Supabase 完全ガイド

動画CM分析アプリにおける導入・実装・拡張の全体像

本ドキュメントでは、動画CM分析アプリケーションにおいて
Supabase をなぜ・どのように導入したかを
初学者にもわかりやすく解説します。

作成日: 2026年2月9日
プロジェクト: 動画CM書き起こし・分析アプリ

目次

1. Supabase とは何か

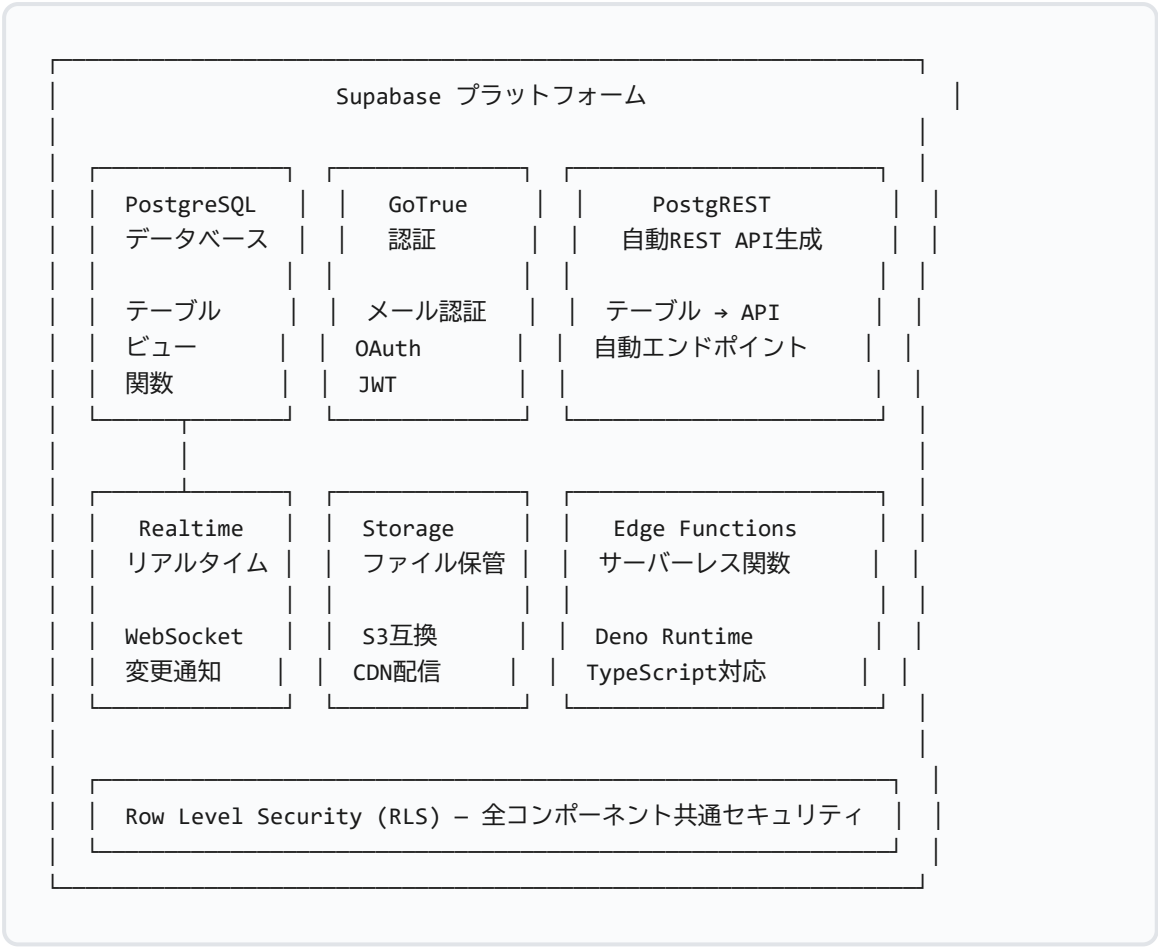
1.1 誕生の背景と思想

Supabase（スーパーベース）は、2020年にシンガポールで設立されたオープンソースのBaaS（Backend as a Service）プラットフォームです。Google が提供する Firebase の代替として生まれましたが、根本的な設計思想が異なります。

> **Supabase の設計思想**
>
> - **オープンソース:** すべてのコンポーネントがオープンソース。ベンダーロックインなし
> - **PostgreSQL 中心:** 世界で最も信頼されるリレーショナルDBを基盤にしている
> - **既存ツールの活用:** 新しい独自技術ではなく、実績のあるOSSを組み合わせている
> - **標準SQL:** 独自クエリ言語ではなく、業界標準のSQLがそのまま使える

1.2 主要コンポーネントの全体像

Supabase は単一の製品ではなく、複数のオープンソースツールを統合したプラットフォームです。



1.3 Firebase との違い

比較項目	Firebase	Supabase
提供元	Google（クローズドソース）	Supabase Inc.（オープンソース）
データベース	Firestore（NoSQL / ドキュメント型）	PostgreSQL（リレーショナル / SQL）
クエリ言語	独自API（制限あり）	標準SQL（制限なし）
テーブル結合	できない（非正規化が必要）	JOIN で自由に結合可能
データ整合性	アプリ側で担保	外部キー・制約でDB側が担保
セルフホスト	不可能	Docker で自社サーバーに設置可能
ベンダーロックイン	高い（Google依存）	低い（標準的な技術の組み合わせ）
Hosting	Firebase Hosting あり	なし（別途必要）
無料枠 DB	Firestore: 1GiB	PostgreSQL: 500MB
無料枠 Storage	Cloud Storage: 5GB	1GB

なぜ両方を使うのか？

Firebase は **Hosting（静的ファイル配信）** が非常に優秀で無料枠も大きいですが、データベースの Firestore は複雑なクエリに向きません。Supabase は逆に **データベースとストレージ** が強力ですが Hosting 機能がありません。それぞれの得意分野を組み合わせることで、**すべて無料枠内で本格的なアプリ** を構築しています。

