.setResult(result);

devicePerformance.setLastModifiedDate(Timestamp.from(Instant.now()));

devicePerformance.setCreationDate(Timestamp.from(Instant.now()));

return devicePerformance;

}

}

`

**DevicePerformancePKBuilder**

Questa classe fornisce un costruttore per creare e configurare chiavi primarie (`DevicePerformancePK`) per l'entità `DevicePerformance`.

**Metodi principali:**

- `withDevice(Device device)`: Imposta l'oggetto `Device` nella chiave primaria.

- `withPerformanceType(PerformanceType performanceType)`: Imposta il tipo di prestazione nella chiave primaria.

- `build()`: Costruisce e restituisce un'istanza di `DevicePerformancePK` con i valori impostati.

Serve a creare chiavi primarie per l'entità `DevicePerformance` con valori specificati in modo fluido e leggibile.

`java

public final class DevicePerformancePKBuilder {

private Device device;

private PerformanceType performanceType;

public static DevicePerformancePKBuilder aDevicePerformancePK() {

return new DevicePerformancePKBuilder();

}

public DevicePerformancePKBuilder withDevice(Device device) {

this.device = device;

return this;

}

public DevicePerformancePKBuilder withPerformanceType(PerformanceType performanceType) {

this.performanceType = performanceType;

return this;

}

public DevicePerformancePK build() {

DevicePerformancePK devicePerformancePK = new DevicePerformancePK();

devicePerformancePK.setDevice(device);

devicePerformancePK.setPerformanceType(performanceType);

return devicePerformancePK;

}

}

`

**PerformanceTypeBuilder**

Questa classe fornisce un costruttore per creare e configurare oggetti `PerformanceType`.

**Metodi principali:**

- `withId(Long id)`: Imposta l'ID del tipo di prestazione.

- `withDescription(String description)`: Imposta la descrizione del tipo di prestazione.

- `build()`: Costruisce e restituisce un'istanza di `PerformanceType` con i valori impostati.

Serve a creare oggetti `PerformanceType` con valori specificati in modo fluido e leggibile.

`java

public final class PerformanceTypeBuilder {

private Long id;

private String description;

public static PerformanceTypeBuilder aPriceRangeBuilder() {

return new PerformanceTypeBuilder();

}

public PerformanceTypeBuilder withId(Long id) {

this.id = id;

return this;

}

public PerformanceTypeBuilder withDescription(String description) {

this.description = description;

return this;

}

public PerformanceType build() {

PerformanceType performanceType = new PerformanceType();

performanceType.setId(id);

performanceType.setDescription(description);

return performanceType;

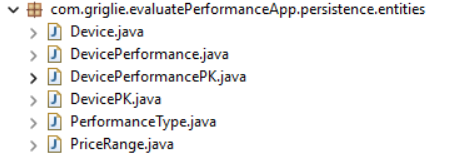
}

}

`

## **Package** com.griglie.evaluatePerformanceApp.persistence.entities

Queste classi sono utilizzate per rappresentare le entità nel database e gestire i dati delle prestazioni dei dispositivi, i tipi di prestazioni e le fasce di prezzo. Le chiavi primarie composte aiutano a identificare univocamente le entità nelle tabelle del database.



Di seguito una panoramica delle classi.

**Device**

Questa classe rappresenta l'entità di un dispositivo.

**Campi:**

- `Long id`: L'ID del dispositivo.

- `String description`: La descrizione del dispositivo.

- `String operatingSystem`: Il sistema operativo del dispositivo.

- `Long idPriceRange`: L'ID della fascia di prezzo del dispositivo.

- `Timestamp creationDate`: La data di creazione del dispositivo.

- `Timestamp lastModifiedDate`: La data dell'ultimo aggiornamento del dispositivo.

Serve a rappresentare e gestire i dati di un dispositivo nel database.

`java

@Entity

@Setter

@Table(name = "DEVICE")

public class Device implements Serializable {

private static final long serialVersionUID = -3805544452344967627L;

@Id

@SequenceGenerator(name="device\_id\_seq", sequenceName="device\_id\_seq", allocationSize=1)

@GeneratedValue(strategy= GenerationType.SEQUENCE, generator= "device\_id\_seq")

@Column(name = "ID", insertable=false, updatable=false)

private Long id;

@Column(name = "DESCRIPTION")

private String description;

@Column(name = "operating\_system")

private String operatingSystem;

@Column(name = "id\_price\_range")

private Long idPriceRange;

@CreationTimestamp

@Column(name = "creation\_date", updatable = false)

private Timestamp creationDate;

@UpdateTimestamp

@Column(name = "last\_modified\_date")

private Timestamp lastModifiedDate;

// equals and hashCode methods

}

`

**DevicePK**

Questa classe rappresenta la chiave primaria composta per l'entità `Device`.

**Campi:**

- `String description`: La descrizione del dispositivo.

- `String operatingSystem`: Il sistema operativo del dispositivo.

Serve a definire una chiave primaria composta per identificare univocamente un dispositivo nel database.

`java

@Embeddable

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class DevicePK implements Serializable {

private static final long serialVersionUID = -3805544452344967627L;

@Column(name = "DESCRIPTION")

private String description;

@Column(name = "operating\_system")

private String operatingSystem;

// equals and hashCode methods

}

`

**DevicePerformance**

Questa classe rappresenta l'entità delle prestazioni di un dispositivo.

**Campi:**

- `DevicePerformancePK key`: La chiave primaria composta dell'entità `DevicePerformance`.

- `BigDecimal result`: Il risultato delle prestazioni del dispositivo.

- `Timestamp creationDate`: La data di creazione delle prestazioni del dispositivo.

- `Timestamp lastModifiedDate`: La data dell'ultimo aggiornamento delle prestazioni del dispositivo.

Serve a rappresentare e gestire i dati delle prestazioni di un dispositivo nel database.

`java

@Entity

@Setter

@Table(name = "DEVICE\_PERFORMANCE")

public class DevicePerformance implements Serializable {

private static final long serialVersionUID = -3805544452344967627L;

@EmbeddedId

private DevicePerformancePK key;

@Column(name = "result")

private BigDecimal result;

@CreationTimestamp

@Column(name = "creation\_date", updatable = false)

private Timestamp creationDate;

@UpdateTimestamp

@Column(name = "last\_modified\_date")

private Timestamp lastModifiedDate;

// equals and hashCode methods

}

`

**DevicePerformancePK**

Questa classe rappresenta la chiave primaria composta per l'entità `DevicePerformance`.

**Campi**:

- `Device device`: L'entità `Device`.

- `PerformanceType performanceType`: Il tipo di prestazione.

Serve a definire una chiave primaria composta per identificare univocamente le prestazioni di un dispositivo nel database.

`java

@Embeddable

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class DevicePerformancePK implements Serializable {

private static final long serialVersionUID = -3805544452344967627L;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "ID\_DEVICE")

private Device device;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "ID\_PERFORMANCE\_TYPE")

private PerformanceType performanceType;

// equals and hashCode methods

}

`

**PerformanceType**

Questa classe rappresenta l'entità del tipo di prestazione.

**Campi**:

- `Long id`: L'ID del tipo di prestazione.

- `String description`: La descrizione del tipo di prestazione.

Serve a rappresentare e gestire i dati del tipo di prestazione nel database.

`java

@Entity

@Setter

@Table(name = "PERFORMANCE\_TYPES")

public class PerformanceType implements Serializable {

private static final long serialVersionUID = -3805544452344967627L;

@Id

@Column(name = "ID", insertable=false, updatable=false)

private Long id;

@Column(name = "DESCRIPTION")

private String description;

// equals and hashCode methods

}

`

**PriceRange:**

Questa classe rappresenta l'entità della fascia di prezzo.

**Campi**:

- `Long id`: L'ID della fascia di prezzo.

- `String description`: La descrizione della fascia di prezzo.

- `String shortDescription`: Una breve descrizione della fascia di prezzo.

Serve a rappresentare e gestire i dati della fascia di prezzo nel database.

`java

@Entity

@Setter

@Table(name = "PRICE\_RANGE")

public class PriceRange implements Serializable {

private static final long serialVersionUID = -3805544452344967627L;

@Id

@Column(name = "ID", insertable=false, updatable=false)

private Long id;

@Column(name = "DESCRIPTION")

private String description;

@Column(name = "SHORT\_DESCRIPTION")

private String shortDescription;

