リモートで勉強会#5 – テーマ何でもLT会–

Firebaseで**Webアプリ**開発した話

～ Firestoreのデータ設計について考える ～

#remosta
@_takeshi_24

自己紹介

無職

西 武史(にし たけし)

@_takeshi_24

- 福岡のSIerでシステムエンジニア
- フリーランスプログラマー
- 東京のスタートアップCo-Founder取締役CTO
- 福岡のスタートアップで新規事業立ち上げ
- 株式会社diffeasy取締役CTO
- スタートアップ創業準備中 ←**今ここ**

#Nuxt.js #Firebase #Vue.js #Ruby #GCP #ブロックチェーン
#スタートアップ #起業
#焼き肉 #温泉 #キャンプ #旅行

最近やっている事

達成.me

目標と期限を宣言してシェア！目標への熱量を
可視化し、応援して繋がるサービス。

https://tassei.me

Nuxt.js / Firebase / CloudFunctions / Node.js

HashTalk(仮)

価値観や目標への共感で繋がるマッチングプラットフォームSNS

Nuxt.js / Firebase / CloudFunctions / Node.js

HashTalk

受託開発
IT支援

新規事業開発における技術支援、DX支援。
ソフトウェア受託開発。

Firestoreのデータ設計について考える

Firebase/Firestoreとは？

Firebaseとは？

Googleが提供するBaaS(Backend as a Service)

バックエンドの処理（認証、ストレージの管理、データベースの管理など）が不要。

Firestoreとは？

FirebaseのNoSQLデータベース。

Firestoreのデータ設計について考える

1. Firestoreのリレーションデータをどう取り扱うか？
   - SubCollection
   - キー参照モデル
   - Reference型
2. セキュリティを意識したデータ設計
3. 権限管理はカスタムクレームを利用
4. N+1問題どうする？

SubCollection

SubCollectionとは？

users
- [UserID]
  - name: “にし”
  - age: 38
  - diaries
    - [DiaryID1]
      - title: “タイトル”
      - content: “本文。本文。”
    - [DiaryID2]
      - title: “タイトル2”
      - content: “本文2。本文2。”

- データの関係がわかりやすい。
- 階層が深いとわかりにくくなる。

SubCollection - CollectionGroup

```
const snapshot = await app.firestore()
    .collectionGroup("diaries")
    .where()
    .get()
```

- collectionGroupでSubCollectionを横断的に検索可能。

Key参照モデル

Key参照モデルとは？

users
- [UserID]
  - name: “にし”
  - age: 38

diaries
- [DiaryID1]
  - userId: 1
  - title: “タイトル”
  - content: “本文。本文。”
- [DiaryID2]
  - userId: 1
  - title: “タイトル2”
  - content: “本文2。本文2。”

SubCollectionとKey参照モデルどっち使う？

SubCollectionとKey参照モデルどちらを使うか？

判断基準は？？

私の判断基準

1. 単純な親子関係だったら、SubCollection

2. 検索のパターンが複数あれば、Key参照モデル

3. 階層が深くなるものは、Key参照モデル

Reference型(参照型)

Reference型とは？

diaries
- [DiaryID1]
  - title: “タイトル”
  - content: “本文。本文。”
  - user: /users/[userID1]
- [DiaryID2]
  - title: “タイトル2”
  - content: “本文2。本文2。”
  - user: /users/[userID2]

- N:1の親子関係のデータではReference型を利用。

Reference型の取得

```
const diary = await app.firestore()
    .collection("diaries")
    .doc([diaryID])
    .get()
const user = diary.data().user.get()
```

- 子のデータの取得が容易。

セキュリティを意識したデータ設計

Firestoreではセキュリティルールの設計が大事

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read;
      allow create: if request.auth.uid != null;
      allow update, delete: if request.auth.uid == userId;
    }
  }
}
```

基本的にフロントエンドから直接操作できるので、フロントエンド側でUserIDを偽装可能。

セキュリティを意識したデータ設計

例) ユーザーの氏名、住所、連絡先などの個人情報と、ニックネーム、プロフィール写真などの公開情報を持つデータ

users

L [UserID]

L nickname: “にし”

L profile: “xxxxxxx.jpg”

L name: “西武史”

L address: “福岡市東区・・・”

L email: “takeshi@hogehoge.com”

フィールド単位でセキュリティルール設定できないので、公開されてしまう！！

セキュリティを意識したデータ設計

users

- [UserID]
  - nickname: “にし”
  - profile: “xxxxxxx.jpg”

privates

- [UserID]
  - name: “西武史”
  - address: “福岡市東区・・・”
  - email: “takeshi@hogehoge.com”

collectionを分ける。

権限管理はカスタムクレームを利用

権限管理はカスタムクレームを利用

例) adminユーザーは管理者画面を利用できて、一般ユーザーは管理者画面を利用できない。

カスタムクレームに「admin」を設定

```
await admin.auth().setCustomUserClaims(
    uid, { admin: true }
)
```

adminの場合のみ処理を行う

```
if(user.customClaims.admin) {
}
```

- カスタムクレームは権限管理のみ利用。
- ユーザーの情報はcollectionを利用。

N+1問題どうする？

N+1問題どうする？

```
db.collection(“diaries”).get()
  .then(function(snapshot) {
     snapshot(function(diary) {
        diary.data().user.get()
     })
  })
```

N+1問題。ループごとにリクエスト。

N+1はある程度仕方ない・・

例えば、役職名など数が**限定的**、**ほとんど変わらない**データは、ループの前に取得して変数に保持しておいて、ループ内で、変数から取得するなどの工夫。

Firebaseどうなの？

Firestoreどうなの？

- 基本的には**最高**です！
- 開発速度、サーバーの維持管理を考えると**スモールスタート**に最適！
- フワッとした要件からだんだん機能を増やしていくと、**ツラミ・・・**。
- 最初から**一度捨てて作り直す**ことも考慮しておく。
- 要件がある程度固まっていれば、良い。

宣伝です！

HashTalk(仮)

HashTalkとは？

HashTalkとは？

価値観や**目標**への**共感**で繋がる

マッチングプラットフォームSNS

HashTalkが提供する価値

自分の目標や悩みを**本音**で**安心**して投稿することで、

今出会いたい潜在的な**未来の仲間**と繋がる

HashTalkとは？

友達申請/フォローがなく、まずはやりたいこと/誰にも言えない悩みなどを**匿名**でただ投稿するだけ。
投稿に「**共感**」したりコメントのやり取りすることで「**共感度**」が上がっていき、実名プロフィールが見れるようになり、**最終的に友達として繋がる**。

匿名の公開プロフィールと
実名のプロフィールを登録

匿名でやり取り。誰が言ってるか？
ではなく言葉に共感。

やり取りすることで共感度が
上がり、最終的に友達になる。

利用技術

- Firebase Authentication
- Firestore
- Cloud Storage
- Cloud Functions
- BigQuery

HashTalk(仮)

事前登録受付中