```
{
    タイトル：２０２１　二学期情報レポート
    クラス：D
    グループ番号：3
    グループ名：不明
    メンバー:[
        {
            氏名：網代健人,
            番号：2
        },
        {
            氏名：吉野慧那,
            番号：45
        }
    ]
}
```

完成品

[俺らの考える最強のタスク管理アプリ（oretasu.page.link/app）](oretasu.page.link/app)

企画書の内容

企画書 ― 4D02網代健人　4D45吉野慧那

動機

まず，私はとてつもなく自分のやるべきこと管理が苦手です。
そこで，自分の理想的なタスク管理アプリを作ろうとしていたのですが，まとまった時間がなく，作れていませんでした。
さらに，自分自身，理想のタスク管理アプリを作ると言っておきながら理想が具体的にわかっていないので，2人で作れば真・完璧タスク管理アプリができると考えました。
今回は，ChromeBookでできるのものみという制約があったため，自分のスキルアップのためにも，いつも慣れているAndroidプラットフォームではなく，WebAppとして記述することにしました。

別の案

Twitter2のようなものを作ろうとは思いましたが，独創性を出す部分が全く思いつかなかったので，没になりました。

概要

実現

WebAppとして作成するには，まず，ドメインを取得しなければなりませんが，運良くGoogleさんがFirebaseで以下のドメインを貸してくれたので，使うことにします。
https://ajiken4610-task-mng.web.app/

サーバーは自分の家に置くのも厳しいので，FirebaseHostingを使用します。
費用は一日３６０MBまでのアクセスなら無料とのことでしたので，サイトの容量を少なくする方針で行こうと思います。

タスク管理のデーターベースですが，FirestoreDatabaseを使おうと思います。
こんな小規模のプロジェクトにFirestoreを使用すると怒られそうですが，いかんせん使い勝手がいいのでそうさせてもらうことにしました。

Firestoreのデータ構造は以下のようにします。

/users/{document}/tasks/{task}

users直下にはユーザーのデータを，taskはuserからリンクが外れることはないので，その下に置くことにしました。CollectionGroupを使用すれば，全taskに対してクエリすることもできるので，問題ないと思われます。

タスクに添付したいこともあると思ったので，その時はGoogleCloudStorageからBucketで取得します。

これは，私がまだ18歳になっていないので実現できませんが，ぜひAdMobで広告を付けたい。

実装予定

- index.html :aboutus等
- タスク追加
- タスク削除
- タスク編集

- タスククエリ ＋　カスタムなメタタグでのクエリ

タスクに紐付けるデータ

- 進捗
    - 新規
    - 進行中
    - 完了
    - 保留
    - 取消
- 期限
    - 日にち指定
    - 時間指定
- タイトル
- 説明文
- 他メタ
- カスタムメタタグ

必要機能

・タスクの作成。
→タイトル、期限日、状態を持つ。
・状態は、「未着手」「着手中」「完了」の3種類。
・タスクの一覧表示。
・タイトル、期限日、状態の編集。
・フォルダに分類して管理。
→フォルダを作成することができる必要あり。
また、ユーザーはフォルダの一覧を表示できる必要がある。
・アカウントを持ち、ログインしたユーザーは自分のフォルダないしはタスクだけを閲覧・編集することができる。
・パスワードを忘れた場合の再登録。

カスタムメタタグについて

タスク管理するにあたって、ポケモンに付けることができる印(□○▲★など)を拡張性のあるようにタスクに紐づけて、それをクエリの条件に加えることができるようにすることにしました。(例が不適切で申し訳ありません)

進行状況　（9月8日記）

[俺らの考える最強のタスク管理アプリ（ajiken4610-task-mng.web.app)](ajiken4610-task-mng.web.app)

作成するまでの過程

サーバレスアプリケーションとして実装するために、Firebaseを使用しました。

全文一致検索

FitebaseFirestoreは、文字列の全文一致検索機能が提供されていないのです。
ですので、検索アルゴリズムを自分で設計する必要がありました。
そこで、今回はBiGramを採用することにしました。
しかし、BiGramの配列をbi-gramフィールドにsetするにしても、
Firestoreはクエリ内で１度しかarray-contains句を使えない制約があります。
なので、配列でBiGramを保存するのではなく、Mapで保存することにしました。
具体的には、
配列；

```
bi-gram:["おは","はよ","よう","うご","ござ","ざい","いま","ます",]
```

とするところを、

```
bi-gram:{
  "おは":true,
  "はよ":true,
  "よう":true,
  "うご":true,
  "ござ":true,
  "ざい":true,
  "いま":true,
  "ます":true,
}
```