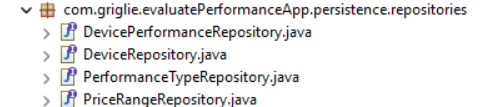
// equals and hashCode methods

}

```

## **Package** com.griglie.evaluatePerformanceApp.persistence.repositories

Queste interfacce repository sono utilizzate per interagire con il database e gestire le operazioni CRUD e le query personalizzate per le rispettive entità.



**DevicePerformanceRepository**

Questa interfaccia estende `JpaRepository` per fornire metodi CRUD e query personalizzate per l'entità `DevicePerformance`.

**Metodi principali:**

- `getKpiPositionByDescriptionAndOperatingSystem`: Restituisce la posizione KPI di un dispositivo specifico in base alla descrizione, al sistema operativo e al tipo di prestazione.

- `getKpiTopTenByOperatingSystemAndPriceRange`: Restituisce i primi 10 dispositivi in classifica in base alla fascia di prezzo, al sistema operativo e al tipo di prestazione.

- `getKpiTopTenByPriceRange`: Restituisce i primi 10 dispositivi in classifica in base alla fascia di prezzo e al tipo di prestazione.

Serve a eseguire operazioni di persistenza e query personalizzate sulle prestazioni dei dispositivi.

`java

@Repository

public interface DevicePerformanceRepository extends JpaRepository<DevicePerformance, DevicePerformancePK> {

@Query("SELECT A.RowNr "

+ "FROM ("

+ "SELECT ROW\_NUMBER() OVER (ORDER BY dp.result) AS RowNr, dev.description AS deviceDescription "

+ "FROM DevicePerformance dp "

+ "JOIN dp.key.performanceType pt on pt.id = :performanceType "

+ "JOIN dp.key.device dev on dev.operatingSystem = :operatingSystem "

+ ") A "

+ "WHERE A.deviceDescription = :description ")

Long getKpiPositionByDescriptionAndOperatingSystem (@Param("description") String description, @Param("operatingSystem") String operatingSystem, @Param("performanceType") Long performanceType);

@Query("SELECT new com.griglie.evaluatePerformanceApp.persistence.projections.DeviceProjection("

+ " ROW\_NUMBER() OVER (ORDER BY dp.result) AS devicePosition, "

+ "dp.result AS resultValue, "

+ "pt.id AS idPerformanceType, "

+ "dev.description AS deviceDescription, "

+ "dev.operatingSystem AS deviceOperatingSystem, "

+ "dev.idPriceRange AS idPriceRange) "

+ "FROM DevicePerformance dp "

+ "JOIN dp.key.performanceType pt on pt.id = :performanceType "

+ "JOIN dp.key.device dev on dev.operatingSystem = :operatingSystem AND dev.idPriceRange = :idPriceRange ")

List<DeviceProjection> getKpiTopTenByOperatingSystemAndPriceRange(@Param("operatingSystem") String operatingSystem, @Param("idPriceRange") Long idPriceRange, @Param("performanceType") Long performanceType);

@Query("SELECT new com.griglie.evaluatePerformanceApp.persistence.projections.DeviceProjection("

+ " ROW\_NUMBER() OVER (ORDER BY dp.result) AS devicePosition, "

+ "dp.result AS resultValue, "

+ "pt.id AS idPerformanceType, "

+ "dev.description AS deviceDescription, "

+ "dev.operatingSystem AS deviceOperatingSystem, "

+ "dev.idPriceRange AS idPriceRange) "

+ "FROM DevicePerformance dp "

+ "JOIN dp.key.performanceType pt on pt.id = :performanceType "

+ "JOIN dp.key.device dev on dev.idPriceRange = :idPriceRange ")

List<DeviceProjection> getKpiTopTenByPriceRange(@Param("idPriceRange") Long idPriceRange, @Param("performanceType") Long performanceType, Pageable pageable);

}

`

**DeviceRepository**

Questa interfaccia estende `JpaRepository` per fornire metodi CRUD e query personalizzate per l'entità `Device`.

**Metodi principali:**

- `findAllByDescriptionAndOperatingSystem`: Trova tutti i dispositivi in base alla descrizione e al sistema operativo.

- `existsByDescriptionAndOperatingSystem`: Verifica se un dispositivo esiste in base alla descrizione e al sistema operativo.

- `update`: Aggiorna la fascia di prezzo di un dispositivo in base alla descrizione e al sistema operativo.

Serve a eseguire operazioni di persistenza e query personalizzate sui dispositivi.

`java

@Repository

public interface DeviceRepository extends JpaRepository<Device, Long> {

Device findAllByDescriptionAndOperatingSystem(String description, String operatingSystem);

boolean existsByDescriptionAndOperatingSystem(String description, String operatingSystem);

@Transactional

@Modifying

@Query("UPDATE Device "

+ "SET idPriceRange = :idPriceRange, "

+ "lastModifiedDate = current\_timestamp "

+ "WHERE description = :description "

+ "AND operatingSystem = :operatingSystem")

void update(@Param("description") String description, @Param("operatingSystem") String operatingSystem, @Param("idPriceRange") Long idPriceRange);

}

`

**PerformanceTypeRepository**

Questa interfaccia estende `JpaRepository` per fornire metodi CRUD per l'entità `PerformanceType`.

Serve a eseguire operazioni di persistenza sui tipi di prestazione.

`java

@Repository

public interface PerformanceTypeRepository extends JpaRepository<PerformanceType, Long> {

}

`

**PriceRangeRepository**

Questa interfaccia estende `JpaRepository` per fornire metodi CRUD per l'entità `PriceRange`.

Serve a eseguire operazioni di persistenza sulle fasce di prezzo.

`java

@Repository

public interface PriceRangeRepository extends JpaRepository<PriceRange, Long> {

}

`

## **Package** com.griglie.evaluatePerformanceApp.persistence.projections



La classe `DeviceProjection` è una proiezione utilizzata per rappresentare una vista o un insieme di dati specifici relativi a un dispositivo in una query.

Questa classe viene utilizzata principalmente per proiettare e rappresentare i dati dei dispositivi in base a specifiche query nel repository, facilitando la gestione e la visualizzazione dei dati filtrati o calcolati.

**Importazioni**:

- `java.math.BigDecimal`: Utilizzato per rappresentare valori numerici precisi.

- `lombok.AllArgsConstructor`: Genera un costruttore con tutti gli argomenti.

- `lombok.Getter`: Genera i metodi getter per tutti i campi.

- `lombok.NoArgsConstructor`: Genera un costruttore senza argomenti.

- `lombok.Setter`: Genera i metodi setter per tutti i campi.

**Annotazioni:**

- `@Getter`: Genera metodi getter per tutti i campi.

- `@Setter`: Genera metodi setter per tutti i campi.

- `@AllArgsConstructor`: Genera un costruttore con tutti gli argomenti.

- `@NoArgsConstructor`: Genera un costruttore senza argomenti.

**Campi**:

- `Long devicePosition`: Posizione del dispositivo nella classifica.

- `BigDecimal resultValue`: Valore del risultato delle prestazioni.

- `Long idPerformanceType`: ID del tipo di prestazione.

- `String deviceDescription`: Descrizione del dispositivo.

- `String deviceOperatingSystem`: Sistema operativo del dispositivo.

- `Long idPriceRange`: ID della fascia di prezzo.

**Costruttori:**

- Costruttore senza argomenti generato da Lombok (`@NoArgsConstructor`).

- Costruttore con tutti gli argomenti generato da Lombok (`@AllArgsConstructor`).

- Un costruttore personalizzato che accetta i campi in un ordine specifico.

Esempio della classe:

`java

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

public class DeviceProjection {

private Long devicePosition;

private BigDecimal resultValue;

private Long idPerformanceType;

private String deviceDescription;

private String deviceOperatingSystem;

private Long idPriceRange;

public DeviceProjection(Long idPerformanceType, BigDecimal resultValue,

String deviceDescription, String deviceOperatingSystem, Long idPriceRange, Long devicePosition ) {

this.idPerformanceType = idPerformanceType;

this.deviceDescription = deviceDescription;

this.deviceOperatingSystem = deviceOperatingSystem;

this.idPriceRange = idPriceRange;

this.resultValue = resultValue;

this.devicePosition = devicePosition;

}

}

``

## **Package** com.griglie.evaluatePerformanceApp



La classe `EvaluatePerformanceAppApplication` è la classe principale dell'applicazione Spring Boot. Serve come punto di ingresso per l'applicazione Spring Boot. Configura automaticamente l'applicazione, abilita la gestione delle transazioni, l'auditing JPA e specifica i pacchetti per le entità e i repository. Quando si esegue l'applicazione, Spring Boot avvierà il contesto dell'applicazione, configurerà i bean e avvierà il server integrato (ad esempio, Tomcat).

Di seguito una panoramica delle sue funzioni e annotazioni:

**Importazioni**:

- `org.springframework.boot.SpringApplication`: Classe principale di Spring Boot per avviare l'applicazione.

- `org.springframework.boot.autoconfigure.SpringBootApplication`: Annotazione per abilitare la configurazione automatica di Spring Boot.

- `org.springframework.boot.autoconfigure.domain.EntityScan`: Annotazione per scansionare pacchetti specifici per le entità JPA.

- `org.springframework.data.jpa.repository.config.EnableJpaAuditing`: Annotazione per abilitare l'auditing JPA.

- `org.springframework.data.jpa.repository.config.EnableJpaRepositories`: Annotazione per abilitare la creazione di repository JPA.

- `org.springframework.transaction.annotation.EnableTransactionManagement`: Annotazione per abilitare la gestione delle transazioni.

**Annotazioni:**

- `@SpringBootApplication`: Indica che questa è una classe di configurazione Spring Boot, abilitando l'autoconfigurazione, la scansione dei componenti e la configurazione delle proprietà.

- `@EnableTransactionManagement`: Abilita la gestione delle transazioni per l'applicazione.

- `@EnableJpaAuditing`: Abilita il supporto per l'auditing JPA, come la registrazione delle date di creazione e modifica delle entità.

- `@EntityScan("com.griglie.evaluatePerformanceApp.persistence.entities")`: Specifica il pacchetto in cui cercare le entità JPA.

- `@EnableJpaRepositories("com.griglie.evaluatePerformanceApp.persistence.repositories")`: Specifica il pacchetto in cui cercare i repository JPA.

**Metodo principale**

- `public static void main(String[] args)`: Metodo principale che avvia l'applicazione chiamando `SpringApplication.run()` con la classe `EvaluatePerformanceAppApplication` come argomento.

# EVALUATE PERFORMANCE APP ANDROID

Il progetto `evaluate\_performance\_app` è un'applicazione Flutter progettata per valutare le prestazioni. Questa relazione fornisce una panoramica dei file chiave analizzati finora, inclusi `analysis\_options.yaml`, `pubspec.yaml`, `widget\_test.dart`, e `index.html`, delineando la loro struttura, scopo e configurazione. Inoltre, viene fornita una descrizione generale dei progetti Flutter, dei file Dart e delle cartelle viste nel repository.

Flutter è un framework open-source sviluppato da Google per la creazione di applicazioni nativamente compilate per mobile, web e desktop da un singolo codice sorgente. Utilizza il linguaggio di programmazione Dart, anch'esso sviluppato da Google, per costruire interfacce utente reattive e ad alte prestazioni.

Un tipico progetto Flutter include diverse directory e file chiave:

- \*\*lib/\*\*: Contiene il codice sorgente dell'applicazione, solitamente organizzato in più file Dart.

- \*\*test/\*\*: Include i test unitari e widget per l'applicazione.

- \*\*web/\*\*: Contiene risorse specifiche per la versione web dell'applicazione.

- \*\*pubspec.yaml\*\*: File di configurazione che gestisce le dipendenze e le risorse del progetto.

- \*\*analysis\_options.yaml\*\*: File di configurazione per l'analizzatore di codice Dart.

I file Dart sono i componenti principali di un'applicazione Flutter. Essi possono essere suddivisi in varie categorie, come:

- \*\*Widgets\*\*: Costruiscono l'interfaccia utente.

- \*\*Models\*\*: Definiscono le strutture dati.

- \*\*Services\*\*: Gestiscono la logica di business e l'interazione con le API.

- \*\*Utilities\*\*: Funzioni e classi di supporto.

#### File Analizzati

1. \*\*analysis\_options.yaml\*\*

- \*\*Scopo\*\*: Configura l'analizzatore di codice Dart per identificare errori, avvisi e lints.

- \*\*Contenuto Principale\*\*:

- \*\*Inclusione di Regole Raccomandate\*\*: Include un set di regole di lint raccomandate per le app Flutter tramite `package:flutter\_lints/flutter.yaml`.

- \*\*Regole del Linter\*\*: Personalizza le regole del linter per il progetto.

- \*\*Esclusioni dell'Analizzatore\*\*: Esclude directory specifiche dall'analisi (`lib/custom\_code/\*\*` e `lib/flutter\_flow/custom\_functions.dart`).

2. \*\*pubspec.yaml\*\*

- \*\*Scopo\*\*: Configura le dipendenze, le risorse e altre impostazioni del progetto Flutter.

- \*\*Contenuto Principale\*\*:

- \*\*Nome e Descrizione del Progetto\*\*: `name: evaluate\_performance\_app`, `description: A new Flutter project`.

- \*\*Versione e Ambiente SDK\*\*: `version: 1.0.0+1`, `sdk: ">=3.0.0 <4.0.0"`.

- \*\*Dipendenze\*\*: Elenco delle dipendenze necessarie, come `flutter`, `flutter\_localizations`, `http`, `provider`, ecc.

- \*\*Override delle Dipendenze\*\*: Override di specifiche versioni di pacchetti.

- \*\*Dipendenze di Sviluppo\*\*: Include `flutter\_lints` e `flutter\_test`.

- \*\*Risorse e Configurazioni Specifiche di Flutter\*\*: Configura l'uso di `material-design` e specifica le risorse come immagini e font.

3. \*\*widget\_test.dart\*\*

- \*\*Scopo\*\*: Contiene test di base per i widget Flutter.

- \*\*Contenuto Principale\*\*:

- \*\*Importazioni\*\*: Importa `flutter\_test` e il file principale dell'app.

- \*\*Test di Base\*\*: Test di smoke per verificare che l'app possa essere costruita e visualizzata correttamente.

- \*\*Espansione del Test\*\*: Possibilità di aggiungere ulteriori interazioni e verifiche per rendere il test più completo.

4. \*\*index.html\*\*

- \*\*Scopo\*\*: Punto di ingresso principale per l'applicazione web.

- \*\*Contenuto Principale\*\*:

- \*\*Meta Tag e Base URL\*\*: Configura il charset, compatibilità con IE, configurazioni per app web mobili su iOS, e tag SEO.

- \*\*Favicon\*\*: Imposta l'icona del sito web.

- \*\*Manifest e Script di Flutter\*\*: Include il file di manifest e lo script di inizializzazione di Flutter.

- \*\*Script di Caricamento dell'Entrypoint di Flutter\*\*: Carica l'entrypoint di Flutter (`main.dart.js`) e avvia l'applicazione.

#### Cartelle Analizzate

1. \*\*web/\*\*:

- \*\*Scopo\*\*: Contiene le risorse specifiche per la versione web dell'applicazione.

- \*\*Contenuto Principale\*\*:

- \*\*favicon.png\*\*:

- Un file immagine utilizzato come icona del sito web.

- [favicon.png](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/web/favicon.png)

- \*\*icons/\*\*:

- Una directory che probabilmente contiene altre icone utilizzate nell'applicazione web.

- [icons](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/web/icons)

- \*\*index.html\*\*:

- Il file HTML principale che funge da punto di ingresso per l'applicazione web.

- [index.html](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/web/index.html)

#### Conclusioni

Il progetto `evaluate\_performance\_app` è ben strutturato con configurazioni dettagliate nei file chiave:

- \*\*analysis\_options.yaml\*\* garantisce che il codice segua le buone pratiche e sia privo di errori e avvisi.

- \*\*pubspec.yaml\*\* gestisce efficacemente le dipendenze e le risorse del progetto, assicurando che tutte le librerie necessarie siano incluse.

- \*\*widget\_test.dart\*\* fornisce una base per il testing automatizzato dei widget, migliorando l'affidabilità dell'app.

- \*\*index.html\*\* funge da punto di ingresso per l'applicazione web, configurando meta tag, favicon, manifest e script di inizializzazione.

- \*\*web/\*\* directory contiene le risorse specifiche per la versione web dell'applicazione, inclusi favicon, icone e il file HTML principale.

L'analisi di questi file suggerisce che il progetto è ben organizzato e segue le migliori pratiche per lo sviluppo di applicazioni Flutter. Se hai ulteriori domande o necessiti di chiarimenti aggiuntivi, non esitare a chiedere.

### Relazione sul Progetto `evaluate\_performance\_app`

#### Introduzione

Il progetto `evaluate\_performance\_app` è un'applicazione Flutter progettata per valutare le prestazioni. Questa relazione fornisce una panoramica dei file chiave analizzati finora, inclusi `analysis\_options.yaml`, `pubspec.yaml`, `widget\_test.dart`, e `index.html`, delineando la loro struttura, scopo e configurazione.

#### File Analizzati

1. \*\*analysis\_options.yaml\*\*

- \*\*Scopo\*\*: Configura l'analizzatore di codice Dart per identificare errori, avvisi e lints.

- \*\*Contenuto Principale\*\*:

- \*\*Inclusione di Regole Raccomandate\*\*: Include un set di regole di lint raccomandate per le app Flutter tramite `package:flutter\_lints/flutter.yaml`.

- \*\*Regole del Linter\*\*: Personalizza le regole del linter per il progetto.

- \*\*Esclusioni dell'Analizzatore\*\*: Esclude directory specifiche dall'analisi (`lib/custom\_code/\*\*` e `lib/flutter\_flow/custom\_functions.dart`).

2. \*\*pubspec.yaml\*\*

- \*\*Scopo\*\*: Configura le dipendenze, le risorse e altre impostazioni del progetto Flutter.

- \*\*Contenuto Principale\*\*:

- \*\*Nome e Descrizione del Progetto\*\*: `name: evaluate\_performance\_app`, `description: A new Flutter project`.

- \*\*Versione e Ambiente SDK\*\*: `version: 1.0.0+1`, `sdk: ">=3.0.0 <4.0.0"`.

- \*\*Dipendenze\*\*: Elenco delle dipendenze necessarie, come `flutter`, `flutter\_localizations`, `http`, `provider`, ecc.

- \*\*Override delle Dipendenze\*\*: Override di specifiche versioni di pacchetti.

- \*\*Dipendenze di Sviluppo\*\*: Include `flutter\_lints` e `flutter\_test`.

- \*\*Risorse e Configurazioni Specifiche di Flutter\*\*: Configura l'uso di `material-design` e specifica le risorse come immagini e font.

3. \*\*widget\_test.dart\*\*

- \*\*Scopo\*\*: Contiene test di base per i widget Flutter.

- \*\*Contenuto Principale\*\*:

- \*\*Importazioni\*\*: Importa `flutter\_test` e il file principale dell'app.

- \*\*Test di Base\*\*: Test di smoke per verificare che l'app possa essere costruita e visualizzata correttamente.

- \*\*Espansione del Test\*\*: Possibilità di aggiungere ulteriori interazioni e verifiche per rendere il test più completo.

4. \*\*index.html\*\*

- \*\*Scopo\*\*: Punto di ingresso principale per l'applicazione web.

- \*\*Contenuto Principale\*\*:

- \*\*Meta Tag e Base URL\*\*: Configura il charset, compatibilità con IE, configurazioni per app web mobili su iOS, e tag SEO.

- \*\*Favicon\*\*: Imposta l'icona del sito web.

- \*\*Manifest e Script di Flutter\*\*: Include il file di manifest e lo script di inizializzazione di Flutter.

- \*\*Script di Caricamento dell'Entrypoint di Flutter\*\*: Carica l'entrypoint di Flutter (`main.dart.js`) e avvia l'applicazione.

#### Conclusioni

Il progetto `evaluate\_performance\_app` è ben strutturato con configurazioni dettagliate nei file chiave:

- \*\*analysis\_options.yaml\*\* garantisce che il codice segua le buone pratiche e sia privo di errori e avvisi.

- \*\*pubspec.yaml\*\* gestisce efficacemente le dipendenze e le risorse del progetto, assicurando che tutte le librerie necessarie siano incluse.

- \*\*widget\_test.dart\*\* fornisce una base per il testing automatizzato dei widget, migliorando l'affidabilità dell'app.

- \*\*index.html\*\* funge da punto di ingresso per l'applicazione web, configurando meta tag, favicon, manifest e script di inizializzazione.

L'analisi di questi file suggerisce che il progetto è ben organizzato e segue le migliori pratiche per lo sviluppo di applicazioni Flutter. Se hai ulteriori domande o necessiti di chiarimenti aggiuntivi, non esitare a chiedere.

In un progetto Flutter, la cartella android contiene tutto il necessario per costruire e configurare l'app per la piattaforma Android. Ecco una panoramica dei principali file e cartelle che potresti trovare all'interno:

1. **app**: Questa cartella contiene il codice sorgente principale dell'app Android. Include sotto-cartelle come:
   * **src**: Contiene il codice Java o Kotlin dell'app.
     + **main**: La cartella principale del codice sorgente Android.
       - **AndroidManifest.xml**: Il file di manifest dell'app che definisce componenti fondamentali, permessi e altre configurazioni.
       - **java**: Contiene il codice sorgente Java (o Kotlin) dell'app.
       - **res**: Contiene le risorse dell'app come layout XML, stringhe, immagini e altro.
2. **gradle**: Contiene script di configurazione per il build system Gradle.
   * **wrapper**: Contiene i file per utilizzare una versione specifica di Gradle.
3. **build.gradle**: Ci sono due file build.gradle:
   * **A livello di progetto**: Configura le impostazioni globali del progetto.
   * **A livello di modulo/app**: Configura le impostazioni specifiche dell'app Android.
4. **gradle.properties**: File di configurazione delle proprietà di Gradle.
5. **local.properties**: Specifica il percorso dell'SDK Android sul computer locale.
6. **settings.gradle**: Include moduli nel progetto Gradle.

Questi sono i componenti chiave che troverai nella cartella android di un progetto Flutter.

Un progetto Flutter è una struttura organizzata per creare applicazioni mobili, web e desktop utilizzando il framework Flutter di Google. Flutter permette di sviluppare applicazioni cross-platform con un unico codice sorgente scritto principalmente in Dart. Ecco una panoramica della struttura tipica di un progetto Flutter:

### Struttura di un Progetto Flutter

1. \*\*`android/`\*\*:

- Contiene i file specifici per la piattaforma Android.

- \*\*`app/src`\*\*: Include il codice sorgente Java/Kotlin e le risorse dell'app Android.

- \*\*`build.gradle`\*\*: File di configurazione per Gradle, il sistema di build di Android.

2. \*\*`ios/`\*\*:

- Contiene i file specifici per la piattaforma iOS.

- \*\*`Runner.xcodeproj` e `Runner.xcworkspace`\*\*: Progetti e workspace Xcode.

- \*\*`AppDelegate.swift` e `Info.plist`\*\*: File di configurazione e inizializzazione per l'app iOS.

3. \*\*`lib/`\*\*:

- Contiene il codice sorgente principale dell'app Flutter.

- \*\*`main.dart`\*\*: Il punto di ingresso dell'app Flutter. Esempio di codice:

```dart

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

appBar: AppBar(

title: Text('Flutter Demo'),

),

body: Center(

child: Text('Hello, Flutter!'),

),

),

);

}

}

```

4. \*\*`test/`\*\*:

- Contiene i test unitari per l'app.

- \*\*`widget\_test.dart`\*\*: Un esempio di test per i widget Flutter.

5. \*\*`build/`\*\*:

- Directory generata automaticamente che contiene i file di build.

6. \*\*`pubspec.yaml`\*\*:

- File di configurazione del progetto Flutter.

- Specifica dipendenze, asset, font e altre configurazioni.

- Esempio di contenuto:

```yaml

name: my\_flutter\_app

description: A new Flutter project.

dependencies:

flutter:

sdk: flutter

cupertino\_icons: ^1.0.2

dev\_dependencies:

flutter\_test:

sdk: flutter

flutter:

uses-material-design: true

assets:

- assets/images/

```

7. \*\*`pubspec.lock`\*\*:

- File generato automaticamente che blocca le versioni delle dipendenze.

8. \*\*`README.md`\*\*:

- File di documentazione del progetto.

9. \*\*`.gitignore`\*\*:

- File di configurazione per Git che specifica quali file e cartelle ignorare.

### Creazione di un Progetto Flutter

Per creare un nuovo progetto Flutter, puoi utilizzare il comando:

```sh

flutter create my\_flutter\_app

```

Questo comando genera automaticamente la struttura del progetto come descritto sopra.

### Esecuzione del Progetto Flutter

Per eseguire l'app Flutter, puoi utilizzare i seguenti comandi:

- \*\*Android\*\*: `flutter run` (assicurati di avere un emulatore Android o un dispositivo connesso).

- \*\*iOS\*\*: `flutter run` (assicurati di avere un simulatore iOS o un dispositivo connesso e Xcode installato).

### Conclusione

Un progetto Flutter è ben organizzato con una chiara separazione tra codice sorgente, risorse specifiche della piattaforma e configurazioni di build. Questa struttura ti consente di sviluppare e gestire facilmente applicazioni cross-platform da un unico codice sorgente.

Il progetto Flutter nel repository [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app) è configurato attraverso il file `pubspec.yaml`. Questo file definisce le dipendenze del progetto, la configurazione dell'ambiente, le risorse utilizzate e altre impostazioni specifiche del progetto. Ecco una descrizione dettagliata del progetto basata sul contenuto del file `pubspec.yaml`:

### Dettagli del Progetto

- \*\*Nome:\*\* evaluate\_performance\_app

- \*\*Descrizione:\*\* A new Flutter project.

- \*\*Versione:\*\* 1.0.0+1

- \*\*Ambiente:\*\* Dart SDK versione >=3.0.0 <4.0.0

### Dipendenze Principali

Il progetto utilizza diverse dipendenze per aggiungere funzionalità specifiche:

- \*\*flutter:\*\* SDK di Flutter

- \*\*flutter\_localizations:\*\* Supporto per la localizzazione

- \*\*auto\_size\_text:\*\* Per il ridimensionamento automatico del testo

- \*\*cached\_network\_image:\*\* Per la gestione delle immagini memorizzate in cache

- \*\*data\_table\_2:\*\* Tabelle avanzate per Flutter

- \*\*equatable:\*\* Per comparazioni di oggetti

- \*\*flutter\_animate:\*\* Animazioni per Flutter

- \*\*flutter\_cache\_manager:\*\* Gestione della cache

- \*\*font\_awesome\_flutter:\*\* Icone FontAwesome

- \*\*google\_fonts:\*\* Utilizzo di Google Fonts

- \*\*http:\*\* Per le richieste HTTP

- \*\*intl:\*\* Per la gestione delle internazionalizzazioni

- \*\*provider:\*\* Gestione dello stato

- \*\*shared\_preferences:\*\* Per la memorizzazione delle preferenze dell'utente

- \*\*sqflite:\*\* Per l'accesso a database SQLite

- \*\*url\_launcher:\*\* Per aprire URL

### Override delle Dipendenze

- \*\*http:\*\* Versione 1.2.1

- \*\*uuid:\*\* Versione ^4.0.0

### Dipendenze di Sviluppo

- \*\*flutter\_lints:\*\* Linter per Flutter

- \*\*lints:\*\* Linter generico

- \*\*flutter\_test:\*\* SDK per i test con Flutter

### Risorse

Il progetto include diverse risorse come font, immagini, video, audio e animazioni:

- \*\*Font:\*\* Directory `assets/fonts/`

- \*\*Immagini:\*\* Directory `assets/images/`

- \*\*Video:\*\* Directory `assets/videos/`

- \*\*Audio:\*\* Directory `assets/audios/`

- \*\*Animazioni Lottie:\*\* Directory `assets/lottie\_animations/`

- \*\*Animazioni Rive:\*\* Directory `assets/rive\_animations/`

- \*\*PDF:\*\* Directory `assets/pdfs/`

### Configurazione Flutter

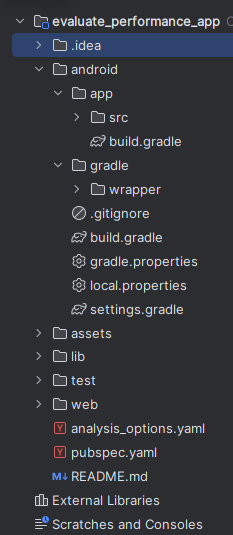
- \*\*uses-material-design:\*\* true (inclusione delle icone Material)

- \*\*assets:\*\* Definizione delle directory delle risorse

### Conclusione

Il progetto `evaluate\_performance\_app` è configurato per essere un'app nativa Flutter con supporto per localizzazione, gestione delle immagini e cache, animazioni, accesso a database, e molto altro. La configurazione delle dipendenze e delle risorse permette di sviluppare un'app completa e funzionale con supporto per diverse funzionalità e piattaforme.

Puoi esplorare ulteriormente il progetto visitando il [repository su GitHub](https://github.com/arizzo86/evaluate\_performance\_app).



I file con estensione .dart sono file sorgente scritti nel linguaggio di programmazione Dart. Dart è un linguaggio sviluppato da Google, utilizzato principalmente per lo sviluppo di applicazioni client, specialmente con il framework Flutter. Ecco alcune caratteristiche e utilizzi dei file .dart:

Android.app.build.grandle

Il file `build.gradle` presente nel percorso `android/app` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app) definisce la configurazione di build per l'applicazione Android. Ecco una descrizione dettagliata del file:

### Plugin

Utilizza i seguenti plugin:

- `com.android.application`: Plugin per applicazioni Android.

- `kotlin-android`: Plugin per il supporto di Kotlin.

- `dev.flutter.flutter-gradle-plugin`: Plugin per l'integrazione con Flutter.

### Proprietà Locali

Carica proprietà locali dai file `local.properties` e `key.properties`:

- `flutterVersionCode` e `flutterVersionName`: Versione dell'app Flutter.

- `keystoreProperties`: Proprietà per la firma dell'applicazione.

### Configurazione di Android

- \*\*`compileSdkVersion 34`\*\*: Specifica la versione SDK per la compilazione.

- \*\*`sourceSets`\*\*: Aggiunge la directory `src/main/kotlin` come sorgente Java.

- \*\*`lintOptions`\*\*: Disabilita il controllo dei pacchetti non validi e le verifiche delle build di release.

### Configurazione di Default

- \*\*`applicationId`\*\*: Identificatore univoco dell'applicazione (`com.griglie.evaluateperformanceapp`).

- \*\*`minSdkVersion 21`\*\*: Versione minima SDK supportata.

- \*\*`targetSdkVersion 34`\*\*: Versione target SDK.

- \*\*`versionCode`\*\*: Codice versione dell'app.

- \*\*`versionName`\*\*: Nome versione dell'app.

### Configurazione di Firma

- \*\*`signingConfigs`\*\*: Configurazione per la firma delle build di release.

### Tipi di Build

- \*\*`buildTypes`\*\*: Configura i tipi di build.

- `release`: Utilizza la configurazione di debug per la firma delle build di release (TODO: Aggiungere configurazione di firma per release).

### Flutter

- \*\*`flutter`\*\*: Specifica la directory sorgente di Flutter (`'../..'`).

### Dipendenze

- \*\*`dependencies`\*\*: Aggiunge la dipendenza per Kotlin:

- `implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.8.10"`

### Conclusione

Il file `build.gradle` configura la build dell'app Android, specificando le proprietà dell'applicazione, le versioni SDK, la configurazione di firma e le dipendenze necessarie. Se desideri ulteriori dettagli o modifiche specifiche, puoi esplorare direttamente il file nel [repository su GitHub](<https://github.com/arizzo86/evaluate_performance_app/blob/main/android/app/build.gradle>).

android/app/src/main

La directory `android/app/src/main` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app) contiene i file e le cartelle principali necessari per l'applicazione Android. Ecco una panoramica dei contenuti:

1. \*\*[AndroidManifest.xml](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/android/app/src/main/AndroidManifest.xml)\*\*

- Questo file di manifest definisce i componenti fondamentali dell'app Android, come attività, servizi e ricevitori di broadcast. Contiene anche informazioni come permessi richiesti dall'app.

2. \*\*[java/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/java)\*\*

- Questa directory contiene il codice sorgente Java dell'app. La struttura della cartella segue il namespace del pacchetto dell'app. Ad esempio, se il pacchetto dell'app è `com.example.myapp`, la cartella avrà il percorso `java/com/example/myapp/`.

3. \*\*[kotlin/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/kotlin)\*\*

- Questa directory contiene il codice sorgente Kotlin dell'app. Simile alla directory `java`, la struttura segue il namespace del pacchetto dell'app.

4. \*\*[res/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res)\*\*

- Questa directory contiene tutte le risorse dell'app, come layout XML, stringhe, immagini, stili e temi.

- \*\*layout/\*\*: Contiene i file di layout XML.

- \*\*values/\*\*: Contiene risorse come stringhe, colori e stili.

- \*\*drawable/\*\*: Contiene immagini e altre risorse grafiche.

### Conclusione

Questi file e cartelle formano la struttura di base del codice sorgente e delle risorse per l'app Android. Puoi esplorare ulteriormente i contenuti di ogni directory direttamente nel repository su GitHub utilizzando i link forniti.

android/app/src/main/res

La directory `android/app/src/main/res` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app) contiene diverse sottocartelle che organizzano le risorse dell'applicazione Android. Ecco una panoramica dei contenuti principali:

### Contenuti della Directory `res`

1. \*\*[drawable/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res/drawable)\*\*

- Contiene risorse grafiche come immagini e icone.

2. \*\*[mipmap-hdpi/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res/mipmap-hdpi)\*\*

- Contiene risorse mipmap ad alta densità per icone dell'applicazione.

3. \*\*[mipmap-mdpi/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res/mipmap-mdpi)\*\*

- Contiene risorse mipmap a densità media per icone dell'applicazione.

4. \*\*[mipmap-xhdpi/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res/mipmap-xhdpi)\*\*

- Contiene risorse mipmap a densità extra alta per icone dell'applicazione.

5. \*\*[mipmap-xxhdpi/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res/mipmap-xxhdpi)\*\*

- Contiene risorse mipmap a densità extra extra alta per icone dell'applicazione.

6. \*\*[mipmap-xxxhdpi/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res/mipmap-xxxhdpi)\*\*

- Contiene risorse mipmap a densità extra extra extra alta per icone dell'applicazione.

7. \*\*[values-night-v31/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res/values-night-v31)\*\*

- Contiene risorse di valori per la modalità notturna specifiche per la versione 31 e successive.

8. \*\*[values-night/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res/values-night)\*\*

- Contiene risorse di valori per la modalità notturna.

9. \*\*[values-v31/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res/values-v31)\*\*

- Contiene risorse di valori specifiche per la versione 31 e successive.

10. \*\*[values/](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/android/app/src/main/res/values)\*\*

- Contiene risorse di valori come stringhe, colori e stili.

### Ulteriori Dettagli

Puoi esplorare ulteriormente i contenuti di ogni directory direttamente nel repository su GitHub utilizzando i link forniti. Per visualizzare tutti i contenuti della directory `res`, visita [questa pagina](<https://github.com/arizzo86/evaluate_performance_app/tree/main/android/app/src/main/res>).

android/app/src/main/kotlin/com/example/my\_project

Nella directory `android/app/src/main/kotlin/com/example/my\_project` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app) è presente il file `MainActivity.kt`. Ecco il contenuto del file:

### [MainActivity.kt](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/android/app/src/main/kotlin/com/example/my\_project/MainActivity.kt)

```kotlin

package com.example.my\_project

import io.flutter.embedding.android.FlutterActivity

class MainActivity: FlutterActivity() {

}

```

### Descrizione

Il file `MainActivity.kt` definisce l'attività principale dell'applicazione Android utilizzando Kotlin. Eredita dalla classe `FlutterActivity` fornita dal framework Flutter. Questo permette all'attività di integrarsi con Flutter e di eseguire un'app Flutter all'interno di un'app Android.

- \*\*Package:\*\* `com.example.my\_project`

- \*\*Import:\*\* `io.flutter.embedding.android.FlutterActivity` - Importa la classe `FlutterActivity` necessaria per l'integrazione con Flutter.

- \*\*Class:\*\* `MainActivity` - La classe principale dell'attività che estende `FlutterActivity`.

Puoi esplorare ulteriormente il file direttamente su GitHub utilizzando [questo link](<https://github.com/arizzo86/evaluate_performance_app/blob/main/android/app/src/main/kotlin/com/example/my_project/MainActivity.kt>).

android/app/src/main/java/io/flutter/plugins

Nella directory `android/app/src/main/java/io/flutter/plugins` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app) è presente il file `GeneratedPluginRegistrant.java`. Ecco il contenuto del file:

### [GeneratedPluginRegistrant.java](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/android/app/src/main/java/io/flutter/plugins/GeneratedPluginRegistrant.java)

```java

package io.flutter.plugins;

import androidx.annotation.Keep;

import androidx.annotation.NonNull;

import io.flutter.embedding.engine.FlutterEngine;

import io.flutter.embedding.engine.plugins.shim.ShimPluginRegistry;

/\*\*

\* Generated file. Do not edit.

\*/

@Keep

public final class GeneratedPluginRegistrant {

public static void registerWith(@NonNull FlutterEngine flutterEngine) {

ShimPluginRegistry shimPluginRegistry = new ShimPluginRegistry(flutterEngine);

// Add your plugin registration code here

}

}

```

### Descrizione

Il file `GeneratedPluginRegistrant.java` è responsabile della registrazione dei plugin Flutter con il motore Flutter (`FlutterEngine`). Questo file è tipicamente generato automaticamente e non dovrebbe essere modificato manualmente. Ecco una panoramica delle sue componenti principali:

- \*\*Package:\*\* `io.flutter.plugins`

- \*\*Import:\*\* Importa annotazioni di Android e classi necessarie per la registrazione dei plugin.

- \*\*Class:\*\* `GeneratedPluginRegistrant`

- \*\*Metodo:\*\* `registerWith(@NonNull FlutterEngine flutterEngine)`

- Questo metodo utilizza `ShimPluginRegistry` per registrare i plugin con il motore Flutter.

### Annotazioni

- \*\*`@Keep`\*\*: Indica che il codice annotato non deve essere rimosso o rinominato durante la fase di ottimizzazione.

- \*\*`@NonNull`\*\*: Indica che l'argomento `flutterEngine` non può essere nullo.

Puoi esplorare ulteriormente il file direttamente su GitHub utilizzando [questo link](<https://github.com/arizzo86/evaluate_performance_app/blob/main/android/app/src/main/java/io/flutter/plugins/GeneratedPluginRegistrant.java>).

android/app/src/main/AndroidManifest.xml

Ecco il contenuto del file `AndroidManifest.xml` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/android/app/src/main/AndroidManifest.xml):

### Contenuto del File

```xml

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.griglie.evaluateperformanceapp"

xmlns:tools="http://schemas.android.com/tools">

<uses-permission android:name="android.permission.INTERNET"/>

<application

android:label="evaluatePerformanceApp"

tools:replace="android:label"

android:icon="@mipmap/ic\_launcher"

android:requestLegacyExternalStorage="true">

<activity

android:name=".MainActivity"

android:exported="true"

android:launchMode="singleTop"

android:theme="@style/LaunchTheme"

android:configChanges="orientation|keyboardHidden|keyboard|screenSize|smallestScreenSize|locale|layoutDirection|fontScale|screenLayout|density|uiMode"

android:hardwareAccelerated="true"

android:windowSoftInputMode="adjustResize">

<!-- Specifies an Android theme to apply to this Activity as soon as

the Android process has started. This theme is visible to the user

while the Flutter UI initializes. After that, this theme continues

to determine the Window background behind the Flutter UI. -->

<meta-data

android:name="io.flutter.embedding.android.NormalTheme"

android:resource="@style/NormalTheme"

/>

<!-- Displays an Android View that continues showing the launch screen

Drawable until Flutter paints its first frame, then this splash

screen fades out. A splash screen is useful to avoid any visual

gap between the end of Android's launch screen and the painting of

Flutter's first frame. -->

<meta-data

android:name="io.flutter.embedding.android.SplashScreenDrawable"

android:resource="@drawable/launch\_background"

/>

<intent-filter>

<action android:name="android.intent.action.MAIN"/>

<category android:name="android.intent.category.LAUNCHER"/>

</intent-filter>

<!-- Deep linking -->

<meta-data android:name="flutter\_deeplinking\_enabled" android:value="true" />

<intent-filter android:autoVerify="true">

<action android:name="android.intent.action.VIEW" />

<category android:name="android.intent.category.DEFAULT" />

<category android:name="android.intent.category.BROWSABLE" />

<data android:scheme="evaluateperformanceapp" android:host="evaluateperformanceapp.com" />

</intent-filter>

</activity>

<!-- Don't delete the meta-data below.