2. なぜ Firebase だけでは完結しなかったのか

2.1 Firebase Hosting の役割と限界

Firebase Hosting は、ビルドした HTML / CSS / JavaScript ファイルを世界中の CDN（Content Delivery Network）に配置し、ユーザーに高速に配信するサービスです。いわば**「Webサイトの看板を掲げる場所」**です。

Firebase Hosting ができること

- HTML/CSS/JS を世界中に高速配信
- HTTPS（暗号化通信）を自動で設定
- カスタムドメインの設定
- デプロイが `firebase deploy` 一発
- 月10GBまで無料のデータ転送

Firebase Hosting ができないこと

- データの保存・検索・更新
- ファイル（動画・画像）の保管
- ユーザー認証の管理
- サーバー側のプログラム実行
- 複数ユーザー間のデータ共有

2.2 BaaS（Backend as a Service）の必要性

Web アプリケーションは通常、**フロントエンド**（画面表示）と**バックエンド**（データ管理）の2層で構成されます。

```
通常のWebアプリの構成:

  ユーザー ──→ フロントエンド ──→ バックエンド ──→ データベース
  (ブラウザ)    (HTML/CSS/JS)     (サーバー)       (データ保管)

本プロジェクトでの構成:

  ユーザー ──→ Firebase Hosting ──→ Supabase
  (ブラウザ)    (フロントエンド)      (バックエンド全部)
                                    ├── PostgreSQL (データ)
                                    ├── Storage (動画ファイル)
                                    └── Realtime (リアルタイム通知)
```

従来であれば、バックエンドには自分でサーバーを用意し、Node.js や Python などでプログラムを書き、データベースをインストール・管理する必要がありました。これには専門知

識が必要で、サーバーの維持費もかかります。

BaaS（Backend as a Service）は、このバックエンド部分をクラウドサービスとして提供してくれるものです。サーバーの構築・運用をすべてサービス側が行うため、開発者はフロントエンドの開発に集中できます。

2.3 Supabase を選んだ理由

> **本プロジェクトの要件**
>
> 1. **複数人でのデータ共有:** チームで同じ動画・分析データを閲覧・編集したい
> 2. **複雑なデータ関係:** 動画 → 書き起こし → コンバージョン → 分析結果 という多段階の関連データ
> 3. **大容量ファイル保管:** 動画ファイル（数十MB〜数百MB）を安全に保管
> 4. **無料で運用:** 個人・小規模チームのため、コストをゼロに抑えたい
> 5. **将来の拡張性:** ユーザー認証やAI機能の追加に対応できること

これらすべてを満たすのが Supabase でした。特に、**PostgreSQL による複雑なデータ管理**と**Storage によるファイル保管**が両方無料枠で使えることが決定的でした。

> **移行の経緯**
>
> 本プロジェクトは以下の段階を経て現在の構成になりました:
>
> 1. **Phase 1: Firebase Firestore** → NoSQL のため複雑なクエリに限界
> 2. **Phase 2: ブラウザ内 IndexedDB** → オフラインで動くが、複数人で共有できない
> 3. **Phase 3: Supabase PostgreSQL** → すべての要件を満たす（現在の構成）

3. Supabase の主要機能 — 詳細解説

3.1 PostgreSQL データベース

PostgreSQL とは

PostgreSQL（ポストグレスキューエル）は、1996年に最初のバージョンがリリースされた**世界で最も先進的なオープンソース・リレーショナルデータベース**です。30年近い歴史の中で、銀行、政府、大企業のシステムで使われてきた信頼性があります。

リレーショナルデータベースの基本概念

データを**テーブル（表）**の形で管理します。Excel のスプレッドシートに似ていますが、テーブル同士を**関連付ける（リレーション）**ことができます。

テーブルのイメージ:

```
  videos テーブル（動画の情報）
┌────┬──────────────────┬────────────┬────────────┐
│ id │ filename         │ status     │ duration   │
├────┼──────────────────┼────────────┼────────────┤
│  1 │ CM_夏セール.mp4  │ transcribed│ 30.5       │
│  2 │ CM_新商品.mp4    │ uploaded   │ 15.2       │
│  3 │ CM_キャンペーン  │ archived   │ 45.0       │
└────┴──────────────────┴────────────┴────────────┘
        │
        │ video_id で関連付け（外部キー）
        ▼
  transcriptions テーブル（書き起こし）
┌────┬────────────┬──────────────────────────────┐
│ id │ video_id   │ full_text                    │
├────┼────────────┼──────────────────────────────┤
│  1 │     1      │ この夏、最大50%オフ...       │
│  2 │     3      │ 期間限定キャンペーン...      │
└────┴────────────┴──────────────────────────────┘
```

SQL の基本操作

SQL（Structured Query Language）は、データベースを操作するための標準言語です。

```sql
-- データの取得（SELECT）
SELECT filename, status FROM videos WHERE status = 'transcribed';

-- データの挿入（INSERT）
INSERT INTO videos (id, filename, status) VALUES (4, '新CM.mp4', 'uploaded');

-- データの更新（UPDATE）
UPDATE videos SET status = 'transcribed' WHERE id = 2;

-- テーブル結合（JOIN）— 動画と書き起こしを一緒に取得
SELECT v.filename, t.full_text
FROM videos v
JOIN transcriptions t ON v.id = t.video_id;
```

本プロジェクトで活用している PostgreSQL 機能

機能	説明	活用場面
外部キー制約	テーブル間の参照整合性を保証	動画削除時に書き起こしも自動削除（CASCADE）
CHECK 制約	カラムの値を制限	status が決められた値以外に設定されるのを防止
JSONB 型	JSON データを効率的に保存・検索	タグ配列、サムネイル情報、セグメントデータ
TIMESTAMPTZ	タイムゾーン付き日時	作成日時・更新日時の自動管理
DEFAULT 値	挿入時の初期値	`status = 'uploaded'`, `tags = '[]'` など
インデックス	検索の高速化	status, ranking, created_at での検索を高速化

