Firestore

In Firebase, ci sono principalmente due tipi di storage per memorizzare dati: Firebase Realtime Database e Firebase Cloud Firestore. Entrambi i servizi sono offerti da Firebase e sono utilizzati per scopi diversi a seconda delle esigenze dell'applicazione. Ecco una panoramica dei due tipi di storage:

1. Firebase Realtime Database:

- Firebase Realtime Database è un database NoSQL in tempo reale che memorizza i dati in un formato JSON.
- È progettato per offrire sincronizzazione in tempo reale dei dati tra client e server, il che significa che le modifiche ai dati vengono immediatamente propagate a tutti i client connessi.
- È ottimo per applicazioni che richiedono aggiornamenti in tempo reale dei dati, come chat in tempo reale, giochi multiplayer e app collaborative.
- La struttura dei dati è organizzata in un albero gerarchico di "nodi", dove ogni nodo può contenere altri nodi o dati.
- Offre un'API semplice e potente per l'accesso e la modifica dei dati, consentendo di leggere e scrivere dati in modo efficiente.

2. Firebase Cloud Firestore:

- Firebase Cloud Firestore è un database NoSQL documentale scalabile che offre una maggiore flessibilità e scalabilità rispetto al Realtime Database.
- Memorizza i dati in documenti JSON-like, ma offre funzionalità avanzate come query complesse, indicizzazione avanzata e transazioni.
- È progettato per offrire prestazioni elevate e scalabilità automatica, anche per applicazioni con grandi volumi di dati e requisiti di query complessi.
- Supporta anche la sincronizzazione in tempo reale dei dati tra client e server, consentendo di mantenere i dati aggiornati in tempo reale.

- Offre un'API più flessibile e potente rispetto al Realtime Database, consentendo di eseguire query complesse e operazioni di lettura/scrittura avanzate.

Scelta del tipo di storage:

- La scelta tra Realtime Database e Cloud Firestore dipende dalle esigenze specifiche dell'applicazione.
- Se l'applicazione richiede principalmente aggiornamenti in tempo reale dei dati e una struttura dati semplice, il Realtime Database potrebbe essere la scelta migliore.
- Se l'applicazione ha requisiti più complessi, come query complesse, transazioni e scalabilità avanzata, Cloud Firestore potrebbe essere più adatto.
- È possibile anche utilizzare entrambi i servizi in un'applicazione, a seconda delle esigenze specifiche dei diversi componenti dell'applicazione.

Firestore è un database NoSQL completamente gestito fornito da Firebase, una piattaforma di sviluppo di applicazioni mobile e web sviluppata da Google. Firestore è progettato per offrire prestazioni elevate, scalabilità e facilità d'uso, consentendo agli sviluppatori di memorizzare e sincronizzare facilmente i dati delle loro applicazioni in tempo reale. Ecco una panoramica dei principali concetti e funzionalità di Firestore:

Struttura dei dati:

- Firestore memorizza i dati in collezioni e documenti. Ogni documento contiene un insieme di campi chiave-valore, simile ai documenti JSON.
- Le collezioni contengono documenti e possono essere utilizzate per organizzare i dati in modo logico.
- I documenti possono avere sotto-collezioni nidificate all'interno di essi, consentendo di modellare dati complessi e relazioni tra i dati.

Query:

- Firestore supporta query potenti e flessibili che consentono agli sviluppatori di recuperare dati in base a criteri specifici.
- Le query possono essere filtrate, ordinate e paginate per ottenere esattamente i dati richiesti.
- Firestore offre supporto per query in tempo reale, il che significa che le query restituiranno automaticamente i risultati aggiornati quando i dati cambiano nel database.

Sincronizzazione in tempo reale:

- Firestore fornisce sincronizzazione in tempo reale dei dati tra client e server. Ciò significa che ogni modifica ai dati viene immediatamente propagata a tutti i client connessi.
- Gli aggiornamenti dei dati vengono inviati tramite WebSockets o altre tecnologie di comunicazione in tempo reale per garantire una reattività e una coerenza elevate delle applicazioni.

Scalabilità e affidabilità:

- Firestore è completamente gestito da Google Cloud Platform, il che significa che offre scalabilità automatica e alta affidabilità senza la necessità di gestire infrastrutture di database.
- I dati vengono replicati automaticamente su più server e data center per garantire la disponibilità e la ridondanza dei dati.

Sicurezza:

- Firestore offre un modello di sicurezza basato su regole che consente agli sviluppatori di definire regole di accesso granulari per controllare chi può leggere, scrivere e modificare i dati nel database.

- Le regole di sicurezza sono definite utilizzando un linguaggio di espressione che consente di specificare condizioni complesse basate su identità utente, ruoli e altre informazioni.

Integrazione con Firebase:

- Firestore è completamente integrato con altri servizi Firebase, come l'autenticazione, le funzioni cloud e il cloud storage.
- Questa integrazione semplifica lo sviluppo di applicazioni end-to-end utilizzando Firebase come piattaforma unificata per sviluppare, testare e distribuire applicazioni web e mobile.