This is used by the Flutter tool to generate GeneratedPluginRegistrant.java -->

<meta-data

android:name="flutterEmbedding"

android:value="2" />

</application>

</manifest>

```

### Descrizione

- \*\*Permessi\*\*

- `<uses-permission android:name="android.permission.INTERNET"/>`: Permette all'app di accedere a Internet.

- \*\*Applicazione\*\*

- \*\*`android:label`\*\*: Imposta l'etichetta dell'app come "evaluatePerformanceApp".

- \*\*`android:icon`\*\*: Imposta l'icona dell'app.

- \*\*`android:requestLegacyExternalStorage`\*\*: Richiede l'uso dell'archiviazione legacy.

- \*\*Attività Principale (`MainActivity`)\*\*

- \*\*`android:name`\*\*: Specifica il nome della classe attività principale.

- \*\*`android:exported`\*\*: Permette l'esportazione dell'attività.

- \*\*`android:launchMode`\*\*: Imposta la modalità di lancio `singleTop`.

- \*\*`android:theme`\*\*: Imposta il tema di lancio.

- \*\*`android:configChanges`\*\*: Specifica le modifiche di configurazione gestite dall'attività.

- \*\*`android:hardwareAccelerated`\*\*: Abilita l'accelerazione hardware.

- \*\*`android:windowSoftInputMode`\*\*: Imposta la modalità del soft input.

- \*\*Meta-dati\*\*

- \*\*`io.flutter.embedding.android.NormalTheme`\*\*: Specifica il tema normale.

- \*\*`io.flutter.embedding.android.SplashScreenDrawable`\*\*: Specifica il drawable della schermata di avvio.

- \*\*Intent Filters\*\*

- \*\*`android.intent.action.MAIN`\*\* e \*\*`android.intent.category.LAUNCHER`\*\*: Definiscono l'intento principale e la categoria di avvio.

- \*\*Deep Linking\*\*: Abilita il deep linking per l'app.

Puoi visualizzare ulteriormente il file direttamente su GitHub utilizzando [questo link](<https://github.com/arizzo86/evaluate_performance_app/blob/main/android/app/src/main/AndroidManifest.xml>).

android/app/build.gradle

Ecco il contenuto del file `build.gradle` per il modulo `android/app` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/android/app/build.gradle):

### Contenuto del File

```gradle

plugins {

id "com.android.application"

id "kotlin-android"

id "dev.flutter.flutter-gradle-plugin"

}

def localProperties = new Properties()

def localPropertiesFile = rootProject.file('local.properties')

if (localPropertiesFile.exists()) {

localPropertiesFile.withReader('UTF-8') { reader ->

localProperties.load(reader)

}

}

def flutterVersionCode = localProperties.getProperty('flutter.versionCode')

if (flutterVersionCode == null) {

flutterVersionCode = '1'

}

def flutterVersionName = localProperties.getProperty('flutter.versionName')

if (flutterVersionName == null) {

flutterVersionName = '1.0'

}

def keystoreProperties = new Properties()

def keystorePropertiesFile = rootProject.file('key.properties')

if (keystorePropertiesFile.exists()) {

keystoreProperties.load(new FileInputStream(keystorePropertiesFile))

}

android {

compileSdkVersion 34

sourceSets {

main.java.srcDirs += 'src/main/kotlin'

}

lintOptions {

disable 'InvalidPackage'

checkReleaseBuilds false

}

defaultConfig {

// TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id.html).

applicationId "com.griglie.evaluateperformanceapp"

minSdkVersion 21

targetSdkVersion 34

versionCode flutterVersionCode.toInteger()

versionName flutterVersionName

}

signingConfigs {

release {

keyAlias keystoreProperties['keyAlias']

keyPassword keystoreProperties['keyPassword']

storeFile keystoreProperties['storeFile'] ? file(keystoreProperties['storeFile']) : null

storePassword keystoreProperties['storePassword']

}

}

buildTypes {

release {

// TODO: Add your own signing config for the release build.

// Signing with the debug keys for now, so `flutter run --release` works.

signingConfig signingConfigs.debug

}

}

}

flutter {

source '../..'

}

dependencies {

implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.8.10"

}

```

### Descrizione

#### Plugins

- \*\*`com.android.application`\*\*: Plugin necessario per le applicazioni Android.

- \*\*`kotlin-android`\*\*: Plugin per supportare Kotlin in Android.

- \*\*`dev.flutter.flutter-gradle-plugin`\*\*: Plugin per supportare Flutter in Android.

#### Properties

- \*\*`localProperties`\*\*: Carica proprietà locali dal file `local.properties`.

- \*\*`flutterVersionCode` e `flutterVersionName`\*\*: Definiscono la versione dell'app Flutter.

- \*\*`keystoreProperties`\*\*: Carica proprietà del keystore dal file `key.properties`.

#### Configurazioni Android

- \*\*`compileSdkVersion`\*\*: Versione SDK di compilazione impostata a 34.

- \*\*`sourceSets`\*\*: Include directory Kotlin.

- \*\*`lintOptions`\*\*: Disabilita il controllo `InvalidPackage` e disabilita i controlli per le build di rilascio.

- \*\*`defaultConfig`\*\*:

- \*\*`applicationId`\*\*: ID unico dell'applicazione.

- \*\*`minSdkVersion`\*\*: Versione minima SDK impostata a 21.

- \*\*`targetSdkVersion`\*\*: Versione target SDK impostata a 34.

- \*\*`versionCode` e `versionName`\*\*: Versioni dell'applicazione.

#### Configurazione di Firma

- \*\*`signingConfigs`\*\*: Configura la firma per la build di rilascio utilizzando le proprietà del keystore.

#### Tipi di Build

- \*\*`release`\*\*: Configura la build di rilascio utilizzando la firma di debug per ora.

#### Configurazione Flutter

- \*\*`flutter`\*\*: Specifica la directory sorgente di Flutter.

#### Dipendenze

- \*\*`implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.8.10"`\*\*: Aggiunge la dipendenza della libreria standard di Kotlin.

Puoi visualizzare ulteriormente il file direttamente su GitHub utilizzando [questo link](<https://github.com/arizzo86/evaluate_performance_app/blob/main/android/app/build.gradle>).

android/gradle/wrapper/gradle-wrapper.properties

Ecco il contenuto del file `gradle-wrapper.properties` per il modulo `android/gradle/wrapper` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/android/gradle/wrapper/gradle-wrapper.properties):

### Contenuto del File

```properties

#Fri Jun 23 08:50:38 CEST 2017

distributionBase=GRADLE\_USER\_HOME

distributionPath=wrapper/dists

zipStoreBase=GRADLE\_USER\_HOME

zipStorePath=wrapper/dists

distributionUrl=https\://services.gradle.org/distributions/gradle-7.5-all.zip

```

### Descrizione

- \*\*`distributionBase`\*\*: Specifica la base sulla quale è memorizzata la distribuzione di Gradle. In questo caso, è impostata su `GRADLE\_USER\_HOME`.

- \*\*`distributionPath`\*\*: Specifica il percorso della distribuzione di Gradle all'interno della base specificata.

- \*\*`zipStoreBase`\*\*: Indica la base dove sono memorizzati i file zip della distribuzione di Gradle. Anche questo è impostato su `GRADLE\_USER\_HOME`.

- \*\*`zipStorePath`\*\*: Specifica il percorso dei file zip della distribuzione all'interno della zipStoreBase.

- \*\*`distributionUrl`\*\*: URL da cui scaricare la distribuzione di Gradle. In questo caso, punta alla versione 7.5 di Gradle.

Puoi visualizzare ulteriormente il file direttamente su GitHub utilizzando [questo link](<https://github.com/arizzo86/evaluate_performance_app/blob/main/android/gradle/wrapper/gradle-wrapper.properties>).

android/build.gradle

Ecco il contenuto del file `build.gradle` per il modulo `android` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/android/build.gradle):

### Contenuto del File

```gradle

allprojects {

repositories {

google()

mavenCentral()

}

}

rootProject.buildDir = '../build'

subprojects {

project.buildDir = "${rootProject.buildDir}/${project.name}"

}

subprojects {

project.evaluationDependsOn(':app')

}

task clean(type: Delete) {

delete rootProject.buildDir

}

```

### Descrizione

- \*\*`allprojects`\*\*: Configura i repository per tutte le sottoprogetti. In questo caso, utilizza `google()` e `mavenCentral()` per gestire le dipendenze.

- \*\*`rootProject.buildDir`\*\*: Imposta la directory di build del progetto root a `../build`.

- \*\*`subprojects`\*\*:

- \*\*`project.buildDir`\*\*: Configura la directory di build per ciascun sottoprogetto, concatenando la directory di build del progetto root con il nome del sottoprogetto.

- \*\*`project.evaluationDependsOn(':app')`\*\*: Assicura che il progetto `:app` venga valutato prima degli altri sottoprogetti.

- \*\*`task clean(type: Delete)`\*\*: Definisce un task `clean` per eliminare la directory di build del progetto root.

Puoi visualizzare ulteriormente il file direttamente su GitHub utilizzando [questo link](<https://github.com/arizzo86/evaluate_performance_app/blob/main/android/build.gradle>).

android/gradle.properties

Ecco il contenuto del file `gradle.properties` per il modulo `android` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/android/gradle.properties):

### Contenuto del File

```ini

org.gradle.jvmargs=-Xmx4608m

android.useAndroidX=true

android.enableJetifier=true

android.enableR8=true

```

### Descrizione

- \*\*`org.gradle.jvmargs=-Xmx4608m`\*\*: Configura la JVM per utilizzare un massimo di 4608 MB di memoria.

- \*\*`android.useAndroidX=true`\*\*: Abilita l'uso di AndroidX.

- \*\*`android.enableJetifier=true`\*\*: Abilita il Jetifier per la compatibilità con le vecchie librerie.

- \*\*`android.enableR8=true`\*\*: Abilita R8 per il minification e l'ottimizzazione del codice.

android/local.properties

Ecco il contenuto del file `local.properties` per il modulo `android` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/android/local.properties):

### Contenuto del File

```ini

sdk.dir=/opt/android-sdk

flutter.sdk=/sdks/flutter

```

### Descrizione

- \*\*`sdk.dir`\*\*: Specifica la directory dell'SDK di Android. In questo caso, è impostata su `/opt/android-sdk`.

- \*\*`flutter.sdk`\*\*: Specifica la directory dell'SDK di Flutter. In questo caso, è impostata su `/sdks/flutter`.

Puoi visualizzare ulteriormente il file direttamente su GitHub utilizzando [questo link](<https://github.com/arizzo86/evaluate_performance_app/blob/main/android/local.properties>).

android/settings.gradle

Ecco il contenuto del file `settings.gradle` per il modulo `android` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/android/settings.gradle):

### Contenuto del File