とします。

こうすることにより、where==句を複数連ねることで全文一致検索を行うことができます。

ユーザー認証

ユーザー認証ではFirebaseAuthとAuthUIを使用したため、コーディングはほとんどしていません。
firestoreのセキュリティルールは以下のように設定しました

```
rules_version = '2';
service cloud.firestore {
```

```
      function signed(){
           return request.auth != null;
  }
  match /databases/{database}/documents {
    match /user/{userDocument} {
      allow create: if signed()
        && request.auth.uid == request.resource.data.uid;
      allow read,update,delete: if signed()
           && request.auth.uid == resource.data.uid;

      match /{subCollections=**}{
           allow read,write:if signed() &&

get(/databases/$(database)/documents/user/$(userDocument)).data.ui
d == request.auth.uid;
      }
    }
  }
}
```

動的に変更されるUIシステム

１つの画面を表す抽象クラスをFragmentとして定義し、
画面は全てFragmentを実装するものとしてシステムを動かしています。

```
class Fragment {

    // public ::
    onCreate() {}
    onViewCreated(view) {}
    onResume() {}
    onChangeState(oldState, newState) {}
    onPause() {}
    onDestroy() {}
    onSaveState(state) {}
}
```

Fragmentは、上の7つの抽象メソッドを含みます。

Fragment作成時には、

onCreate
onViewCreated
onChangeState
onResume

の順番で、
Fragmentの状態変化時には

onChangeStateが、

Fragment破棄時には

onPause
onDestroy

の順で、
Fragmentが隠れたときは、
onPauseが、
Fragmentが隠された後、また最前面に来たときは
onResumeが、

それぞれ呼ばれます。

これは、Androidのアクティビティのライフサイクルを参考にしました。

Fragmentのインスタンス生成には、Factoryデザインパターンを使用しました。
抽象クラスFragmentFactoryを実装したクラスを、FragmentManagerクラスにコールバックして、動かします。

FragmentがURLが切り替わるのに応じて変わる様子
Fragmentの背景色を透過させています。
https://photos.app.goo.gl/7mzShvqCVJ8mViKZ9

このような動きを実現するために、スレッドの排他制御をする必要があったため、以下のようなロジックを組みました。

```
class Synchronizer {
    _tasks = [];

    async synchronized(func) {
```

```
        if (this._tasks.length > 0) {
            let promise = (async function(synchronizer) {
                await synchronizer._tasks[synchronizer._tasks.length
- 1];
                await func();
                synchronizer._tasks = synchronizer._tasks.filter(task
=> task !== promise);
            })(this);
            this._tasks.push(promise);
            await promise;
        } else {
            let promise = func();
            this._tasks.push(promise);
            await promise;
            this._tasks = this._tasks.filter(task => task !==
promise);
        }
    }
}
```

synchronizedが短時間に2回呼ばれると排他制御が壊れる可能性がありますが、URLの変更はそこまで頻繁に起こるわけではないので、妥協しました。

作成物について

サイトの構成

このサイトには、以下の24のページが含まれます。

1. 404
2. about
3. account
4. send-password-email
5. logout
6. login
7. mc
8. mt
9. mv
10. ml
11. md

12. m
13. mn
14. tc
15. tt
16. te
17. t
18. ta
19. tm
20. tr
21. te
22. td
23. app
24. sc

１．404

ページが見つからなかったときに表示されますが、サーバー側では全てのパスが/index.htmlにオーバーライドされるため、まず表示されませんが、Fragmentで見つからないパスがAjaxされたときに表示されます。

２．about

「ようこそ！」のページです。特にロジックはありませんが、ログイン時と非ログイン時でボタンが変わります。これは、CSSによるものです。

３．account

アカウントの詳細ページです。大した内容はありませんが、ログアウトとパスワード変更はここから行えます。

４．send-password-email

パスワード変更メールの送信先を編集します。デフォルトでは自分自身のメールアドレスが入っています。

５．logout

ログアウト確認のボタンのみが存在します。

６．login

AuthUIを使用して、ログインするためのUIが存在します。そのほかは特にありません。

７．mc

MetaCreateの略です。あたらしいカスタムメタを名前を指定して作成するUIを提供します。

８．mt

MetaTypeの略です。カスタムメタのタイプを以下の2つから選べます。
・EXIST　　メタがタスクについていないか、いるかの2通りの状態が存在します。
・ENUM　　メタがタスクについていないか、Nつの定義値のN＋１通りの状態が存在します。

