

# Supabase について ➡

開発部 石川

# Supabase とは

- いわゆる BaaS (Backend as a Service) と呼ばれるもの
- Firebase の RDB 版
- オープンソース
- セルフホスティングも可能
- Web, App 対応

# 大まかな機能

- Database (PostgreSQL)
- Auth
- Storage
- Realtime
- Logging
- Edge Functions (Beta ~ 2022/08/01)
- GraphQL

etc...

Firebase と違って、Analytics、WebPush はない

## 具体例、サンプルコード (Web) の紹介

## 1 Supabase プロジェクトの初期化

- <https://supabase.com> から GitHub ログインして、プロジェクト名やパスワード、リージョン等を設定。
- 設定後、<https://app.supabase.com/project/<projectId>> から、API エンドポイントと anon key をコピー

## テーブル作成

- SQL Editor か、ダッシュボード 上でテーブルを定義
- Supabase によって auth スキーマに users テーブルがデフォルトで定義されており、ここは変更不可。基本的に public スキーマ上にテーブルを定義する。

## 2 SDK のインストール、クライアントの初期化

```
npm i @supabase/supabase-js
```

```
import { createClient } from "@supabase/supabase-js";  
  
const SUPABASE_URL = process.env.NEXT_PUBLIC_SUPABASE_URL;  
const SUPABASE_ANON_KEY = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY;  
  
export const supabase = createClient(SUPABASE_URL, SUPABASE_ANON_KEY);
```

### 3 認証

```
import { supabase } from "どっか"; // 2で初期化したインスタンスをimport

// ユーザー情報を取得
const user = supabase.auth.user();

// ログイン
const { user, session, error } = await supabase.auth.signIn({
  email: 'example@email.com',
  password: 'example-password',
});
```

- SMS 認証、email による招待なども可能
- プロバイダーも豊富
  - Apple, Discord, Facebook, GitHub, Google, Notion, Twitter...



## 4 データ取得

すべての tasks テーブルのデータを取得

```
const { data, error } = await supabase.from<Task>("tasks").select("*")  
// .eq({ user_id: "hoge" })
```

コメント部分のようにフィルターも可能だが、後述する RLS とどちらを使用するかは要検討

## Realtime

- Websocket を用いた Realtime 通信が可能
- ダッシュボードから、適用したいテーブルの設定をオンにすることで使用可能

```
const tasks = supabase
  .from('tasks')
  .on('*', payload => {
    console.log('Change received!', payload)
  })
  .subscribe()
```

## 5 作成、更新、削除

```
const { data, error } = await supabase
  .from('tasks')
  .insert([ { some_column: 'someValue' } ], { upsert: true })
```

```
const { data, error } = await supabase
  .from('tasks')
  .update({ other_column: 'otherValue' })
  .eq('some_column', 'someValue')
```

```
const { data, error } = await supabase
  .from('tasks')
  .delete()
  .eq('some_column', 'someValue')
```

## 6 RLS (Row Level Security)

- PostgreSQL の機能
- 行単位でのセキュリティの設定が可能
- Supabase 側で、ポリシーで利用できる便利関数がいくつか用意されている。
  - `auth.uid()` -> リクエストを行ったユーザーの ID を返す、など

## 具体例

```
CREATE POLICY "policy_name"  
ON public.tasks  
FOR SELECT USING (  
    auth.uid() = user_id  
);
```

↑ tasks テーブルを Read したときに、リクエストを行ったユーザー ID と tasks.user\_id が一致するレコードしか返さない

**細かい認証ルールが必要な場合には、基本的にこの RLS のポリシーで設定する**

## おまけ (GraphQL)

- Supabase で GraphQL も利用可能に 🎉
- ダッシュボード で少しポチポチして、SQL を実行するだけですぐ利用できる。
- 自動で Query, Mutation を生成

```
comment on schema public is e'@graphql({"inflect_names": true});
```

```
select graphql.rebuild_schema();
```

## その他

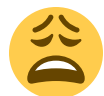
- トランザクションや副作用をもった処理は、`Database Functions` や `Triggers` を使えばできそう
- 全文検索も可能
- 匿名ログインができない
  - → Email 認証の設定で、メールを確認しなくても登録可にすることでそれっぽくはできる
- `Edge Functions` は Deno ランタイム上で JS,TS を実行できる（ユースケースがまだあんまわかってない）
- `supabase-cli` と Docker を使ったローカル開発も可能。ただ、UI 実装が追いついてない...

**所感**





- 爆速でバックエンド作れて幸せ
- API も直感的でわかりやすいし、GraphQL も使える
- 個人、小規模のプロダクトなら全然 Supabase で良さそう
- Twitter で「Supabase」っていれてつぶやくと、中の人たまに反応してくれる（質問答えてくれた）



- ダッシュボード だけで完結するのは厳しそう。SQL の知識は必要だと感じた
  - Enum、RLS、Delete Cascade など
- すべての機能が Production Ready なわけではないので、注意は必要
- サブドメイン間の認証状態の共有ができなそう

**まだ完全に使いこなせているわけではないが、結構良い感じ。伸びているので将来性もありそう**

**個人開発等でぜひさわってみてください！**

**ご清聴ありがとうございました**