Tipi di dato di un documento in Firestore:

Firestore supporta vari tipi di dati per i campi di un documento, tra cui stringhe, numeri, booleani, date, array e mappe.

```
// Esempio di documento Firestore con vari tipi di dati
const documentData = {
  stringField: "Hello Firestore",
  numberField: 42,
  booleanField: true,
  dateField: new Date(),
  arrayField: ["apple", "banana", "cherry"],
  mapField: { key1: "value1", key2: "value2" }
};
```

Accedere a Firestore:

Puoi accedere a Firestore utilizzando l'oggetto `firebase.firestore()`.

```
import firebase from 'firebase/app';
import 'firebase/firestore';
```

```
const db = firebase.firestore();
```

Set Document:

Per impostare un nuovo documento in Firestore, puoi utilizzare il metodo `set()`.

```
const docRef = db.collection('users').doc('user1');
docRef.set({
  name: 'John Doe',
  age: 30
});
```

Add Document:

Per aggiungere un nuovo documento con un ID generato automaticamente, puoi utilizzare il metodo `add()`.

```
db.collection('users').add({
  name: 'Jane Smith',
  age: 25
});
```

Update:

Per aggiornare un documento esistente in Firestore, puoi utilizzare il metodo `update()`.

```
const docRef = db.collection('users').doc('user1');
docRef.update({
  age: 35
});
```

Delete:

Per eliminare un documento esistente in Firestore, puoi utilizzare il metodo `delete()`.

```
const docRef = db.collection('users').doc('user1');
docRef.delete();
```

Ottenere i dati:

Per ottenere i dati di un documento, puoi utilizzare il metodo `get()`.

```
const docRef = db.collection('users').doc('user1');
docRef.get().then((doc) => {
  if (doc.exists) {
    console.log("Document data:", doc.data());
  } else {
    console.log("No such document!");
  }
});
```

Realtime updates:

Firestore offre supporto per gli aggiornamenti in tempo reale dei dati. Puoi registrare un listener per i cambiamenti utilizzando il metodo `onSnapshot()`.

```
const docRef = db.collection('users').doc('user1');
docRef.onSnapshot((doc) => {
  console.log("Current data:", doc.data());
});
```

Query sui documenti:

Firestore supporta query flessibili per recuperare documenti in base a criteri specifici.

```
// Esempio di query su Firestore
const query = db.collection('users').where('age', '>', 25).or
```

```
derBy('age').limit(5);
query.get().then((querySnapshot) => {
  querySnapshot.forEach((doc) => {
    console.log(doc.id, " => ", doc.data());
  });
});
```

And/Or:

Puoi utilizzare i metodi `where()` per combinare più criteri di query con operazioni logiche come AND e OR.

```
const query = db.collection('users').where('age', '>', 25).wh
ere('city', '==', 'New York');
```

orderBy, limit, count:

Puoi ordinare i risultati della query, limitare il numero di risultati e contare il numero totale di risultati utilizzando i metodi `orderBy()`, `limit()` e `size()`.

```
const query = db.collection('users').orderBy('age').limit(1
0);
query.get().then((querySnapshot) => {
  console.log("Total documents:", querySnapshot.size);
});
```

Pagination:

Puoi implementare la paginazione per recuperare grandi insiemi di dati utilizzando cursors.

```
let lastVisible = null;

// Prima pagina
const firstPageQuery = db.collection('users').orderBy('nam
e').limit(10);
```

```
firstPageQuery.get().then((documentSnapshots) => {
  lastVisible = documentSnapshots.docs[documentSnapshots.doc
s.length - 1];
  // Mostra i documenti della prima pagina
});

// Pagina successiva
const nextPageQuery = db.collection('users').orderBy('name').
startAfter(lastVisible).limit(10);
nextPageQuery.get().then((documentSnapshots) => {
  lastVisible = documentSnapshots.docs[documentSnapshots.doc
s.length - 1];
  // Mostra i documenti della pagina successiva
});
```

Offline data:

Firestore offre il supporto per l'accesso ai dati offline e la sincronizzazione automatica dei dati una volta che il dispositivo è di nuovo online.

```
firebase.firestore().enablePersistence()
  .catch((err) => {
    if (err.code === 'failed-precondition') {
      console.log('Persistence failed because multiple tabs a
re open.');
    } else if (err.code === 'unimplemented') {
      console.log('Persistence is not available.');
    }
  });
```

GetData from cache:

Puoi specificare esplicitamente di ottenere i dati dalla cache utilizzando l'opzione `cacheFirst`.

```
db.collection("cities").where("state", "==", "CA")
  .get({ source: "cache" })
  .then((querySnapshot) => {
    querySnapshot.forEach((doc) => {
      console.log(doc.id, " => ", doc.data());
    });
  });
```

Controllare se i dati vengono dalla cache:

Puoi controllare se i dati vengono ottenuti dalla cache utilizzando la proprietà `fromCache` nell'oggetto `QuerySnapshot`.

```
docRef.get().then((doc) => {
  console.log("Document data:", doc.data());
  console.log("Data came from cache:", doc.metadata.fromCach
e);
});
```