９．mv

MetaValueの略です。ENUMのメタの定義値を編集します。

１０．ml

MetaListの略です。カスタムメタの一覧です。申し訳程度に検索機能がついています。

１１．md

MetaDeleteの略です。カスタムメタを削除します。

１２．m

Metaの略です。カスタムメタの詳細を表示します。

１３．mn

MetaNameの略です。カスタムメタの名前を編集します。

１４．tc

TaskCreateの略です。タスクをタイトルを指定して作成します。

１５．tt

TaskTitleの略です。タスクのタイトルを編集します。

１６．te

TaskdEscriptionの略です。タスクの説明文を編集します。

１７．t

Taskの略です。タスクの詳細を表示します。

１８．ta

TaskdAteの略です。タスクの日時を編集します。

１９．tm

TaskMetadeleteの略です。タスクとメタの紐づけを解除します。

２０．tr

TaskmetacReateの略です。タスクとメタを紐づけます。

２１．tl

TaskListの略です。appのほうが性能がいいクエリが可能です。

２２．td

TaskDeleteの略です。タスクを削除すると、サブタスクも連鎖的に削除されます。

２３．app

タスクの検索および並べ替えを行うことができます。

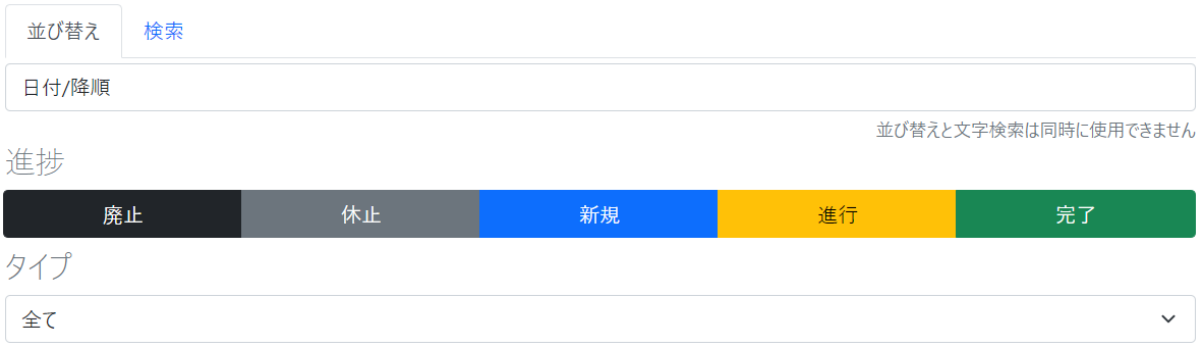
２４．sc

SubtaskCreateの略です。タスクのサブタスクを作成します。

タスクの検索機能について

普通のタスク管理アプリ同様、日付順に並べたりすることができます。
これには、Firestoreの複合インデックスを使用しています。

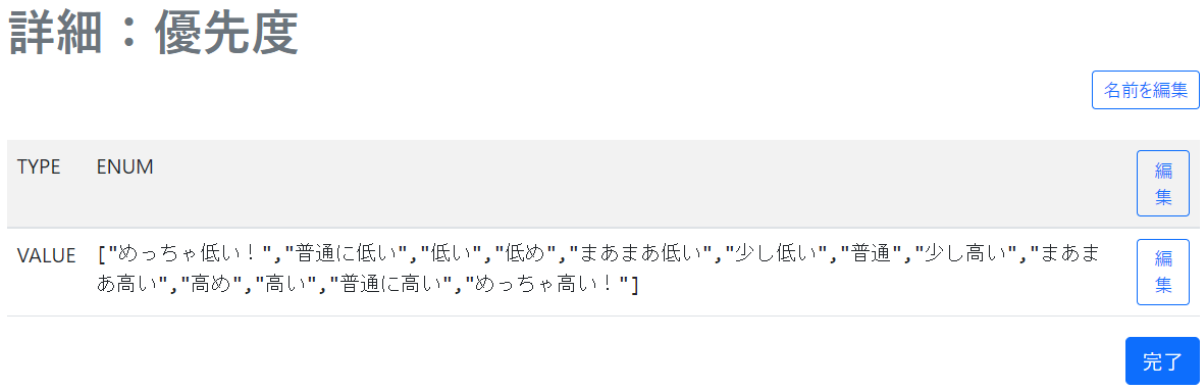
この8つのインデックスが有効となっています。
タスクの検索は、以下のようになっています。



一番上の「並び替え・検索」では、並び替えを利用するか、文字列を利用した検索にするかを切り替えることができます。Firestoreの制約で、並び替えと全文一致検索は同時に利用できません。

