実装計画: フォトアルバムオーガナイザー

ブランチ: `001-photo-album-organizer` | **日付**: 2025-11-18 | **仕様**: spec.md, requirements.md

概要

フォトアルバムオーガナイザーは、ローカルストレージ内の写真を撮影日時に基づいて自動的に日付別アルバムにグループ化し、ドラッグ&ドロップで並び替え、タイル状プレビュー表示、フルサイズ表示・ダウンロード機能を提供するシングルページアプリケーション。実装方法は Vanilla JavaScript + Vite + SQL.js で、GitHub Pages での完全クライアント側実行を実現し、外部サーバーへのアップロードは行わない。

技術コンテキスト

言語/バージョン: JavaScript (ES6+) + HTML5 + CSS3

主要依存関係:

- Vite 5.0.0（バンドラー・開発サーバー）
- sql.js 1.8.0（SQLiteのWebAssembly実装）
- Vitest 1.0.0（テストフレームワーク）
- terser 5.44.1（minify）

ストレージ:

- ローカルストレージ（metadata、DBstate）
- IndexedDB相当: SQL.js の localStorage 永続化（base64エンコード）
- File API（ユーザーが選択したファイルの読み込み）

テスト: Vitest（ユニット・統合・コントラクトテスト対応）

対象プラットフォーム:

- Chrome, Firefox, Safari, Edge（最新2版）
- デスクトップ・タブレット・モバイル対応

パフォーマンスゴール:

- UI 応答時間 ≤1秒（ドラッグ&ドロップ、写真読込、タイル表示）
- メモリ使用量 ≤200MB
- バンドルサイズ ≤300KB

制約:

- GitHub Pages 静的ホスティング（サーバーコンポーネントなし）
- 外部 API 依存なし
- IndexedDB 容量制限（通常 5-50MB、ユーザー設定で増加可能）

スケール/スコープ:

- 初期対応: 1 ユーザー・1000+ 写真・5+ アルバム
- 拡張予定: マルチデバイス同期（将来）

憲法遵守チェック

ゲート: Phase 0 リサーチ前に合格する必要があります。Phase 1 設計後に再評価します。

原則 I - テスト駆動開発:

- ✅ すべてのユーザーストーリー（US1-US3）は実装前に受け入れシナリオを定義
- ✅ テストファーストアプローチで実装（Red-Green-Refactor）
- ✅ 単体テスト・統合テスト・コントラクトテスト配置予定
- **検証方法**: `npm run test` で全テスト実行、coverage > 80%

原則 II - セキュリティ優先:

- ✅ メタデータ（撮影日時・ファイル名）は SQLite DB に保存、暗号化対象
- ✅ 画像ファイルはローカルのみ（外部アップロード禁止）
- ✅ ファイル型・メタデータ形式を厳密検証
- ✅ 入出力エスケープ、XSS 対策を実装
- **検証方法**: SEC-001~004 の手動セキュリティテスト、OWASP チェックリスト

原則 III - パフォーマンス定量化:

- ✅ PERF-001: UI 応答 ≤1秒（Lighthouse スコア 90+）
- ✅ PERF-002: 大規模アルバム 1000+ 写真対応（仮想スクロール）
- ✅ PERF-003: メモリ ≤200MB（DevTools Memory Profiler で検証）
- ✅ PERF-004: バンドル ≤300KB（実装: 10.26KB JS + 5.05KB CSS = 63KB）
- **検証方法**: `npm run build` + Lighthouse + Chrome DevTools

ゲート結果: 合格 ✅ （すべての原則が実装計画に明記）

プロジェクト構造

ドキュメント

```
specs/001-photo-album-organizer/
├── spec.md                     # ユーザーストーリー・要件
├── requirements.md             # 技術背景・憲法チェック
├── plan.md                     # このファイル（実装計画）
├── research.md                 # Phase 0 技術リサーチ結果（予定）
├── data-model.md               # Phase 1 データモデル詳細（予定）
├── quickstart.md               # Phase 1 クイックスタートガイド（予定）
├── contracts/                  # Phase 1 API/ストレージコントラクト（予定）
│   ├── photo-storage.contract.md
│   ├── album-storage.contract.md
│   └── order-persistence.contract.md
└── tasks.md                    # Phase 2 実装タスク分解（予定）
```

