Anton Kornholt

Google Firestore

En artikel om en cloudbaseret realtime database

Introduktion

I denne artikel vil jeg fortælle om googles cloud-baseret realtime database Firestore.

Jeg vil herunder komme ind på fordele og ulemper ved at bruge af denne database til ens app.

Derudover vil jeg komme med eksempler på, hvordan data kan gemmes og hentes med denne databaseløsning.

Hvad er Firestore?

Firestore er en flexibel, skalerbar, cloud database, der hostes på googles servere. Når data ændres på serveren, så opdateres den i realtime på de forskellige klienter, der er tilknyttet denne database. Den tillader derfor brugere at tilgå hinandens data på en meget flydende måde, der tillader hurtig og nem udveksling af data.

Derudover så kan databasen også fungerer offline, så den virker ligegyldig, hvor dårlig internetforbindelse brugeren har, og hvis de slet ikke har nogen.

Sidst men ikke mindst, så er Firestore også lavet til at arbejde sammen med googles andre Cloud-baserede services, så integration af disse bliver nemmere ved brug af Firestore.

Det er dog også vigtigt at sige, at Firestore er i beta for nu. Det betyder, at der er nogle restriktioner og potentielle ulemper ved at bruge denne database for nu, som jeg vil komme ind på senere.

Man kan godt sige, at med en Firestore database så bliver ens programmering af denne ret high level. Hvilket vil sige, at man abstrahere en del væk fra standard java kode og prøver at gøre udviklingen mere brugervenlig og let forståelig. Dette gøres ved hjælp af en række google dependencies, der tillader en at bruge en række metoder og klasser, der ikke er i standard java-biblioteket. Så for at bruge googles Firestore, så skal man sætte sig ind i et specifikt google API, der tillader en, at arbejde med Firestore.

Firestore kan bruges sammen med en række forskellige udviklingssprog. Disse inkluderer:

Web

IOS

Android

Java

Python

Node.JS

GO

PHP

C#

Hvordan bruger man Firestore?

For at begynde at bruge Firestore skal man igennem en række forskellige steps. Der er her 4 ting, der skal på plads, før man er sat op:

1. Integrer Cloud Firestore SDK i dit projekt.
2. Gør din data sikker ved hjælp af Firestore security rules.
3. Tilføj data til din Firestore database.
4. Hent data fra databasen, enten med realtime listeners eller queries.

At kode med Firestore

Som et eksempel på, hvordan man simpelt opretter og laver en database, er jeg gået gennem googles guide, der kan findes her:

https://firebase.google.com/docs/firestore/quickstart

Her går man de førnævnte steps igennem på en let og pædagogisk måde. Der tillader, at man koder i ens mest familiære sprog.

Step 1 - Integration af firestore

Integreringen af firestore sker ved at sætte ens authentication op til at starte med. Her ses hvordan dette gøres i Java ved brug af google cloud platformen:

```
import com.google.auth.oauth2.GoogleCredentials;
import com.google.cloud.firestore.Firestore;

import com.google.firebase.FirebaseApp;
import com.google.firebase.FirebaseOptions;

// Use the application default credentials
GoogleCredentials credentials = GoogleCredentials.getApplicationDefault();
FirebaseOptions options = new FirebaseOptions.Builder()
    .setCredentials(credentials)
    .setProjectId(projectId)
    .build();
FirebaseApp.initializeApp(options);

Firestore db = FirestoreClient.getFirestore();
```

Step 2 - Tilføj sikkerhedsregler

For at sikre sig, at det kun er de rigtige personer kan man begrænse brugeres adgang til ens firestore database. Dette kan man gøre ved hjælp af Firestore security rules. Det er nemlig vigtigt, at ens data er ordentligt beskyttet og ikke kan læses og ses af alle, når man går i produktion med ens produkt. Herunder ses, hvordan security rules kan sættes op til at kræve authentication, når en bruger vil skrive til databasen:

```
// Allow read/write access on all documents to any user signed in to the application
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth.uid != null;
    }
  }
}
```