3.2 Storage（ファイルストレージ）

Supabase Storage は、画像・動画・ドキュメントなどの**大きなファイル**を保管するサービスです。Amazon S3 と互換性のある API を持ち、ファイルは「バケット」と呼ばれるフォルダのような単位で管理されます。

```
Storage の構造:

  videos バケット（非公開）
  ├── 1/                          ← 動画ID=1 のフォルダ
  │   ├── video.mp4               ← 元の動画ファイル
  │   ├── thumb_0.jpg             ← 0秒時点のサムネイル
  │   ├── thumb_5.jpg             ← 5秒時点のサムネイル
  │   └── thumb_10.jpg            ← 10秒時点のサムネイル
  ├── 2/
  │   └── video.mp4
  └── 3/
      ├── thumb_0.jpg             ← アーカイブ後（動画削除済み）
      ├── thumb_5.jpg
      └── thumb_10.jpg
```

署名付き URL（Signed URL）

非公開バケットのファイルにアクセスするために、**期限付きの一時的な URL** を生成します。この URL を知っている人だけが、指定された時間内にファイルにアクセスできます。

```javascript
// 署名付き URL の生成（4時間有効）
const { data } = await supabase.storage
  .from('videos')
  .createSignedUrl('1/video.mp4', 60 * 60 * 4);

// 結果: https://xxx.supabase.co/storage/v1/object/sign/videos/1/video.mp4?token=
// この URL は 4時間後に無効になる
```

3.3 Authentication（認証）

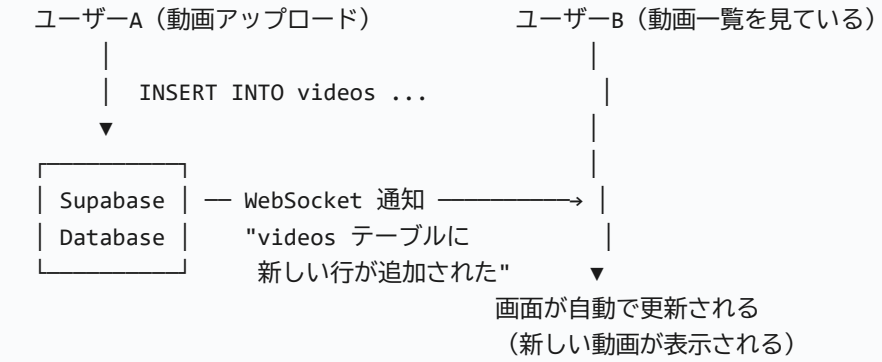
Supabase Auth（内部的には GoTrue というOSSを使用）は、ユーザーの本人確認を行うサービスです。**本プロジェクトでは独自のパスワードゲートを実装しているため、Supabase Auth は使用していません**が、将来の拡張として活用可能です。

対応する認証方式

認証方式	説明
メール & パスワード	最も基本的な方式。メール確認付き
マジックリンク	パスワード不要。メールに送られたリンクをクリックでログイン
OAuth（ソーシャルログイン）	Google, GitHub, Apple, Twitter, Facebook, LINE など20以上に対応
電話番号（SMS）	SMSで送られるコードで認証
SAML SSO	企業のシングルサインオン（有料プラン）

3.4 Realtime（リアルタイム通信）

データベースの変更をリアルタイムで全クライアントに通知する機能です。WebSocket 技術を使い、ページをリロードしなくてもデータが自動更新されます。




```
// Realtime の購読（本プロジェクトでの実装）
supabase
  .channel('videos-changes')
  .on('postgres_changes', {
    event: '*',        // INSERT, UPDATE, DELETE すべて
    schema: 'public',
    table: 'videos'
  }, (payload) => {
    console.log('変更検知:', payload);
    refreshVideoList();  // 画面を更新
  })
  .subscribe();
```

3.5 Edge Functions（サーバーレス関数）

サーバー上でコードを実行できる機能です。Deno（DenoはNode.jsの作者が開発した新しいJavaScript/TypeScriptランタイム）で動作し、TypeScript をそのまま実行できます。**本プロジェクトでは未使用**ですが、拡張として活用可能です。

> **Edge Functions が有用な場面**
>
> - APIキーをサーバー側で安全に管理したい場合
> - 外部APIとの連携（Webhook受信など）
> - 重い計算処理をサーバーで実行したい場合
> - 定期的なバッチ処理（日次レポート生成など）

3.6 Row Level Security（RLS）

RLS は Supabase の最も重要なセキュリティ機能です。データベースの各行（レコード）に対して「誰がアクセスできるか」をSQL で定義します。

RLS の概念

```
RLS なし:
  全ユーザーが全データにアクセス可能（危険）

RLS あり:
  ┌──────────────────────────────────────────┐
  │ videos テーブル                          │
  ├────┬──────────────┬──────────────┬───────┤
  │ id │ filename     │ owner_id     │ status │
  ├────┼──────────────┼──────────────┼───────┤
  │  1 │ CM_A.mp4     │ user_001     │ ✅ 見える │  ← user_001 がログイン中
  │  2 │ CM_B.mp4     │ user_002     │ ❌ 見えない│
  │  3 │ CM_C.mp4     │ user_001     │ ✅ 見える │
  └────┴──────────────┴──────────────┴───────┘

  ポリシー例: WHERE owner_id = auth.uid()
```

本プロジェクトでの RLS 設定

```sql
-- 1. RLS を有効化（これにより、デフォルトで全アクセスがブロックされる）
ALTER TABLE videos ENABLE ROW LEVEL SECURITY;

-- 2. ポリシーを作成（anon ユーザーに全操作を許可）
CREATE POLICY "Allow all for anon" ON videos
  FOR ALL
  USING (true)          -- SELECT/UPDATE/DELETE 時の条件（全許可）
  WITH CHECK (true);    -- INSERT/UPDATE 時の条件（全許可）
```

> **なぜ全許可なのか？**
>
> 本プロジェクトでは、アプリにアクセスする前に **パスワードゲート**（SHA-256 ハッシュによる認証画面）を設けています。パスワードを知っている人 = チームメンバー = 全データにアクセス可能、という設計のため、DB側では全許可にしています。もし個人ごとのアクセス制御が必要になった場合は、Supabase Auth と組み合わせて RLS ポリシーを厳格化できます。

