📊 ゆるVibe Pages データフロー詳細図

> *情報の流れと変換プロセスの包括的可視化*

データフロー概要

このデータフロー図は、ゆるVibe Pagesにおけるデータの生成・変換・永続化・表示の全プロセスを詳細に表現しています。ユーザー入力からAI生成、Firebase保存、画面表示まで、データがどのように流れ、どこで変換されるかを正確に示しています。

メインデータフロー

```
graph TD
    %% 入力層
    A[ユーザーテーマ入力] --> B[入力バリデーション]
    B --> C{有効性チェック}
    C -->|無効| D[エラーメッセージ]
    C -->|有効| E[API リクエスト生成]

    %% API処理層
    E --> F[POST /api/generate-storage]
    F --> G[並列AI処理開始]

    %% AI生成層
    G --> H[GPT-4o 詩生成]
    G --> I[DALL-E 画像生成]
```

```
%% データ変換層
H --> J[詩データ構造化]
I --> K[画像URL取得]

%% データ統合層
J --> L[生成結果統合]
K --> L
L --> M[PoemDocument作成]

%% 永続化層
M --> N{Storage保存試行}
N -->|成功| O[Firebase Storage URL]
N -->|失敗| P[DALL-E URL フォールバック]
O --> Q[Firestore保存]
P --> Q

%% レスポンス層
Q --> R[API レスポンス生成]
R --> S[クライアント返却]

%% 表示層
S --> T[ページ遷移]
T --> U[詩データ取得]
U --> V[画像読み込み]
V --> W{CORS対応読み込み}
W -->|getBlob成功| X[Object URL生成]
W -->|getBlob失敗| Y[直接URL使用]
W -->|完全失敗| Z[フォールバック背景]

%% 最終表示
X --> AA[詩ページ完全表示]
Y --> AA
Z --> AA

%% スタイル定義
classDef input fill:#e3f2fd
classDef processing fill:#fff3e0
classDef aiGeneration fill:#e8f5e8
classDef storage fill:#fce4ec
classDef display fill:#f3e5f5

class A,B,C,E input
```

```
    class F,G,J,K,L,M processing
    class H,I aiGeneration
    class N,O,P,Q,R storage
    class T,U,V,W,X,Y,Z,AA display
```

詳細データ変換プロセス

入力データ変換

```
graph LR
    A[ユーザー入力: string] --> B[trim処理]
    B --> C[長さ制限チェック]
    C --> D[文字種類バリデーション]
    D --> E[サニタイズ処理]
    E --> F[テーマオブジェクト作成]

    F --> G[{"{ theme: string }"}]
    G --> H[JSON文字列化]
    H --> I[HTTP POST Body]

    %% バリデーション詳細
    J[バリデーションルール] --> K[必須: 1文字以上]
    J --> L[最大: 100文字]
    J --> M[XSS対策: HTML エスケープ]

    classDef input fill:#e3f2fd
    classDef validation fill:#fff3e0
    classDef output fill:#e8f5e8

    class A,B,C,D,E input
    class J,K,L,M validation
    class F,G,H,I output
```

AI データ生成変換

```
sequenceDiagram
    participant C as クライアント
    participant A as API Endpoint
    participant G as GPT-4o
    participant D as DALL-E 3
    participant P as データ処理

    C->>A: { theme: "ざわざわした気分" }

    par GPT-4o 詩生成
        A->>G: プロンプト生成
        Note over G: "ざわざわした気分"という気持ちを<br/>表現する美し
        G-->>A: 生成詩テキスト
        Note over A: "ざわめきの中で<br/>ほんの少し<br/>風が鳴った"
    and DALL-E 画像生成
        A->>D: 英語プロンプト生成
        Note over D: A serene landscape that evokes<br/>the fee
        D-->>A: 画像URL
        Note over A: https://oaidalleapi...blob.core.windows.ne
    end

    A->>P: データ統合処理
    Note over P: PoemDocument構造作成
    P-->>A: 統合データオブジェクト

    A-->>C: 最終レスポンス
```