ソースコード

```
src/
├── index.html                  # メインエントリ（UIテンプレート）
├── main.js                     # アプリケーションロジック（~400行）
│                                # ├ ファイル読込処理
│                                # ├ メタデータ抽出
│                                # ├ DB 保存
│                                # ├ グループ化
│                                # ├ UI レンダリング
│                                # └ ドラッグ&ドロップハンドラ
├── components/
│   ├── MainPage.js             # メインページ UI（予定）
│   ├── AlbumView.js            # アルバムビュー（予定）
│   ├── TileGrid.js             # タイル状グリッド（予定）
│   └── FullsizeModal.js        # フルサイズモーダル（予定）
├── services/
│   ├── DatabaseService.js      # SQL.js DB 操作（CRUD）
│   ├── AlbumService.js         # アルバムグループ化・並び替え
│   ├── StorageService.js       # ローカルストレージ管理（予定）
│   └── MetadataService.js      # メタデータ抽出（予定）
├── utils/
│   ├── dateUtils.js            # YYYY-MM-DD フォーマット、日付比較
│   ├── fileValidation.js       # ファイル型検証（JPEG/PNG/WebP）
│   └── logger.js               # ログシステム（Level: INFO/WARN/ERROR）
└── styles/
    ├── main.css                # グローバルスタイル（リセット・base）
    ├── components.css          # UI コンポーネント（button, card, modal）
    └── responsive.css          # レスポンシブ（breakpoints: 1200px/768px/480px）

tests/
├── unit/
│   ├── dateUtils.test.js       # 日付処理テスト
│   ├── fileValidation.test.js  # ファイル検証テスト
│   ├── AlbumService.test.js    # グループ化ロジックテスト
│   └── logger.test.js          # ログシステムテスト
├── integration/
│   ├── PhotoUpload.test.js     # ファイル読込 → DB 保存
│   ├── AlbumGrouping.test.js   # メタデータ抽出 → グループ化
│   ├── DragDrop.test.js        # アルバム並び替え → 永続化
│   └── Download.test.js        # フルサイズ表示 → ダウンロード
└── contract/
    ├── DatabaseService.contract.test.js    # DB I/F 検証
    ├── AlbumService.contract.test.js       # グループ化 I/F 検証
    └── FileAPI.contract.test.js            # ファイルAPI I/F 検証

public/
├── index.html                  # GitHub Pages 静的ファイル
└── favicon.ico

package.json                     # 依存関係定義
├── vite.config.js                 # Vite 設定

**設定ファイルの目的**:

vite.config.js                   # Vite 設定（base: '/photo-album-organizer/'）
package.json                     # 依存関係・スクリプト定義

**重要ポイント**:

- GitHub Pages での `/photo-album-organizer/` パス対応のため Vite の base 設定
```

構造決定根拠:

- GitHub Pages での `/photo-album-organizer/` パス対応のため Vite の base 設定
- ドキュメント（specs/）と実装（src/）の完全分離
- サービス層で業務ロジックを集約し、テスト可能性向上
- utils 層で再利用可能なユーティリティを個別ファイル化

複雑性トラッキング

> *Fill ONLY if Constitution Check has violations that must be justified*

現在、憲法違反なし。すべての原則が実装計画に組み込まれています。

違反	必要な理由	より単純な代替案が不適切な理由
（なし）		

Phase 0: 技術リサーチ（予定）

リサーチ対象

1. **SQLite.js (sql.js) vs IndexedDB トレードオフ**
   - 決定: sql.js（関連型DB の SQL QL で複雑クエリ対応可能）
   - 根拠: アルバム並び替え・メタデータ検索で SQL が便利
   - 検証: プロトタイプ実装で パフォーマンステスト
2. **Vite 最適化戦略**
   - Code Splitting: コンポーネント遅延ロード
   - Tree-Shaking: 未使用コード削除
   - Minify: Terser で JS/CSS 圧縮
   - 目標: バンドル < 300KB（現在: 63KB で達成）
3. **大規模ファイル処理戦略**
   - 仮想スクロール: 1000+ 写真の効率的レンダリング
   - 遅延ロード: スクロール時のサムネイル動的読込
   - メモリプロファイリング: Chrome DevTools で監視
4. **ブラウザ互換性検証**
   - 対象: Chrome/Firefox/Safari/Edge 最新2版
   - File API, IndexedDB, CSS Grid 互換性確認
   - 手動テスト: 各ブラウザでスモークテスト実施

出力: research.md (Phase 0)

最終決定・根拠・代替案・検証方法をまとめたドキュメント

Phase 1: 設計・コントラクト（予定）

1a. データモデル定義（data-model.md）

エンティティ:

- **Photo**
  - id: UUID
  - file_name: string (max 255)
  - file_size: number (bytes)
  - photo_date: ISO 8601 (YYYY-MM-DD)
  - photo_time: ISO 8601 (HH:MM:SS)
  - data_uri: string (base64 encoded image)
  - created_at: ISO 8601
  - checksum: string (MD5 of file content, for deduplication)
- **Album**
  - id: UUID
  - album_date: ISO 8601 (YYYY-MM-DD, unique)
  - display_order: number (for ranking)
  - created_at: ISO 8601
  - photo_count: number (denormalized for UI)
- **AlbumOrder** (or stored in Album.display_order)
  - album_id: UUID
  - display_order: number

インデックス:

- photos(photo_date): クエリ最適化
- photos(created_at): タイムスタンプ検索
- albums(display_order): 表示順序ソート
- albums(album_date): 日付検索

制約:

- albums.album_date: UNIQUE
- photos.photo_date: NOT NULL
- Foreign key: photos(album_id) → albums(id)