3.7 Auto-generated API（自動生成 REST API）

Supabase の大きな特徴の一つが、**テーブルを作成するだけで REST API が自動的に生成される**ことです。内部的には PostgREST というOSSが動いています。

```javascript
// テーブルを作るだけで、以下のようなAPI操作が自動で可能になる

// 全件取得
const { data } = await supabase.from('videos').select('*');

// 条件付き取得
const { data } = await supabase.from('videos')
  .select('*')
  .eq('status', 'transcribed')
  .order('created_at', { ascending: false });

// 挿入
await supabase.from('videos').insert({ filename: '新CM.mp4', status: 'uploaded' }

// 更新
await supabase.from('videos').update({ status: 'transcribed' }).eq('id', 1);

// 削除
await supabase.from('videos').delete().eq('id', 1);
```

これにより、バックエンドのAPI開発が**ゼロ**になります。通常であれば Express.js や FastAPI などでエンドポイントを一つ一つ実装する必要がありますが、Supabase では テーブル設計 = API設計 となります。

4. 本プロジェクトでの実装手順 — Step by Step

1 プロジェクト作成と接続設定

Supabase 側の作業

1. supabase.com でアカウント作成
2. 「New Project」で新規プロジェクトを作成
3. リージョン（サーバーの場所）を選択 — 日本からのアクセスなら東京リージョン推奨
4. プロジェクト作成後、Settings → API から以下を取得:
   - **Project URL:** `https://xxxxx.supabase.co`
   - **anon public key:** `eyJhbGciOi...`（長い文字列）

フロントエンド側の作業

```
// .env ファイルに接続情報を設定
VITE_SUPABASE_URL=https://xxxxx.supabase.co
VITE_SUPABASE_ANON_KEY=eyJhbGciOi...

// ※ VITE_ プレフィックスにより、Vite ビルド時に環境変数として読み込まれる
```

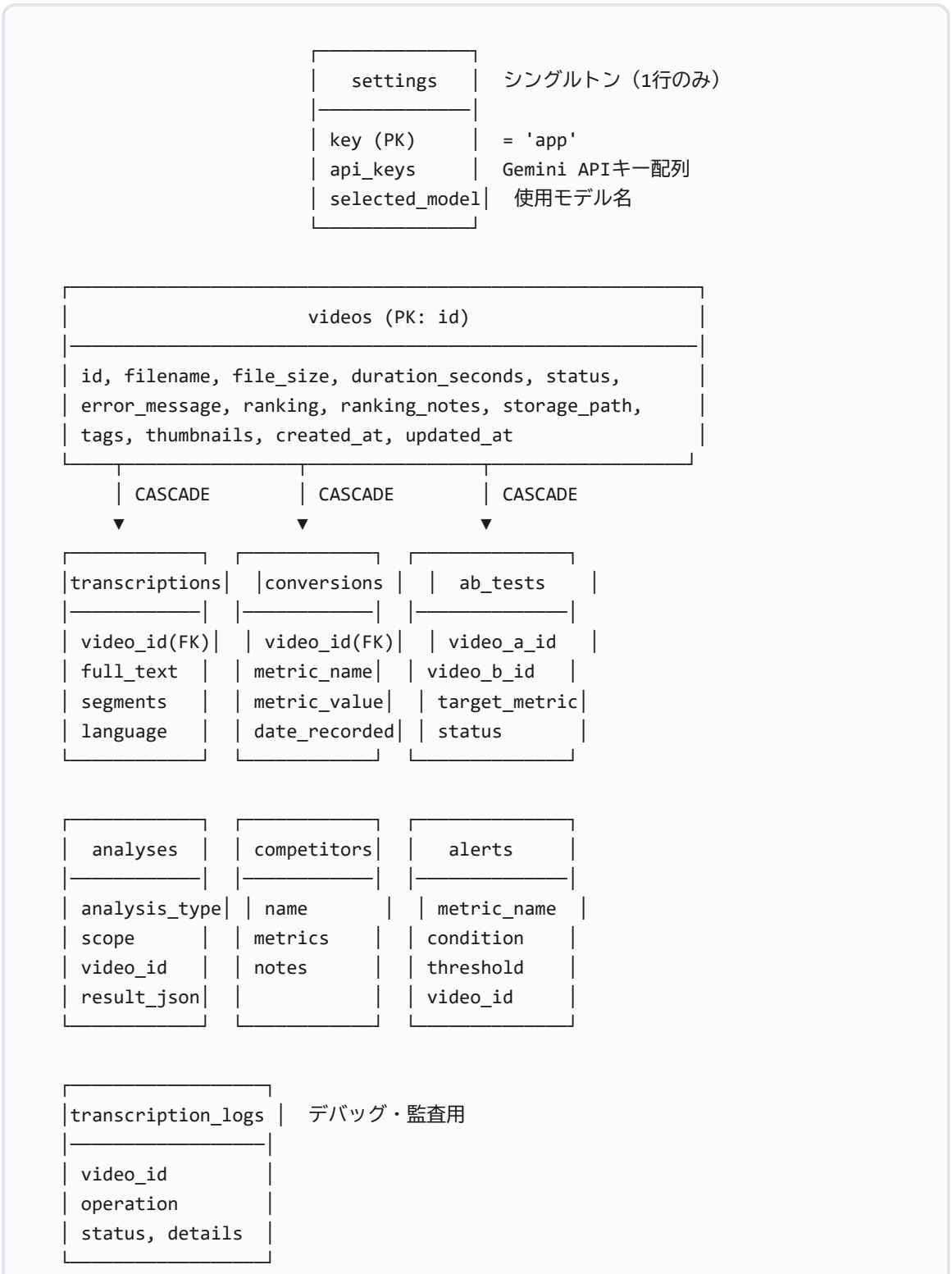
```
// services/supabase.ts — Supabase クライアントの初期化
import { createClient } from "@supabase/supabase-js";

const supabaseUrl = import.meta.env.VITE_SUPABASE_URL;
const supabaseAnonKey = import.meta.env.VITE_SUPABASE_ANON_KEY;

export const supabase = createClient(supabaseUrl, supabaseAnonKey);
```