データモデル変換階層

```
graph TD
    %% 入力データモデル
    A[Input Model] --> B["{ theme: string }"]

    %% 中間データモデル
    C[Processing Models] --> D[GPT Request Model]
    C --> E[DALL-E Request Model]
    C --> F[Storage Upload Model]

    D --> G["{ model: 'gpt-4o', messages: [...], temperature: 0
```

```
    E --> H["{ model: 'dall-e-3', prompt: string, size: '1792x1
    F --> I["{ imageId: string, imageUrl: string, metadata: {..

    %% 出力データモデル
    J[Storage Models] --> K[PoemDocument]
    J --> L[StorageFile]

    K --> M["{ id, theme, phrase, imageUrl, imagePrompt, create
    L --> N["{ path: 'generated-images/timestamp-id.png', metad

    %% レスポンスデータモデル
    O[Response Models] --> P[API Success Response]
    O --> Q[API Error Response]

    P --> R["{ success: true, data: {...}, timing: {...} }"]
    Q --> S["{ success: false, error: string, details: string }

    classDef inputModel fill:#e3f2fd
    classDef processingModel fill:#fff3e0
    classDef storageModel fill:#e8f5e8
    classDef responseModel fill:#fce4ec

    class A,B inputModel
    class C,D,E,F,G,H,I processingModel
    class J,K,L,M,N storageModel
    class O,P,Q,R,S responseModel
```

Firebase データフロー詳細

Firestore データ操作

```
graph TD
    A[PoemDocument作成] --> B[nanoid生成]
    B --> C[タイムスタンプ追加]
    C --> D[データ構造検証]
    D --> E[Firestore保存]

    E --> F{保存成功？}
```

```
    F -->|成功| G[Document ID返却]
    F -->|失敗| H[エラーログ出力]

    %% 読み取りフロー
    I[詩ページリクエスト] --> J[Document ID抽出]
    J --> K[Firestore クエリ]
    K --> L{Document存在？}
    L -->|存在| M[データ返却]
    L -->|なし| N[404エラー]

    %% データ構造詳細
    O[Firestore Document] --> P[id: string - nanoid]
    O --> Q[theme: string - ユーザー入力]
    O --> R[phrase: string - GPT-4o生成]
    O --> S[imageUrl: string - Storage/DALL-E URL]
    O --> T[imagePrompt: string - DALL-E プロンプト]
    O --> U[createdAt: Timestamp - 作成日時]

    classDef operation fill:#e8f5e8
    classDef validation fill:#fff3e0
    classDef dataField fill:#f3e5f5

    class A,B,C,D,E,I,J,K operation
    class F,L validation
    class P,Q,R,S,T,U dataField
```

Storage データフロー

```
sequenceDiagram
    participant A as API
    participant S as Storage Service
    participant F as Firebase Storage
    participant D as DALL-E URL
    participant C as Client

    A->>S: uploadImageToStorage(imageId, dalleUrl)
    S->>D: fetch(dalleUrl)
    D-->>S: Image Blob

    S->>S: Blob変換・メタデータ付与
```

```
    Note over S: metadata: { theme, prompt, generatedAt }

    S->>F: uploadBytes(storageRef, blob, metadata)
    F-->>S: Upload Result

    alt Upload成功
        S->>F: getDownloadURL(ref)
        F-->>S: Storage URL
        S-->>A: { success: true, url: storageUrl }
    else Upload失敗
        S-->>A: { success: false, fallbackUrl: dalleUrl }
    end

    %% クライアント側読み込み
    C->>F: getBlob(storageRef)
    alt getBlob成功
        F-->>C: Blob Data
        C->>C: URL.createObjectURL(blob)
        Note over C: CORS回避成功
    else getBlob失敗
        C->>F: getDownloadURL(ref)
        F-->>C: Direct URL
        Note over C: CORS依存
    end
```