```gradle

pluginManagement {

def flutterSdkPath = {

def properties = new Properties()

file("local.properties").withInputStream { properties.load(it) }

def flutterSdkPath = properties.getProperty("flutter.sdk")

assert flutterSdkPath != null, "flutter.sdk not set in local.properties"

return flutterSdkPath

}()

includeBuild("$flutterSdkPath/packages/flutter\_tools/gradle")

repositories {

google()

mavenCentral()

gradlePluginPortal()

}

}

plugins {

id "dev.flutter.flutter-plugin-loader" version "1.0.0"

id "com.android.application" version "7.3.0" apply false

id "org.jetbrains.kotlin.android" version "1.8.10" apply false

}

include ":app"

```

### Descrizione

- \*\*`pluginManagement`\*\*: Configura la gestione dei plugin.

- \*\*`flutterSdkPath`\*\*: Carica il percorso dell'SDK di Flutter dal file `local.properties`.

- \*\*`includeBuild`\*\*: Include il build path per gli strumenti Flutter.

- \*\*`repositories`\*\*: Specifica i repository per i plugin (`google()`, `mavenCentral()`, `gradlePluginPortal()`).

- \*\*`plugins`\*\*:

- \*\*`dev.flutter.flutter-plugin-loader`\*\*: Plugin per caricare i plugin di Flutter.

- \*\*`com.android.application`\*\*: Plugin per applicazioni Android, versione 7.3.0.

- \*\*`org.jetbrains.kotlin.android`\*\*: Plugin per supportare Kotlin in Android, versione 1.8.10.

- \*\*`include ":app"`\*\*: Include il modulo `:app` nel progetto.

Puoi visualizzare ulteriormente il file direttamente su GitHub utilizzando [questo link](<https://github.com/arizzo86/evaluate_performance_app/blob/main/android/settings.gradle>).

[**evaluate\_performance\_app**](https://github.com/arizzo86/evaluate_performance_app/tree/main)/assets

La directory `assets` di solito contiene risorse statiche utilizzate dall'applicazione, come immagini, file audio, font, file di configurazione, ecc. Questi file sono inclusi nel pacchetto dell'applicazione e possono essere utilizzati direttamente dal codice dell'app.

Di seguito una descrizione generale dei tipi di file che potresti trovare nella directory `assets`:

- \*\*Immagini (es. .png, .jpg)\*\*: Utilizzate per le interfacce grafiche, icone, sfondi, ecc.

- \*\*File audio (es. .mp3, .wav)\*\*: Utilizzati per effetti sonori, musica di sottofondo, notifiche sonore, ecc.

- \*\*Font (es. .ttf, .otf)\*\*: Utilizzati per personalizzare i caratteri dell'interfaccia utente.

- \*\*File di configurazione (es. .json, .xml)\*\*: Utilizzati per configurazioni varie dell'applicazione, come impostazioni di gioco, configurazioni di rete, ecc.

- \*\*Altri file statici\*\*: Qualsiasi altra risorsa statica che l'applicazione potrebbe necessitare.

Se desideri una descrizione specifica dei file presenti nella tua directory `assets`, per favore fornisci i nomi dei file o ulteriori dettagli su di essi.

[**evaluate\_performance\_app**](https://github.com/arizzo86/evaluate_performance_app/tree/main)/lib

La directory `lib` in un progetto Flutter contiene solitamente tutto il codice sorgente dell'applicazione. Ecco una descrizione generale dei tipi di file che potresti trovare all'interno della directory `lib` e a cosa servono:

1. \*\*`main.dart`\*\*: Questo è il file di ingresso principale dell'applicazione Flutter. Contiene il metodo `main()` che avvia l'app e definisce il widget radice.

2. \*\*Cartelle Organizzative\*\*:

- \*\*`screens` o `pages`\*\*: Contiene i diversi file Dart che rappresentano le schermate dell'applicazione.

- \*\*`widgets`\*\*: Contiene widget personalizzati che sono utilizzati in più parti dell'applicazione.

- \*\*`models`\*\*: Contiene le classi di modelli che rappresentano i dati dell'applicazione.

- \*\*`services` o `providers`\*\*: Contiene servizi per la gestione della logica di business, come l'interazione con le API o la gestione dello stato.

- \*\*`utils` o `helpers`\*\*: Contiene funzioni di utilità e helper che possono essere utilizzate in tutta l'applicazione.

- \*\*`constants`\*\*: Contiene costanti utilizzate nell'app, come stringhe di testo, colori, ecc.

3. \*\*File di Configurazione\*\*:

- \*\*`routes.dart`\*\*: Gestisce la navigazione tra le diverse schermate dell'applicazione.

- \*\*`theme.dart`\*\*: Definisce il tema dell'applicazione, inclusi colori, stili di testo, ecc.

### Esempio di Struttura della Directory `lib`

```

lib/

├── main.dart

├── screens/

│ ├── home\_screen.dart

│ └── details\_screen.dart

├── widgets/

│ ├── custom\_button.dart

│ └── custom\_card.dart

├── models/

│ └── user\_model.dart

├── services/

│ └── api\_service.dart

├── utils/

│ └── date\_helper.dart

├── constants/

│ └── app\_constants.dart

├── routes.dart

└── theme.dart

```

### Descrizione dei File

- \*\*`main.dart`\*\*: Punto di ingresso dell'applicazione.

- \*\*`home\_screen.dart`, `details\_screen.dart`\*\*: Schermate dell'applicazione.

- \*\*`custom\_button.dart`, `custom\_card.dart`\*\*: Widget personalizzati.

- \*\*`user\_model.dart`\*\*: Classe di modello per rappresentare un utente.

- \*\*`api\_service.dart`\*\*: Servizio per gestire le chiamate API.

- \*\*`date\_helper.dart`\*\*: Funzioni di utilità per la gestione delle date.

- \*\*`app\_constants.dart`\*\*: Costanti dell'applicazione.

- \*\*`routes.dart`\*\*: Gestione delle rotte di navigazione.

- \*\*`theme.dart`\*\*: Definizione del tema dell'applicazione.

Se hai bisogno di informazioni specifiche su uno o più file all'interno della directory `lib` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/lib), fammi sapere!

Lib/flutter\_flow

La directory `lib/flutter\_flow` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app) contiene i seguenti file e directory:

1. \*\*[flutter\_flow\_data\_table.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/flutter\_flow/flutter\_flow\_data\_table.dart)\*\* (11,132 bytes)

2. \*\*[flutter\_flow\_model.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/flutter\_flow/flutter\_flow\_model.dart)\*\* (5,445 bytes)

3. \*\*[flutter\_flow\_theme.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/flutter\_flow/flutter\_flow\_theme.dart)\*\* (12,585 bytes)

4. \*\*[flutter\_flow\_util.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/flutter\_flow/flutter\_flow\_util.dart)\*\* (12,099 bytes)

5. \*\*[flutter\_flow\_widgets.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/flutter\_flow/flutter\_flow\_widgets.dart)\*\* (8,835 bytes)

6. \*\*[lat\_lng.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/flutter\_flow/lat\_lng.dart)\*\* (457 bytes)

7. \*\*[nav](https://github.com/arizzo86/evaluate\_performance\_app/tree/main/lib/flutter\_flow/nav)\*\* (directory)

8. \*\*[place.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/flutter\_flow/place.dart)\*\* (967 bytes)

9. \*\*[uploaded\_file.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/flutter\_flow/uploaded\_file.dart)\*\* (1,701 bytes)

### Descrizione Generica dei File

- \*\*`flutter\_flow\_data\_table.dart`\*\*: Probabilmente contiene la definizione e la gestione di tabelle di dati utilizzate nell'app.

- \*\*`flutter\_flow\_model.dart`\*\*: Probabilmente contiene modelli di dati utilizzati nell'app.

- \*\*`flutter\_flow\_theme.dart`\*\*: Contiene le definizioni del tema dell'app, inclusi colori, stili, ecc.

- \*\*`flutter\_flow\_util.dart`\*\*: Contiene funzioni di utilità varie utilizzate nell'app.

- \*\*`flutter\_flow\_widgets.dart`\*\*: Contiene widget personalizzati utilizzati nell'app.

- \*\*`lat\_lng.dart`\*\*: Probabilmente contiene classi o funzioni per la gestione di coordinate geografiche (latitudine e longitudine).

- \*\*`nav`\*\*: Directory che potrebbe contenere file relativi alla navigazione dell'app.

- \*\*`place.dart`\*\*: Probabilmente contiene classi o funzioni per la gestione di luoghi o posizioni.

- \*\*`uploaded\_file.dart`\*\*: Probabilmente contiene classi o funzioni per la gestione di file caricati.

### Utilità dei File

Questi file formano la base dell'infrastruttura dell'applicazione, gestendo vari aspetti come il tema, i widget personalizzati, i dati di navigazione, e le utilità comuni. Questi componenti sono essenziali per costruire un'app Flutter ben strutturata e funzionale.

Se hai bisogno di dettagli specifici su uno di questi file, fammi sapere!

lib/homepage

La directory `lib/homepage` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app) contiene i seguenti file:

1. \*\*[homepage\_model.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/homepage/homepage\_model.dart)\*\* (605 bytes)

2. \*\*[homepage\_widget.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/homepage/homepage\_widget.dart)\*\* (19,631 bytes)

### Descrizione dei File

- \*\*`homepage\_model.dart`\*\*: Questo file probabilmente contiene il modello di dati per la homepage. I modelli di dati sono utilizzati per rappresentare e gestire i dati che vengono visualizzati o elaborati nella schermata della homepage.

- \*\*`homepage\_widget.dart`\*\*: Questo file probabilmente contiene il widget della homepage. I widget in Flutter sono componenti che rappresentano le varie parti dell'interfaccia utente. Questo file potrebbe includere la logica per costruire e visualizzare la homepage dell'app.

### Utilità dei File

- \*\*`homepage\_model.dart`\*\*: Definisce come i dati della homepage sono strutturati e fornisce metodi per manipolare tali dati.

- \*\*`homepage\_widget.dart`\*\*: Costruisce l'interfaccia utente della homepage, definendo layout, stili e interazioni dell'utente.

Se hai bisogno di ulteriori dettagli su uno di questi file, fammi sapere!

Api\_request

La directory `lib/backend/api\_requests` del progetto [evaluate\_performance\_app](https://github.com/arizzo86/evaluate\_performance\_app) contiene i seguenti file:

1. \*\*[api\_calls.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/backend/api\_requests/api\_calls.dart)\*\* (1,813 bytes)

2. \*\*[api\_manager.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/backend/api\_requests/api\_manager.dart)\*\* (15,211 bytes)

3. \*\*[get\_streamed\_response.dart](https://github.com/arizzo86/evaluate\_performance\_app/blob/main/lib/backend/api\_requests/get\_streamed\_response.dart)\*\* (127 bytes)

### Descrizione dei File

- \*\*`api\_calls.dart`\*\*: Probabilmente contiene definizioni di funzioni per effettuare chiamate API specifiche. Queste funzioni sono utilizzate per interagire con servizi web esterni, inviare richieste HTTP e gestire le risposte.

- \*\*`api\_manager.dart`\*\*: Probabilmente contiene la logica per la gestione delle chiamate API, inclusa la configurazione delle richieste, la gestione delle risposte e degli errori, e forse l'autenticazione. Questo file potrebbe anche gestire la configurazione delle librerie per le richieste HTTP.

- \*\*`get\_streamed\_response.dart`\*\*: Probabilmente contiene una funzione o una classe per ottenere risposte in streaming da un'API. Questo tipo di risposte è utile quando si devono gestire grandi quantità di dati o dati in tempo reale.

### Utilità dei File

- \*\*`api\_calls.dart`\*\*: Fornisce una serie di funzioni pronte all'uso per effettuare chiamate API specifiche, migliorando la modularità e la riusabilità del codice.

- \*\*`api\_manager.dart`\*\*: Centralizza la gestione delle chiamate API, rendendo il codice più organizzato e mantenibile. Gestisce la configurazione delle richieste e l'elaborazione delle risposte.

- \*\*`get\_streamed\_response.dart`\*\*: Gestisce le risposte in streaming dalle API, utile per applicazioni che richiedono aggiornamenti in tempo reale o che trattano grandi quantità di dati.

Se hai bisogno di ulteriori dettagli su uno di questi file, fammi sapere!

lib/main.dart

Il file `main.dart` è il punto di ingresso principale dell'applicazione Flutter. Ecco una panoramica del suo contenuto e del suo scopo:

### Descrizione del Codice

1. \*\*Importazioni (linee 1-7)\*\*:

- Vengono importati i pacchetti necessari, inclusi `flutter/material.dart` per i widget di Material Design, `flutter\_localizations` per il supporto delle localizzazioni, `flutter\_web\_plugins/url\_strategy` per la gestione delle URL nel web, e vari file del progetto (`flutter\_flow\_theme.dart`, `flutter\_flow\_util.dart`, `nav.dart`).

2. \*\*Funzione `main` (linee 9-17)\*\*:

- La funzione `main` è il punto di ingresso dell'applicazione.

- `WidgetsFlutterBinding.ensureInitialized()` assicura che i widget siano inizializzati.

- `GoRouter.optionURLReflectsImperativeAPIs = true` e `usePathUrlStrategy()` configurano la strategia delle URL.

- `FlutterFlowTheme.initialize()` inizializza il tema dell'applicazione.

- `runApp(const MyApp())` avvia l'applicazione con il widget `MyApp`.

3. \*\*Classe `MyApp` (linee 19-28)\*\*:

- `MyApp` è un widget Stateful che rappresenta l'intera applicazione.

- La funzione statica `of` facilita l'accesso allo stato del widget.

4. \*\*Classe `\_MyAppState` (linee 30-70)\*\*:

- Gestisce lo stato del widget `MyApp`.

- `ThemeMode` viene utilizzato per gestire il tema (chiaro o scuro).

- `AppStateNotifier` e `GoRouter` vengono inizializzati per gestire lo stato dell'applicazione e la navigazione.

- La funzione `setThemeMode` permette di cambiare il tema e salvarlo.

- Il metodo `build` costruisce l'interfaccia utente principale:

- `MaterialApp.router` viene utilizzato per configurare l'app con il router e le localizzazioni.

- `theme` e `darkTheme` definiscono i temi chiaro e scuro.

- `themeMode` determina quale tema applicare.

- `routerConfig` configura la navigazione dell'app.

### Esempio di Utilizzo

Questo file configura e avvia l'applicazione Flutter, gestendo il tema e la navigazione. Fornisce una struttura di base su cui costruire ulteriori funzionalità e interfacce utente.

Se hai ulteriori domande o hai bisogno di dettagli su una specifica parte del codice, fammi sapere!

Index.dart

Il file `index.dart` è utilizzato per esportare altri moduli o file all'interno del progetto. In questo caso, il file `index.dart` esporta la schermata della homepage.

### Contenuto del file `index.dart`