> **シングルトンパターン**
>
> Supabase クライアントはアプリ全体で**1つだけ**作成し、全ファイルから共有します。
> 複数のクライアントを作ると、WebSocket 接続が無駄に増え、メモリを消費します。

2 データベース設計（全9テーブル）

Supabase の SQL エディタで以下のスキーマを実行しました。各テーブルの設計理由を詳細に解説します。

テーブル関連図（ER図）



各テーブルの詳細設計

テーブル	主な列	設計の理由
videos	filename, status, tags, thumbnails, ranking	アプリの中核。status で処理状態を管理。tags(JSONB) で媒体分類。ON DELETE CASCADE で関連データを自動削除
transcriptions	video_id, full_text, segments, language	AI書き起こし結果。segments(JSONB) でタイムスタンプ付きテキストを保持
conversions	video_id, metric_name, metric_value	動画ごとの成果指標（CTR, CVR等）。metric_name で柔軟な指標定義
analyses	analysis_type, result_json, gemini_model_used	AI分析結果のキャッシュ。再分析不要で過去結果を参照可能
settings	api_keys, selected_model	アプリ全体設定。1行のみ（key='app'）。チーム全員で共有
ab_tests	video_a_id, video_b_id, target_metric, status	2つの動画のA/B比較テスト。結果はconversionsから動的計算
competitors	name, metrics(JSONB)	競合のベンチマークデータ。metrics を JSONB にして柔軟に指標を追加可能
alerts	metric_name, condition, threshold, enabled	指標監視ルール。condition('above'/'below') と threshold で自動チェック
transcription_logs	video_id, operation, status, details	書き起こし処理のログ。エラー追跡・デバッグ用

重要な設計パターン

ON DELETE CASCADE（カスケード削除）

動画を削除すると、関連する書き起こし・コンバージョンデータが**自動的に**削除されます。これにより、「孤立データ」（親のいない子データ）が発生することを防ぎます。

```
-- 動画ID=1を削除すると...
DELETE FROM videos WHERE id = 1;

-- transcriptions の video_id=1 の行も自動削除
-- conversions の video_id=1 の行も自動削除
-- 手動で削除する必要なし！
```

CHECK 制約（値の制限）

status カラムに無効な値が入ることを防ぎます。アプリのバグで不正な値を保存しようとすると、データベースがエラーを返します。

```
CHECK (status IN ('uploaded', 'transcribing', 'transcribed', 'error', 'arch

-- これにより:
-- status = 'transcribed'  → OK
-- status = 'hoge'         → エラー！挿入できない
```

JSONB 型の活用

リレーショナルDBでありながら、柔軟な構造のデータも扱えるのが PostgreSQL の強みです。タグや分析結果のように**構造が変わりうるデータ**は JSONB 型で保存しています。

```
-- tags の例: ["YouTube", "TikTok", "Instagram"]
-- thumbnails の例: [{"time": 0, "storage_path": "1/thumb_0.jpg"}, ...]
-- segments の例: [{"start_time": 0, "end_time": 5.2, "text": "こんにちは"},
```

3 RLS ポリシー設定

```
-- 全9テーブルで同じパターン:

-- ① RLS を有効化（デフォルトで全アクセスブロック）
ALTER TABLE videos ENABLE ROW LEVEL SECURITY;

-- ② anon ロールに全操作を許可するポリシーを作成
CREATE POLICY "Allow all for anon" ON videos
  FOR ALL USING (true) WITH CHECK (true);
```

セキュリティの多層防御

「全許可」に見えますが、以下の多層防御でセキュリティを確保しています:

1. **パスワードゲート:** アプリ自体にパスワードが必要
2. **anon key の制限:** Supabase の anon key は RLS ポリシーに従った操作のみ可能
3. **Storage の非公開設定:** ファイルは署名付きURLでのみアクセス可能
4. **HTTPS:** 通信はすべて暗号化

4 Storage バケット設定

```
-- videos バケットの作成（非公開）
INSERT INTO storage.buckets (id, name, public)
  VALUES ('videos', 'videos', false)
  ON CONFLICT (id) DO NOTHING;

-- Storage のアクセスポリシー
CREATE POLICY "Allow all storage for anon" ON storage.objects
  FOR ALL
  USING (bucket_id = 'videos')
  WITH CHECK (bucket_id = 'videos');
```

`public: false` により、ファイルに直接アクセスするURLは存在しません。必ず署名付きURL を生成してアクセスする必要があります。

5 フロントエンド接続コード

アプリケーションから Supabase を利用するために、以下のサービス層を実装しました。

services/supabase.ts — 接続

```
import { createClient } from "@supabase/supabase-js";

export const supabase = createClient(
  import.meta.env.VITE_SUPABASE_URL,
  import.meta.env.VITE_SUPABASE_ANON_KEY
);
```

services/videoStorage.ts — ファイル操作

```
// アップロード
export async function uploadVideo(path: string, file: File) {
  await supabase.storage.from("videos").upload(path, file);
}

// 署名付き URL 生成（4時間有効、3.5時間キャッシュ）
export async function getVideoSignedUrl(path: string) {
  const { data } = await supabase.storage
    .from("videos")
    .createSignedUrl(path, 60 * 60 * 4);
  return data.signedUrl;
}

// 削除
export async function deleteFromStorage(path: string) {
  await supabase.storage.from("videos").remove([path]);
}
```

6 CRUD 操作の抽象化

services/db.ts — 共通データ操作

```
// 汎用的なCRUD関数で、全テーブルの操作を統一

export async function put<T>(table: string, record: T) {
  await supabase.from(table).upsert(record as any);
}

export async function get<T>(table: string, id: number): Promise<T | null> {
  const { data } = await supabase.from(table)
    .select("*").eq("id", id).single();
  return data as T;
}

export async function getAll<T>(table: string): Promise<T[]> {
  const { data } = await supabase.from(table).select("*");
  return (data ?? []) as T[];
}

export async function del(table: string, id: number) {
  await supabase.from(table).delete().eq("id", id);
}

export async function update<T>(table: string, id: number, fields: Partial<T>) {
  await supabase.from(table).update(fields as any).eq("id", id);
}
```