1b. ストレージコントラクト（contracts/）

- **photo-storage.contract.md**
  - addPhoto(file: File) → Promise `<Photo>`
  - getPhotosByDate(date: string) → Promise<Photo[]>
  - deletePhoto(id: UUID) → Promise `<boolean>`
  - updatePhotoOrder(photos: Photo[]) → Promise `<void>`
- **album-storage.contract.md**
  - createAlbum(date: string) → Promise `<Album>`
  - getAlbumsByOrder() → Promise<Album[]>
  - updateAlbumOrder(albums: Album[]) → Promise `<void>`
  - deleteAlbum(id: UUID) → Promise `<boolean>`
- **order-persistence.contract.md**
  - saveAlbumOrder(order: AlbumOrder[]) → Promise `<void>`
  - loadAlbumOrder() → Promise<AlbumOrder[]>

1c. クイックスタート（quickstart.md）

開発者向けセットアップガイド:

- 環境構築（Node.js, npm）
- リポジトリクローン・インストール
- `npm run dev` での開発サーバー起動
- テスト実行（`npm run test`）
- ビルド・デプロイ（`npm run build` → GitHub Pages）
- トラブルシューティング

Phase 2: タスク分解（予定）

/speckit.tasks コマンドで生成予定

tasks.md では以下を分解:

ユーザーストーリー US1: 日付別アルバム整理

- Task 1.1: メタデータ抽出テスト（TDD）
- Task 1.2: DatabaseService.addPhoto() 実装
- Task 1.3: AlbumService.groupPhotosByDate() 実装
- Task 1.4: セキュリティテスト（SEC-001）
- Task 1.5: パフォーマンステスト（PERF-001）

ユーザーストーリー US2: タイル状プレビュー

- Task 2.1: TileGrid コンポーネント実装
- Task 2.2: 仮想スクロール実装
- Task 2.3: CSS Grid レスポンシブ対応
- Task 2.4: パフォーマンステスト（PERF-002）

ユーザーストーリー US3: フルサイズ表示・DL

- Task 3.1: FullsizeModal コンポーネント実装
- Task 3.2: ダウンロード機能実装
- Task 3.3: メタデータ保持テスト

実装状況（現在）

既完了項目 ✅

- ✅ **セットアップ**: Node.js, npm, Vite 環境構築
- ✅ **package.json**: 依存関係定義・ロック
- ✅ **vite.config.js**: GitHub Pages base path (`/photo-album-organizer/`) 設定
- ✅ **index.html**: UIテンプレート（3ビュー: メインページ・アルバムビュー・フルサイズモーダル）
- ✅ **main.js**: 約400行のコアアプリケーションロジック
  - ファイル読込・バリデーション
  - メタデータ抽出・DB 保存
  - アルバムグループ化・レンダリング
  - ドラッグ&ドロップハンドラ
  - モーダル表示・ダウンロード
- ✅ **DatabaseService.js**: SQL.js による DB CRUD
  - initDatabase()
  - initializeSchema()
  - addPhoto(), getPhotosByDate(), getAllAlbumDates()
  - createOrUpdateAlbum(), saveDatabase()
- ✅ **AlbumService.js**: ビジネスロジック
  - groupPhotosByDate()
  - getAllAlbums()
  - updateAlbumOrder()
- ✅ **utilities**: dateUtils, fileValidation, logger
- ✅ **styles**: main.css, components.css, responsive.css
- ✅ **README.md**: ユーザー向けドキュメント
- ✅ **起動スクリプト**: start.ps1, start.sh
- ✅ **npm install**: 102パッケージインストール完了
- ✅ **npm run build**: バンドル生成（63KB）
- ✅ **Git commits**: 4コミット完了

予定項目 ⏳

- ⏳ **research.md**: Phase 0 技術リサーチ結果
- ⏳ **data-model.md**: データモデル詳細
- ⏳ **contracts/**: ストレージコントラクト
- ⏳ **quickstart.md**: 開発者ガイド
- ⏳ **テスト実装**:
  - ユニットテスト（dateUtils, fileValidation, etc.）
  - 統合テスト（PhotoUpload, AlbumGrouping, DragDrop）
  - コントラクトテスト（DatabaseService, AlbumService）
- ⏳ **tasks.md**: Phase 2 タスク分解（/speckit.tasks コマンド）

ビルド・デプロイメント

開発

```
npm run dev              # localhost:5173 自動オープン
npm run test            # Vitest 実行
npm run test:ui         # Vitest UI 起動
```

本番

```
npm run build           # dist/ 生成（63KB）
npm run preview         # ビルド結果プレビュー
# dist/ を GitHub Pages にデプロイ
```

パフォーマンス検証

```
# Lighthouse スコア確認
npm run build
npm run preview
# → ブラウザで http://localhost:4173 を Lighthouse で検査

# メモリプロファイリング
# Chrome DevTools → Memory → Heap Snapshot
```

次のステップ

1. **Phase 0 実行**: `/speckit.plan` で research.md 生成
2. **Phase 1 実行**: data-model.md, contracts/ 生成
3. **agent context 更新**: copilot 向けコンテキスト更新
4. **Phase 2 実行**: `/speckit.tasks` で tasks.md 生成
5. **テスト実装**: TDD に従い、各タスクでテスト先行
6. **コード生成**: 実装ブランチでコード生成・検証