その下の「進捗」では、タスクに設定された進捗情報で絞り込みを行います。
その下のタイプは、「タスク・スケジュール・全て」の3つから絞り込むことができます。

検索のタブを選択すると、このようなUIになります。
この状態では、全文一致検索と、カスタムメタによる絞り込みが行えます。



このようなカスタムメタが定義されているとき、

また、このようなタスクがあるとき、

このように検索することで、目的のタスクを見つけることができます。

サブタスクについて

タスクに内訳を作れたらいいなって思うことありますよね、なので、サブタスクを実装しています。

詳細：根元のタスク

名前を編集

説明文はありません

説明文を編集

状態

廃止 | 休止 | 新規 | 進行 | 完了

サブタスク

進行

サブタスク11

説明文はありません

今日中

詳細

完了

サブタスク12

説明文はありません

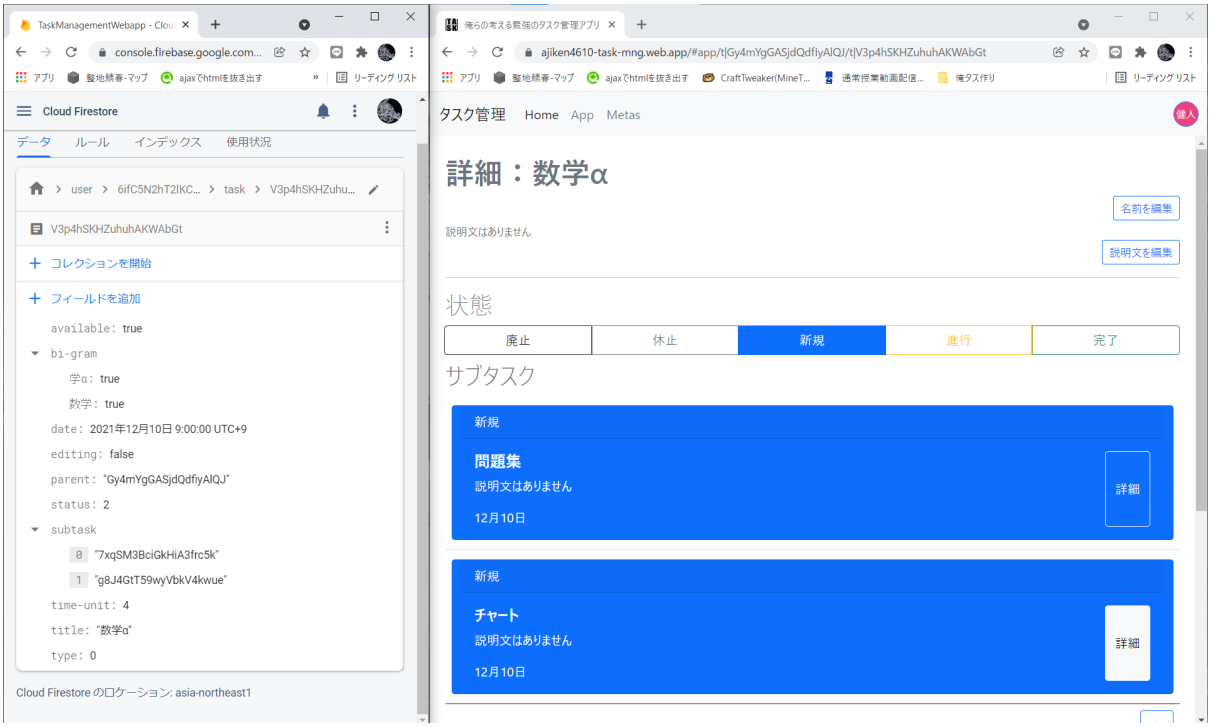
今日中

詳細

サブタスクのサブタスクも作成可能です。

サブタスクは、親のタスクが削除されると連鎖的に削除されます。

この場合では、「根元のタスク」の削除ボタンを押すと、
「根元のタスク」「サブタスク11」「サブタスク12」「サブタスク21」「サブタスク22」の5つのタスクが削除されます。



タスクのデータ構造はこうなっています。

URLは、以下のようになっています。
ただし、このURLはそのアカウントでログインしない限り意味を成しません。
https://ajiken4610-task-mng.web.app/#app/t|Gy4mYgGASjdQdfiyAlQJ/t|V3p4hSKHZuhuhAKWAbGt

考察

初めてWebアプリケーションを作ってみましたが、思っていたよりうまくいっているように見えると思います。
これは、ソースコード全てを1度書き直しているためです。
はじめは、関数をJsファイルに直に書いていったのですが、やはり私はJava慣れしているので、
オブジェクト指向なしでは、大きいプロジェクトを書ききることができそうになかったため、オブジェクト指向で1度書き直しています。
そのおかげで、最後に実装したサブタスクの仕組みの実装は、本当に15分程度で終わりました。