抽象化のメリット

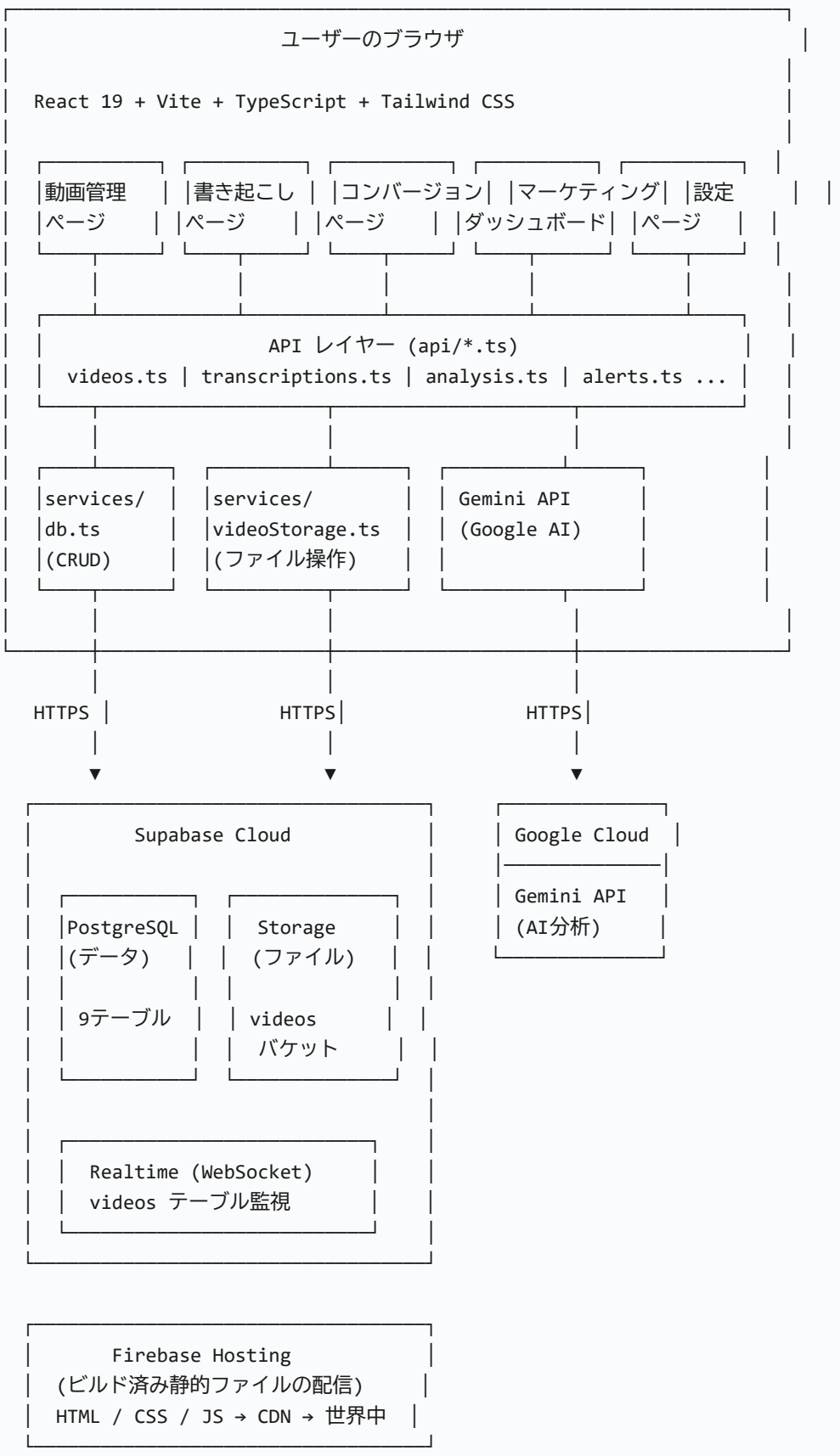
この共通関数により、各APIファイル（api/videos.ts, api/analysis.ts 等）は**ビジネスロジックだけに集中**できます。データベースアクセスの詳細は db.ts に隠蔽されているため、もし将来 Supabase から別のサービスに移行する場合も、db.ts だけを修正すれば済みます。

7 Realtime の活用

```
// videos テーブルのリアルタイム更新を有効化（SQL）
ALTER PUBLICATION supabase_realtime ADD TABLE videos;
```

これにより、あるユーザーが動画をアップロード・編集・削除すると、他のユーザーのブラウザにもリアルタイムで変更が通知されます。

5. アーキテクチャ全体図 — 3サービスの連携


ユーザーのブラウザ
React 19 + Vite + TypeScript + Tailwind CSS
動画管理ページ
書き起こしページ
コンバージョンページ
マーケティングダッシュボード
設定ページ
API レイヤー (api/*.ts)
videos.ts | transcriptions.ts | analysis.ts | alerts.ts ...
services/db.ts (CRUD)
services/videoStorage.ts (ファイル操作)
Gemini API (Google AI)
HTTPS
HTTPS
HTTPS
Supabase Cloud
PostgreSQL (データ)
9テーブル
Storage (ファイル)
videos バケット
Realtime (WebSocket)
videos テーブル監視
Google Cloud
Gemini API (AI分析)
Firebase Hosting
(ビルド済み静的ファイルの配信)
HTML / CSS / JS → CDN → 世界中

各サービスの役割まとめ

サービス	役割	たとえ
Firebase Hosting	アプリ本体の配信	お店の建物・外観
Supabase Database	データの保管・検索	お店の帳簿・台帳
Supabase Storage	動画・画像ファイルの保管	お店の倉庫
Supabase Realtime	変更のリアルタイム通知	店内放送システム
Google Gemini API	AI による分析・提案	外部コンサルタント