Step 3 - tilføj data

For at tilføje data til ens database skal man have en collectionreference i ens database man kan pege på. Herefter kan man pushe objekter til denne reference. Hvis man peger på en ikke eksisterende collectionreference, så oprettes denne for en. Herunder ses et eksempel på, hvordan dette kan foregå i java:

```
DocumentReference docRef = db.collection("users").document("alovelace");
// Add document data  with id "alovelace" using a hashmap
Map<String, Object> data = new HashMap<>();
data.put("first", "Ada");
data.put("last", "Lovelace");
data.put("born", 1815);
//asynchronously write data
ApiFuture<WriteResult> result = docRef.set(data);
// ...
// result.get() blocks on response
System.out.println("Update time : " + result.get().getUpdateTime());
```

Det er ikke nødvendigt at alle objekter indenfor den samme collectionreference har de samme key value pairs i deres maps. Det vil altså sige, at man for eksempel ville kunne tilføje et nyt objekt med et ekstra key value pairs som det kan ses her:

```
DocumentReference docRef = db.collection("users").document("aturing");
// Add document data with an additional field ("middle")
Map<String, Object> data = new HashMap<>();
data.put("first", "Alan");
data.put("middle", "Mathison");
data.put("last", "Turing");
data.put("born", 1912);

ApiFuture<WriteResult> result = docRef.set(data);
System.out.println("Update time : " + result.get().getUpdateTime());
```

Step 4 - læs data

For at læse data fra firestore kan man oprette en query mod databasen, eller man kan tjekke på ens firestore konsol om den er blevet oprettet korrekt. Et eksempel på en query, der henter den data vi lige har tilføjet kan ses herunder:

```
// asynchronously retrieve all users
ApiFuture<QuerySnapshot> query = db.collection("users").get();
// ...
// query.get() blocks on response
QuerySnapshot querySnapshot = query.get();
List<QueryDocumentSnapshot> documents = querySnapshot.getDocuments();
for (QueryDocumentSnapshot document : documents) {
  System.out.println("User: " + document.getId());
  System.out.println("First: " + document.getString("first"));
  if (document.contains("middle")) {
    System.out.println("Middle: " + document.getString("middle"));
  }
  System.out.println("Last: " + document.getString("last"));
  System.out.println("Born: " + document.getLong("born"));
}
```

Alternative databaseløsninger

Hvis man ikke er interesseret i at bruge et beta-produkt, så er et godt alternativ til firestore googles egen firebase database. Denne er også skybaseret og kan opdateres i realtime hos alle brugere af den.

Hvis man vil have en mere lokal database kan man enten benytte sig af Rooms eller SQLLite.

SQLLite er kodning på meget lavt plan, hvilket vil sige, at man ikke overlader meget kontrol til andre. Det er meget tæt på at være helt almindelige SQL man skriver her.

Rooms abstrahere lidt mere af logikken væk. Så hvis man vil have noget, der kan begrænse ens behov for at skrive SQL, så kan Rooms være en god løsning.

Konklusion

Trods en temmelig god guide fra google har jeg haft kæmpe problemer med firestore, da jeg prøvede at bruge det i mit eget projekt. Fik utallige dependency-problemer og versions-konflikter, der faktisk gjorde, at jeg endte med at skrotte det hele og gå tilbage til firebase.

Da firestore er i beta viste det sig lidt for svært at finde løsninger til de problemer jeg stødte ind i. Hver gang jeg fik løst et problem, syntes et andet problem at opstå. Det var så frustrerende, at jeg måtte gå en del ugers arbejde tilbage.

Det har dog været en lærerig process at prøve kræfter med et beta produkt, men en anden gang vil jeg være mere indstillet på de problemer der kan opstå ved at bruge sådan et produkt.

Ressourcer

Firestore hurtig start guide:

https://firebase.google.com/docs/firestore/quickstart

Firestore sikkerhedsregler:

https://firebase.google.com/docs/firestore/security/get-started

Firebase konsol:

https://console.firebase.google.com/u/0/