```dart

// Export pages

export '/homepage/homepage\_widget.dart' show HomepageWidget;

```

### Descrizione

- \*\*`export '/homepage/homepage\_widget.dart' show HomepageWidget;`\*\*: Questa linea esporta solo il widget `HomepageWidget` dal file `homepage\_widget.dart` situato nella directory `homepage`. Utilizzando `export`, è possibile rendere disponibile `HomepageWidget` per l'importazione in altri file del progetto.

### Utilità

L'utilizzo di un file di indice (`index.dart`) per gestire le esportazioni può semplificare l'importazione di moduli in altre parti del progetto. Invece di dover specificare il percorso completo di ogni file, è possibile importare direttamente dall'indice:

```dart

import 'package:your\_project\_name/lib/index.dart';

```

Se hai bisogno di ulteriori dettagli o chiarimenti, fammi sapere!

Il file `analysis\_options.yaml` è utilizzato per configurare l'analizzatore di codice Dart, che controlla staticamente il codice per individuare errori, avvisi e regole di stile (lints). Ecco una panoramica del contenuto e delle sue parti principali:

### Contenuto del file `analysis\_options.yaml`

```yaml

# This file configures the analyzer, which statically analyzes Dart code to

# check for errors, warnings, and lints.

#

# The issues identified by the analyzer are surfaced in the UI of Dart-enabled

# IDEs (https://dart.dev/tools#ides-and-editors). The analyzer can also be

# invoked from the command line by running `flutter analyze`.

# The following line activates a set of recommended lints for Flutter apps,

# packages, and plugins designed to encourage good coding practices.

include: package:flutter\_lints/flutter.yaml

linter:

# The lint rules applied to this project can be customized in the

# section below to disable rules from the `package:flutter\_lints/flutter.yaml`

# included above or to enable additional rules. A list of all available lints

# and their documentation is published at https://dart.dev/lints.

#

# Instead of disabling a lint rule for the entire project in the

# section below, it can also be suppressed for a single line of code

# or a specific dart file by using the `// ignore: name\_of\_lint` and

# `// ignore\_for\_file: name\_of\_lint` syntax on the line or in the file

# producing the lint.

rules:

# avoid\_print: false # Uncomment to disable the `avoid\_print` rule

# prefer\_single\_quotes: true # Uncomment to enable the `prefer\_single\_quotes` rule

unnecessary\_string\_escapes: false

# Additional information about this file can be found at

# https://dart.dev/guides/language/analysis-options

analyzer:

exclude:

- lib/custom\_code/\*\*

- lib/flutter\_flow/custom\_functions.dart

```

### Descrizione delle Parti Principali

1. \*\*Inclusione dei Lints Raccomandati\*\*:

```yaml

include: package:flutter\_lints/flutter.yaml

```

Questa linea include un insieme di regole di lint raccomandate per le app Flutter, progettate per incoraggiare buone pratiche di codifica.

2. \*\*Configurazione del Linter\*\*:

```yaml

linter:

rules:

# avoid\_print: false # Uncomment to disable the `avoid\_print` rule

# prefer\_single\_quotes: true # Uncomment to enable the `prefer\_single\_quotes` rule

unnecessary\_string\_escapes: false

```

La sezione `linter` permette di personalizzare le regole di lint applicate al progetto. È possibile disabilitare o abilitare specifiche regole di lint. Ad esempio, la regola `unnecessary\_string\_escapes` è disabilitata (`false`).

3. \*\*Esclusioni dell'Analizzatore\*\*:

```yaml

analyzer:

exclude:

- lib/custom\_code/\*\*

- lib/flutter\_flow/custom\_functions.dart

```

La sezione `analyzer` permette di escludere specifici file o directory dall'analisi. In questo caso, i file nella directory `lib/custom\_code` e il file `lib/flutter\_flow/custom\_functions.dart` sono esclusi dall'analisi.

### Utilità del file `analysis\_options.yaml`

Il file `analysis\_options.yaml` è fondamentale per mantenere un codice di alta qualità, aiutando a identificare problemi potenziali e a seguire le migliori pratiche di codifica. Configurare correttamente l'analizzatore può migliorare la leggibilità del codice, la sua manutenibilità e ridurre il numero di bug.

Se hai altre domande o necessiti di ulteriori chiarimenti, fammi sapere!

Il file `pubspec.yaml` è un file di configurazione utilizzato in progetti Flutter per specificare le dipendenze, le risorse e altre configurazioni del progetto. Ecco una panoramica del suo contenuto:

### Contenuto del file `pubspec.yaml`

```yaml

name: evaluate\_performance\_app

description: A new Flutter project.

# The following line prevents the package from being accidentally published to

# pub.dev using `pub publish`. This is preferred for private packages.

publish\_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.

# A version number is three numbers separated by dots, like 1.2.43

# followed by an optional build number separated by a +.

# Both the version and the builder number may be overridden in flutter

# build by specifying --build-name and --build-number, respectively.

# In Android, build-name is used as versionName while build-number used as versionCode.

# Read more about Android versioning at https://developer.android.com/studio/publish/versioning

# In iOS, build-name is used as CFBundleShortVersionString while build-number used as CFBundleVersion.

# Read more about iOS versioning at

# https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.html

version: 1.0.0+1

environment:

sdk: ">=3.0.0 <4.0.0"

dependencies:

flutter:

sdk: flutter

flutter\_localizations:

sdk: flutter

auto\_size\_text: 3.0.0

cached\_network\_image: 3.3.1

collection: 1.18.0

data\_table\_2: 2.5.10

equatable: 2.0.5

flutter\_animate: 4.5.0

flutter\_cache\_manager: 3.3.1

font\_awesome\_flutter: 10.6.0

from\_css\_color: 2.0.0

go\_router: 12.1.3

google\_fonts: 6.1.0

http: 1.2.1

intl: 0.19.0

json\_path: 0.7.2

mime\_type: 1.0.0

page\_transition: 2.1.0

path\_provider: 2.1.3

path\_provider\_android: 2.2.5

path\_provider\_foundation: 2.4.0

path\_provider\_platform\_interface: 2.1.2

provider: 6.1.2

shared\_preferences: 2.2.2

shared\_preferences\_android: 2.2.1

shared\_preferences\_foundation: 2.3.4

shared\_preferences\_platform\_interface: 2.3.1

shared\_preferences\_web: 2.2.1

sqflite: 2.2.6

timeago: 3.6.1

url\_launcher: 6.2.5

url\_launcher\_android: 6.3.0

url\_launcher\_ios: 6.2.5

url\_launcher\_platform\_interface: 2.3.2

# The following adds the Cupertino Icons font to your application.

# Use with the CupertinoIcons class for iOS style icons.

cupertino\_icons: ^1.0.0

dependency\_overrides:

http: 1.2.1

uuid: ^4.0.0

dev\_dependencies:

flutter\_lints: 4.0.0

lints: 4.0.0

flutter\_test:

sdk: flutter

# For information on the generic Dart part of this file, see the

# following page: https://dart.dev/tools/pub/pubspec

# The following section is specific to Flutter.

flutter:

# The following line ensures that the Material Icons font is

# included with your application, so that you can use the icons in

# the material Icons class.

uses-material-design: true

# To add assets to your application, add an assets section, like this:

assets:

- assets/fonts/

- assets/images/

- assets/videos/

- assets/audios/

- assets/lottie\_animations/

- assets/rive\_animations/

- assets/pdfs/

# An image asset can refer to one or more resolution-specific "variants", see

# https://flutter.dev/assets-and-images/#resolution-aware.

# For details regarding adding assets from package dependencies, see

# https://flutter.dev/assets-and-images/#from-packages

# To add custom fonts to your application, add a fonts section here,

# in this "flutter" section. Each entry in this list should have a

# "family" key with the font family name, and a "fonts" key with a

# list giving the asset and other descriptors for the font. For

# example:

# fonts:

# - family: Schyler

# fonts:

# - asset: fonts/Schyler-Regular.ttf

# - asset: fonts/Schyler-Italic.ttf

# style: italic

# - family: Trajan Pro

# fonts:

# - asset: fonts/TrajanPro.ttf

# - asset: fonts/TrajanPro\_Bold.ttf

# weight: 700

#

# For details regarding fonts from package dependencies,

# see https://flutter.dev/custom-fonts/#from-packages

```

### Parti Principali del file `pubspec.yaml`

1. \*\*Nome e Descrizione del Progetto\*\*:

```yaml

name: evaluate\_performance\_app

description: A new Flutter project.

```

2. \*\*Configurazione di Pubblicazione\*\*:

```yaml

publish\_to: 'none' # Remove this line if you wish to publish to pub.dev

```

3. \*\*Versione dell'Applicazione\*\*:

```yaml

version: 1.0.0+1

```

4. \*\*Ambiente SDK\*\*:

```yaml

environment:

sdk: ">=3.0.0 <4.0.0"

```

5. \*\*Dipendenze\*\*:

```yaml

dependencies:

flutter:

sdk: flutter

flutter\_localizations:

sdk: flutter

auto\_size\_text: 3.0.0

cached\_network\_image: 3.3.1

collection: 1.18.0

data\_table\_2: 2.5.10

equatable: 2.0.5

flutter\_animate: 4.5.0

flutter\_cache\_manager: 3.3.1

font\_awesome\_flutter: 10.6.0

from\_css\_color: 2.0.0

go\_router: 12.1.3

google\_fonts: 6.1.0

http: 1.2.1

intl: 0.19.0

json\_path: 0.7.2

mime\_type: 1.0.0

page\_transition: 2.1.0

path\_provider: 2.1.3

path\_provider\_android: 2.2.5

path\_provider\_foundation: 2.4.0

path\_provider\_platform\_interface: 2.1.2

provider: 6.1.2

shared\_preferences: 2.2.2

shared\_preferences\_android: 2.2.1

shared\_preferences\_foundation: 2.3.4

shared\_preferences\_platform\_interface: 2.3.1

shared\_preferences\_web: 2.2.1

sqflite: 2.2.6

timeago: 3.6.1

url\_launcher: 6.2.5

url\_launcher\_android: 6.3.0

url\_launcher\_ios: 6.2.5

url\_launcher\_platform\_interface: 2.3.2

```

6. \*\*Override delle Dipendenze\*\*:

```yaml

dependency\_overrides:

http: 1.2.1

uuid: ^4.0.0

```

7. \*\*Dipendenze di Sviluppo\*\*:

```yaml

dev\_dependencies:

flutter\_lints: 4.0.0

lints: 4.0.0

flutter\_test:

sdk: flutter

```

8. \*\*Configurazioni Specifiche di Flutter\*\*:

```yaml

flutter:

uses-material-design: true

assets:

- assets/fonts/

- assets/images/

- assets/videos/

- assets/audios/

- assets/lottie\_animations/

- assets/rive\_animations/

- assets/pdfs/

```

### Utilità del file `pubspec.yaml`

- \*\*Gestione delle Dipendenze\*\*: Specifica quali pacchetti e versioni sono necessari per il progetto.

- \*\*Configurazione delle Risorse\*\*: Definisce le risorse come immagini, font e altri asset utilizzati dall'applicazione.

- \*\*Controllo delle Versioni\*\*: Gestisce le versioni del progetto e delle dipendenze.

- \*\*Configurazione di Pubblicazione\*\*: Impedisce la pubblicazione accidentale su pub.dev per pacchetti privati.

Se hai altre domande o necessiti di ulteriori chiarimenti, fammi sapere!

Il file `widget\_test.dart` contiene un test di base per un'applicazione Flutter. Ecco una panoramica del contenuto e del suo scopo:

### Contenuto del file `widget\_test.dart`

```dart

// This is a basic Flutter widget test.

//

// To perform an interaction with a widget in your test, use the WidgetTester

// utility that Flutter provides. For example, you can send tap and scroll

// gestures. You can also use WidgetTester to find child widgets in the widget

// tree, read text, and verify that the values of widget properties are correct.

import 'package:flutter\_test/flutter\_test.dart';

import 'package:evaluate\_performance\_app/main.dart';

void main() {

testWidgets('Counter increments smoke test', (WidgetTester tester) async {

// Build our app and trigger a frame.

await tester.pumpWidget(MyApp());

});

}

```

### Descrizione delle Parti Principali

1. \*\*Commenti Introduttivi (linee 1-6)\*\*:

- I commenti spiegano che questo è un test di base per un widget Flutter. Viene descritto come utilizzare `WidgetTester` per interagire con i widget durante il test.

2. \*\*Importazioni (linee 8-10)\*\*:

- `import 'package:flutter\_test/flutter\_test.dart';`: Importa il pacchetto `flutter\_test` che contiene strumenti per il testing.

- `import 'package:evaluate\_performance\_app/main.dart';`: Importa il file principale dell'applicazione.

3. \*\*Funzione Principale `main` (linee 12-17)\*\*:

- \*\*`testWidgets`\*\*: Definisce un test di widget chiamato "Counter increments smoke test".

- \*\*Descrizione del Test\*\*: "Counter increments smoke test".

- \*\*Parametro `tester`\*\*: Un'istanza di `WidgetTester` utilizzata per interagire con il widget durante il test.

- \*\*Costruzione dell'App e Trigger di un Frame\*\*:

```dart

await tester.pumpWidget(MyApp());

```

Costruisce l'applicazione `MyApp` e attiva un frame, permettendo al tester di interagire con l'app.

### Utilità del Test

- \*\*Test di Smoke\*\*: Questo test è un test di smoke di base per assicurarsi che l'applicazione possa essere costruita e visualizzata correttamente senza errori.

- \*\*Interazioni con i Widget\*\*: Utilizza `WidgetTester` per costruire l'applicazione e verificare che non ci siano crash o problemi di base.

### Espansione del Test

Per rendere il test più completo, puoi aggiungere ulteriori interazioni e verifiche, come ad esempio:

- Verificare che un testo specifico sia presente.

- Simulare tap su pulsanti e verificare i cambiamenti di stato.

- Interagire con altri widget e verificare le loro proprietà.

Ecco un esempio ampliato del test:

```dart

void main() {

testWidgets('Counter increments smoke test', (WidgetTester tester) async {

// Build our app and trigger a frame.

await tester.pumpWidget(MyApp());

// Verify that our counter starts at 0.

expect(find.text('0'), findsOneWidget);

expect(find.text('1'), findsNothing);

// Tap the '+' icon and trigger a frame.

await tester.tap(find.byIcon(Icons.add));

await tester.pump();

// Verify that our counter has incremented.

expect(find.text('0'), findsNothing);

expect(find.text('1'), findsOneWidget);

});

}

```

In questo esempio, il test verifica che il counter inizi a 0, simula un tap sul pulsante '+' e verifica che il counter sia incrementato a 1.

Se hai altre domande o necessiti di ulteriori chiarimenti, fammi sapere!

Il file `index.html` è il punto di ingresso principale per l'applicazione web. Ecco una panoramica del contenuto e della sua funzione:

### Contenuto del file `index.html`

```html

<!DOCTYPE html>

<html>

<head>

<!--

If you are serving your web app in a path other than the root, change the

href value below to reflect the base path you are serving from.

The path provided below has to start and end with a slash "/" in order for

it to work correctly.

Fore more details:

\* https://developer.mozilla.org/en-US/docs/Web/HTML/Element/base

-->

<base href="/">

<meta charset="UTF-8">

<meta content="IE=Edge" http-equiv="X-UA-Compatible">

<!-- iOS meta tags & icons -->

<meta name="apple-mobile-web-app-capable" content="yes">

<meta name="apple-mobile-web-app-status-bar-style" content="black">

<meta name="apple-mobile-web-app-title" content="evaluatePerformanceApp">

<meta name="robots" content="noindex" />

<!-- Favicon -->

<link rel="icon" type="image/png" href="favicon.png" />

<!-- Open Graph & SEO tags -->

<meta property="og:title" content="evaluatePerformanceApp" />

<meta property="og:description" content="Built with FlutterFlow." />

<meta property="og:image" content="https://storage.googleapis.com/flutterflow-prod-hosting/og\_splash\_gradient.png" />

<meta name="twitter:title" content="evaluatePerformanceApp" />

<meta name="twitter:description" content="Built with FlutterFlow." />

<meta name="twitter:image" content="https://storage.googleapis.com/flutterflow-prod-hosting/og\_splash\_gradient.png" />

<meta name="twitter:card" content="summary\_large\_image" />

<title> evaluatePerformanceApp </title>

<meta name="description" content="Built with FlutterFlow." />

<!-- Status Bar color in Safari browser (iOS) and PWA -->

<meta name="theme-color" media="(prefers-color-scheme: light)" content="#f1f4f8">

<meta name="theme-color" media="(prefers-color-scheme: dark)" content="#1d2428">

<link rel="manifest" href="manifest.json">

<script>

// The value below is injected by flutter build, do not touch.

var serviceWorkerVersion = null;

</script>

<!-- This script adds the flutter initialization JS code -->

<script src="flutter.js" defer></script>

</head>

<body>

<script>

window.addEventListener('load', function (ev) {

// Download main.dart.js

\_flutter.loader.loadEntrypoint({

entrypointUrl: 'main.dart.js',

serviceWorker: {

serviceWorkerUrl: 'flutter\_service\_worker.js?v=',

serviceWorkerVersion: serviceWorkerVersion,

},

onEntrypointLoaded: async function(engineInitializer) {

// Initialize the Flutter engine

let appRunner = await engineInitializer.initializeEngine({});

// Run the app

await appRunner.runApp();

}

});

});

</script>

</body>

</html>

```

### Descrizione delle Parti Principali

1. \*\*Base URL\*\*:

```html

<base href="/">

```

Imposta il URL base per tutte le risorse relative nel documento.

2. \*\*Meta Tag\*\*:

```html

<meta charset="UTF-8">

<meta content="IE=Edge" http-equiv="X-UA-Compatible">

<meta name="apple-mobile-web-app-capable" content="yes">

<meta name="apple-mobile-web-app-status-bar-style" content="black">

<meta name="apple-mobile-web-app-title" content="evaluatePerformanceApp">

<meta name="robots" content="noindex" />

<meta property="og:title" content="evaluatePerformanceApp" />

<meta property="og:description" content="Built with FlutterFlow." />

<meta property="og:image" content="https://storage.googleapis.com/flutterflow-prod-hosting/og\_splash\_gradient.png" />

<meta name="twitter:title" content="evaluatePerformanceApp" />

<meta name="twitter:description" content="Built with FlutterFlow." />

<meta name="twitter:image" content="https://storage.googleapis.com/flutterflow-prod-hosting/og\_splash\_gradient.png" />

<meta name="twitter:card" content="summary\_large\_image" />

<meta name="theme-color" media="(prefers-color-scheme: light)" content="#f1f4f8">

<meta name="theme-color" media="(prefers-color-scheme: dark)" content="#1d2428">

```

Questi meta tag configurano vari aspetti della pagina, inclusi il charset, compatibilità con IE, configurazioni per app web mobili su iOS, tag SEO e di Open Graph, e il colore della barra di stato.

3. \*\*Favicon\*\*:

```html

<link rel="icon" type="image/png" href="favicon.png" />

```

Imposta l'icona del sito web.

4. \*\*Manifest e Script di Flutter\*\*:

```html

<link rel="manifest" href="manifest.json">

<script>

var serviceWorkerVersion = null;

</script>

<script src="flutter.js" defer></script>

```

Include il file di manifest e lo script di inizializzazione di Flutter.

5. \*\*Script per Caricare l'Entrypoint di Flutter\*\*:

```html

<script>

window.addEventListener('load', function (ev) {

\_flutter.loader.loadEntrypoint({

entrypointUrl: 'main.dart.js',

serviceWorker: {

serviceWorkerUrl: 'flutter\_service\_worker.js?v=',

serviceWorkerVersion: serviceWorkerVersion,

},

onEntrypointLoaded: async function(engineInitializer) {

let appRunner = await engineInitializer.initializeEngine({});

await appRunner.runApp();

}

});

});

</script>

```

Questo script carica l'entrypoint di Flutter (`main.dart.js`) e avvia l'applicazione Flutter.

### Utilità del file `index.html`

Il file `index.html` è fondamentale per:

- \*\*Impostare le configurazioni iniziali\*\*: Meta tag, favicon, manifest, ecc.

- \*\*Inizializzare l'app Flutter\*\*: Carica lo script principale dell'app (`main.dart.js`) e avvia l'applicazione.

- \*\*SEO e Open Graph\*\*: Configura i tag per migliorare la visibilità e l'aspetto del sito sui social media.

Se hai altre domande o necessiti di ulteriori chiarimenti, fammi sapere!

# CONCLUSIONI

1. **Bibliografia**
2. [1] http://developer.android.com/
3. [2] http://code.google.com/p/i-jetty/
4. [3] http://maven.apache.org/
5. [4] htp://www.cs.waikato.ac.nz/ml/weka/