6. データの流れ — 主要ユースケース

6.1 動画アップロード → 書き起こし → 分析

① 動画アップロード
ユーザー → [ファイル選択] → videoStorage.uploadVideo()
→ Supabase Storage にファイル保存
→ db.put('videos', {...}) で videos テーブルに登録
→ status = 'uploaded'

② 書き起こし
ユーザー → [書き起こし開始]
→ db.update('videos', id, {status: 'transcribing'})
→ videoStorage.getVideoSignedUrl() で動画URLを取得
→ Gemini API に動画を送信 → 書き起こしテキスト受信
→ db.put('transcriptions', {...}) で結果を保存
→ db.update('videos', id, {status: 'transcribed'})

③ AI分析
ユーザー → [分析実行]
→ db.getAll('transcriptions') で全書き起こし取得
→ db.getAll('conversions') で全コンバージョン取得
→ Gemini API にデータを送信 → 分析結果受信
→ db.put('analyses', {...}) で結果を保存

④ アーカイブ（ストレージ節約）
ユーザー → [アーカイブ]
→ videoStorage.getVideoSignedUrl() で動画URL取得
→ Canvas API でサムネイル抽出（5秒間隔）
→ videoStorage.uploadThumbnail() でサムネイル保存
→ db.update('videos', id, {status:'archived', thumbnails:[...]})
→ videoStorage.deleteFromStorage() で動画ファイル削除
→ ストレージ使用量が大幅に削減（動画10MB → サムネイル200KB）

6.2 媒体別分析（プラットフォーム分析）

① 媒体タグ付け
ユーザー → [YouTube] [TikTok] などのボタンをクリック
→ db.update('videos', id, {tags: ["YouTube", "TikTok"]})
→ video.tags にプラットフォーム情報が保存される

② AI分析実行
ユーザー → [媒体分析を実行]
→ db.getAll('videos') → タグで媒体別にグルーピング
→ 各媒体の書き起こし・コンバージョンデータを集約
→ Gemini API に媒体特性を含むプロンプトを送信:
YouTube: 長尺OK、SEO重要、サムネイル/フック重視
TikTok: 15-60秒、最初1秒が勝負、縦型
Instagram: ビジュアル重視、ストーリーズ/リール
Facebook: 30代以上、シェア誘導、コミュニティ感
LINE: 短尺、直接的CTA、クーポン連動
X(Twitter): 短いメッセージ、リツイート誘導
→ 分析結果を表示 + db.put('analyses', {...}) で保存

7. セキュリティ設計

セキュリティの多層防御:

```
第1層: Firebase Hosting
└── HTTPS（TLS暗号化）でデータ通信を保護

第2層: パスワードゲート（PasswordGate.tsx）
└── SHA-256 ハッシュでパスワード照合
└── パスワードを知らない人はアプリ自体を使えない

第3層: Supabase anon key
└── RLS ポリシーで許可された操作のみ実行可能
└── service_role key はフロントエンドに含めない

第4層: Row Level Security (RLS)
└── 各テーブルにアクセス制御ポリシーを定義
└── SQL レベルでデータ保護

第5層: Storage 署名付き URL
└── ファイルは期限付きURL（4時間）でのみアクセス可能
└── URL を知っていても期限切れならアクセス不可

第6層: 環境変数
└── API キー等は .env ファイルで管理
└── Git リポジトリには含めない（.gitignore）
```

anon key はフロントエンドに含まれる — これは安全か？

Supabase の anon key はフロントエンドの JavaScript に含まれるため、ブラウザの開発者ツールで確認できます。しかし、**これは設計通り**です。anon key でできることは RLS ポリシーで厳密に制限されているため、key が漏洩しても RLS が許可した操作以外は実行できません。

一方、**service_role key**（管理者権限）は絶対にフロントエンドに含めてはいけません。この key は RLS を無視して全データにアクセスできます。

8. 無料枠の詳細と運用戦略

リソース	無料枠	現在の使用量	残り
Supabase Database	500 MB	約 5 MB	約 495 MB（99%）
Supabase Storage	1 GB	約 37 MB（3.7%）	約 963 MB（96.3%）
Supabase 帯域幅	5 GB / 月	少量	ほぼ全量
Supabase API リクエスト	無制限（※）	-	-
Firebase Hosting 転送量	10 GB / 月	少量	ほぼ全量
Firebase Hosting ストレージ	1 GB	ビルド済みJS約数MB	ほぼ全量

※ Supabase 無料枠の API リクエストに明示的な上限はありませんが、同時接続数やレート制限があります。

ストレージ容量の目安

動画の保存可能数（1GB上限）

動画の長さ	1本あたりサイズ（目安）	保存可能本数
15秒 CM	約 5 MB	約 190 本
30秒 CM	約 10 MB	約 96 本
1分 CM	約 20 MB	約 48 本
3分 動画	約 60 MB	約 16 本

アーカイブ機能を使えば: 動画 10MB → サムネイル 200KB（約98%削減）。書き起こし完了済みの動画をアーカイブすれば、はるかに多くの動画を管理できます。

9. 拡張可能性 — Supabase でできること

現在使用していない Supabase の機能を活用することで、以下のような拡張が可能です。

9.1 ユーザー認証（Supabase Auth）

現在 → 拡張後

現在: 共有パスワード1つでチーム全員がアクセス

拡張後: 個人アカウントで管理。Google/GitHub ログイン対応。ユーザーごとの権限設定（閲覧のみ、編集可能、管理者）

```
// 拡張例: Google ログインの実装
const { data, error } = await supabase.auth.signInWithOAuth({
  provider: 'google',
  options: { redirectTo: 'https://movie-analysis-d05fb.web.app/callback' }
});

// RLS を厳格化: 自分のデータのみ操作可能
// CREATE POLICY "Users can manage own videos" ON videos
//   USING (owner_id = auth.uid());
```

9.2 Edge Functions（サーバーレス関数）

現在 → 拡張後

現在: Gemini API キーがフロントエンド（ブラウザ）に存在

拡張後: API キーをサーバー側（Edge Function）で安全に管理。ブラウザからはキー不要

```
// Edge Function の例 (supabase/functions/analyze/index.ts)
import { serve } from "https://deno.land/std/http/server.ts";

serve(async (req) => {
  const GEMINI_KEY = Deno.env.get("GEMINI_API_KEY"); // サーバー側で安全に保持
  const { videoId } = await req.json();

  // Gemini API を呼び出し
  const result = await callGemini(GEMINI_KEY, videoId);

  return new Response(JSON.stringify(result));
});
```

