

🎨 ゆるVibe Pages クラス図

実装済みコードベースからリバースエンジニアリングした包括的なクラス図

システム概要

このクラス図は、ゆるVibe Pagesの実装済みコンポーネント、ライブラリ関数、データモデルの関係性を正確に表現しています。React コンポーネント階層、Firebase 連携、OpenAI API 統合の全体像を俯瞰できます。

メインクラス図



Syntax error in text
mermaid version 11.8.1

コンポーネント階層詳細

React Pages コンポーネント

RootLayout (最上位)

- 役割: アプリケーション全体のレイアウト・メタデータ

- **特徴:** フォント設定、OGP設定、viewport設定
- **子コンポーネント:** 全ページコンポーネント

HomePage (メインページ)

- **パス:** `/`
- **状態:** `theme`, `loading`, `error`
- **機能:** テーマ入力、詩生成トリガー
- **API呼び出し:** `/api/generate-storage`

ViewPoemPage (詩表示)

- **パス:** `/view/[id]`
- **動的ルート:** Next.js App Router
- **OGP:** 動的メタデータ生成
- **依存:** `FloatingParticles`, `BackgroundImage`

UI コンポーネント

FloatingParticles (アニメーション)

- **技術:** Canvas 2D API
- **内部クラス:** `Particle` (50個のインスタンス)
- **機能:** ふわふわ浮遊、境界反射、グラデーション

BackgroundImage (画像表示)

- **CORS対応:** `Firebase getBlob()` + フォールバック
- **状態:** `loaded`, `error`, `isLoading`, `finalImageUrl`
- **パフォーマンス:** `Object URL` 自動クリーンアップ

ライブラリ関数アーキテクチャ

Firestore 連携層

FirestoreService

```
interface FirestoreOperations {
    savePoemToFirestore(poemData: PoemDocument): Promise<string>
    getPoemFromFirestore(id: string): Promise<PoemDocument | null>
}
```

StorageService

```
interface StorageOperations {
    uploadImageToStorage(imageId: string, imageUrl: string): Promise<string>
    uploadBase64ImageToStorage(imageId: string, base64Data: string): Promise<string>
    deleteImageFromStorage(imageId: string): Promise<void>
}
```

FirebaseImageService (CORS特化)

```
interface ImageLoadOperations {
    loadFirebaseImageBlob(imageId: string): Promise<{success: boolean, blob: Blob}>
    loadPoemImage(poemId: string): Promise<{success: boolean, imageUrl: string}>
}
```

OpenAI 連携層

OpenAIService (GPT-4o)

```
interface PoemGeneration {
    generatePoem(theme: string): Promise<string>
    generateImagePrompt(theme: string, poem: string): Promise<string>
}
```

DalleService (DALL-E 3)

```
interface ImageGeneration {
    generateImage(prompt: string): Promise<string>
}
```

```
generateImageFromTheme(theme: string): Promise<{imageUrl: str  
}
```

データモデル

PoemDocument (Firestore)

```
interface PoemDocument {  
  id: string;           // nanoid  
  theme: string;        // ユーザー入力  
  phrase: string;       // GPT-4o生成詩  
  imageUrl: string;     // Storage URL または DALL-E URL  
  imagePrompt: string;  // DALL-E生成プロンプト  
  createdAt: Date;      // 作成日時  
}
```

APIResponse (統一形式)

```
interface APIResponse {  
  success: boolean;  
  data?: {  
    id: string;  
    phrase: string;  
    imageUrl: string;  
    theme: string;  
  };  
  error?: string;  
  timing?: {  
    total: number;  
    gpt: number;  
    dalle: number;  
    storage: number;  
  };  
}
```

API エンドポイント設計

6つのエンドポイント戦略

- **generate-storage**: 本番用（Storage保存）
- **generate-safe**: Safe版（フォールバック強化）
- **generate-simple**: Simple版（Storage回避）
- **generate-direct**: Direct版（DALL-E URL直接）
- **generate-dummy**: Dummy版（オフライン開発）
- **generate**: 基本版（標準実装）

フォールバック戦略

Storage保存成功 → Storage URL
↓（失敗）
DALL-E URL直接保存 → 継続サービス

依存関係分析

外部依存関係

- **Next.js 15**: App Router、動的ルート
- **React 19**: hooks、コンポーネント
- **Firebase SDK**: Firestore、Storage
- **OpenAI SDK**: GPT-4o、DALL-E 3
- **nanoid**: ユニークID生成

内部依存関係

- **Pages → API**: HTTP POST リクエスト

- **Pages → Components:** React コンポーネント階層
- **Components → Libraries:** ユーティリティ関数呼び出し
- **Libraries → Firebase:** SDK操作
- **Libraries → OpenAI:** API呼び出し

設計パターン

アーキテクチャパターン

- **レイヤードアーキテクチャ:** Pages → Components → Libraries → Services
- **ファサードパターン:** 複雑なFirebase操作を単純なインターフェースで提供
- **ストラテジーパターン:** 複数のAPI エンドポイントによる戦略選択
- **フォールバックパターン:** 段階的エラー回復

React パターン

- **カスタムhooks:** useState、useEffect、useRef の組み合わせ
- **Lifting State Up:** 親コンポーネントでの状態管理
- **Composition:** コンポーネント組み合わせによる機能実現

「コードの構造は詩の韻律のように美しく。関係性は心の糸のように繊細に、にや〜」 ✨