エラーハンドリング データフロー

```
graph TD
    A[エラー発生源] --> B{エラー種別判定}

    B -->|入力エラー| C[クライアント側バリデーション]
    B -->|API エラー| D[サーバー側処理]
    B -->|外部サービスエラー| E[外部API依存]
    B -->|データベースエラー| F[Firebase関連]

    %% クライアント側エラー
    C --> G[入力フィールド強調]
    C --> H[エラーメッセージ表示]
    C --> I[フォーカス設定]
```

```
    %% サーバー側エラー
    D --> J[構造化ログ出力]
    D --> K[エラーレスポンス生成]
    D --> L[適切なHTTPステータス]

    %% 外部サービスエラー
    E --> M{OpenAI API？}
    E --> N{Firebase？}
    M -->|レート制限| O[429エラー処理]
    M -->|認証失敗| P[401エラー処理]
    N -->|権限不足| Q[403エラー処理]
    N -->|接続失敗| R[ネットワークエラー処理]

    %% フォールバック処理
    S[フォールバック戦略] --> T[Storage失敗 → DALL-E URL]
    S --> U[画像読み込み失敗 → 背景フォールバック]
    S --> V[API完全失敗 → ダミーエンドポイント]

    classDef error fill:#ffebee
    classDef handling fill:#fff3e0
    classDef fallback fill:#e8f5e8

    class A,B,C,D,E,F error
    class G,H,I,J,K,L,M,N,O,P,Q,R handling
    class S,T,U,V fallback
```

パフォーマンス監視データ

タイミング データ収集

```
graph LR
    A[リクエスト開始] --> B[バリデーション時間]
    B --> C[GPT-4o 処理時間]
    B --> D[DALL-E 処理時間]
    C --> E[AI処理完了]
    D --> E
    E --> F[Storage保存時間]
```

```
    F --> G[Firestore保存時間]
    G --> H[レスポンス生成時間]
    H --> I[総処理時間]

    %% パフォーマンスログ
    J[パフォーマンスログ] --> K["{ total: 8750ms }"]
    J --> L["{ gpt: 3200ms }"]
    J --> M["{ dalle: 4500ms }"]
    J --> N["{ storage: 1050ms }"]
    J --> O["{ firestore: 250ms }"]

    %% クライアント側計測
    P[クライアント計測] --> Q[ページ読み込み時間]
    P --> R[画像読み込み時間]
    P --> S[アニメーション描画時間]
    P --> T[インタラクション応答時間]

    classDef timing fill:#e8f5e8
    classDef serverMetrics fill:#fff3e0
    classDef clientMetrics fill:#f3e5f5

    class A,B,C,D,E,F,G,H,I timing
    class J,K,L,M,N,O serverMetrics
    class P,Q,R,S,T clientMetrics
```

データセキュリティフロー

セキュアデータ処理

```
graph TD
    A[機密データ識別] --> B[API Key管理]
    A --> C[ユーザーデータ保護]
    A --> D[生成コンテンツ管理]

    B --> E[環境変数ストレージ]
    B --> F[サーバーサイド限定]
    B --> G[クライアント露出防止]
```

```
    C --> H[入力サニタイズ]
    C --> I[XSS対策]
    C --> J[データ最小化]

    D --> K[生成詩の公開性]
    D --> L[画像コンテンツ監視]
    D --> M[著作権考慮]

    %% データ暗号化フロー
    N[データ暗号化] --> O[HTTPS通信]
    N --> P[Firebase SSL]
    N --> Q[Vercel TLS]

    %% アクセス制御
    R[アクセス制御] --> S[公開詩の読み取り]
    R --> T[生成APIの使用]
    R --> U[管理機能の保護]

    classDef security fill:#ffebee
    classDef protection fill:#fff3e0
    classDef access fill:#e8f5e8

    class A,B,C,D security
    class E,F,G,H,I,J,K,L,M protection
    class N,O,P,Q,R,S,T,U access
```

将来のデータフロー拡張

リアルタイム機能データフロー

```
graph TD
    A[将来拡張: リアルタイム] --> B[WebSocket接続]
    A --> C[いいね機能]
    A --> D[コメント機能]
    A --> E[リアルタイム通知]

    B --> F[Socket.io統合]
    B --> G[リアルタイム詩表示]
```

```
    B --> H[協調生成機能]

    C --> I[Like Counter更新]
    C --> J[人気詩ランキング]
    C --> K[ユーザー好み学習]

    D --> L[コメントストリーム]
    D --> M[返信機能]
    D --> N[コメント通知]

    E --> O[新詩生成通知]
    E --> P[フォロワー通知]
    E --> Q[システム通知]

    classDef future fill:#f3e5f5
    classDef realtime fill:#e8f5e8
    classDef social fill:#fce4ec

    class A future
    class B,F,G,H realtime
    class C,D,E,I,J,K,L,M,N,O,P,Q social
```

> *「データの流れは川の調べのように。情報が美しく変換され、心に届く詩となる、にゃ〜」* ✨