9.3 Database Functions & Triggers（データベース関数とトリガー）

現在 → 拡張後

現在: updated_at はアプリ側で手動更新

拡張後: データ変更時に自動で updated_at を更新、統計値の自動計算

```
-- updated_at 自動更新トリガー
CREATE OR REPLACE FUNCTION update_updated_at()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at = now();
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER videos_updated_at
  BEFORE UPDATE ON videos
  FOR EACH ROW EXECUTE FUNCTION update_updated_at();
```

9.4 全文検索（PostgreSQL Full Text Search）

現在 → 拡張後

現在: フロントエンドでのテキスト検索（ブラウザのメモリで処理）

拡張後: PostgreSQL のインデックスによる高速全文検索。数万件でも一瞬で検索

```sql
-- 全文検索インデックスの作成
ALTER TABLE transcriptions
  ADD COLUMN fts tsvector
  GENERATED ALWAYS AS (to_tsvector('japanese', full_text)) STORED;

CREATE INDEX idx_transcriptions_fts ON transcriptions USING GIN(fts);

-- 検索クエリ
SELECT * FROM transcriptions
WHERE fts @@ to_tsquery('japanese', '割引 & キャンペーン');
```

9.5 Vector 検索（pgvector — AI類似検索）

現在 → 拡張後

現在: AI分析は都度 Gemini API に問い合わせ

拡張後: テキストをベクトル化して保存 → 「この動画に似た動画」を瞬時に検索

```sql
-- pgvector 拡張を有効化
CREATE EXTENSION vector;

-- ベクトル列を追加
ALTER TABLE transcriptions
  ADD COLUMN embedding vector(768);

-- 類似検索（コサイン距離）
SELECT video_id, full_text,
       1 - (embedding <=> '[0.1, 0.2, ...]') AS similarity
FROM transcriptions
ORDER BY embedding <=> '[0.1, 0.2, ...]'
LIMIT 5;
```

9.6 Database Webhooks（外部連携通知）

現在 → 拡張後

現在: Realtime で画面更新のみ

拡張後: データ変更時に Slack 通知、メール送信、外部 API 呼び出し

```
// Webhook 設定例（Supabase Dashboard から設定）
// トリガー: videos テーブルに INSERT が発生したとき
// アクション: Slack Webhook URL にメッセージを送信
//
// 結果: 動画がアップロードされるたびに
// Slack チャンネルに「新しい動画がアップロードされました: CM_新商品.mp4」と通知
```

9.7 Supabase Cron（定期実行ジョブ）

現在 → 拡張後

現在: 手動でのみ分析・レポート作成

拡張後: 毎朝自動でレポート生成、週次で自動分析実行

```sql
-- pg_cron 拡張を使用（Supabase Pro プランで利用可能）
SELECT cron.schedule(
  'daily-report',
  '0 9 * * *',  -- 毎朝9時
  $$ SELECT generate_daily_report() $$
);
```

9.8 拡張機能の優先度マトリクス

拡張機能	実装難易度	ビジネス価値	無料枠で可能	優先度
DB Triggers（自動更新）	低	中	はい	高
全文検索	中	高	はい	高
Supabase Auth	中	高	はい	中
Edge Functions	中	中	はい（50万回/月）	中
Database Webhooks	低	中	はい	中
pgvector（AI類似検索）	高	高	はい	低
Cron ジョブ	低	中	Pro プランのみ	低

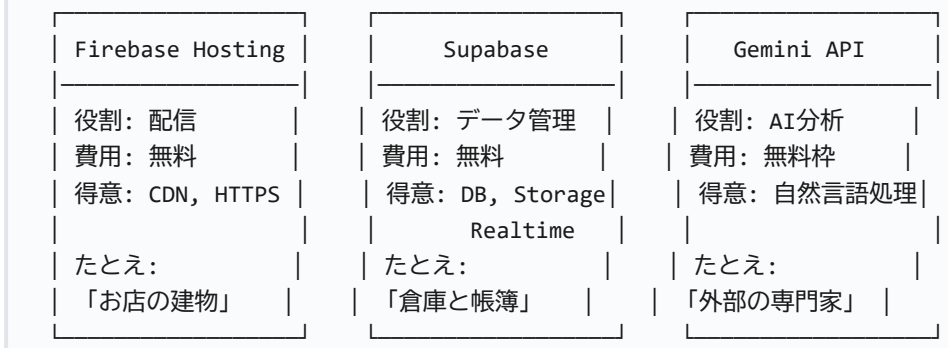
10. まとめ

本プロジェクトにおける Supabase の役割

Supabase は、Firebase Hosting だけでは実現できなかった**データ管理・ファイル保管・リアルタイム通信**を、無料枠の範囲内で提供してくれるバックエンドサービスです。

PostgreSQL という世界標準のデータベースを基盤としているため、学んだスキルは他のプロジェクトでもそのまま活用できます。また、オープンソースのためベンダーロックインのリスクが低く、将来的に自社サーバーに移行することも可能です。

3サービスの組み合わせ



すべて無料枠で、本格的なWebアプリケーションが稼働中
https://movie-analysis-d05fb.web.app

技術選定の教訓

1. **各サービスの得意分野を理解する:** Firebase は配信、Supabase はデータ管理、Gemini は AI。万能なサービスはない
2. **無料枠を賢く組み合わせる:** 複数サービスの無料枠を組み合わせれば、個人でも本格アプリが作れる
3. **標準技術を選ぶ:** SQL, REST API, WebSocket — 標準技術は学習コストが長期的に報われる

4. **段階的に拡張する:** 最初はシンプルに。必要になったら Auth, Edge Functions, pgvector を追加
5. **セキュリティは多層防御:** 1つの対策に頼らず、パスワードゲート + RLS + 署名付きURL + HTTPS を組み合わせる
