

データモデル仕様: Todo App

作成日: 2025-11-13
プロジェクト: Todo App - template-no-delete.tsx ベースアプリケーション
目的: エンティティ定義、フィールド、関係性、バリデーションルール、状態遷移を文書化

概要

Todo Appは3つの主要エンティティで構成されます:

- TodoItem: 個別のタスクを表現
- UserPage: ユーザーごとのTodoページを表現
- StorageKey: LocalStorageのキー命名規則

LocalStorageベースのクライアントサイドアプリケーションのため、リレーションナルデータベースは使用せず、JSON形式でデータを永続化します。

エンティティ定義

1. TodoItem (タスクアイテム)

用途: 個別のタスクを表現するコアエンティティ

TypeScript型定義

```
interface TodoItem {  
  id: number;  
  text: string;  
  completed: boolean;  
  createdAt: string; // ISO 8601形式  
}
```

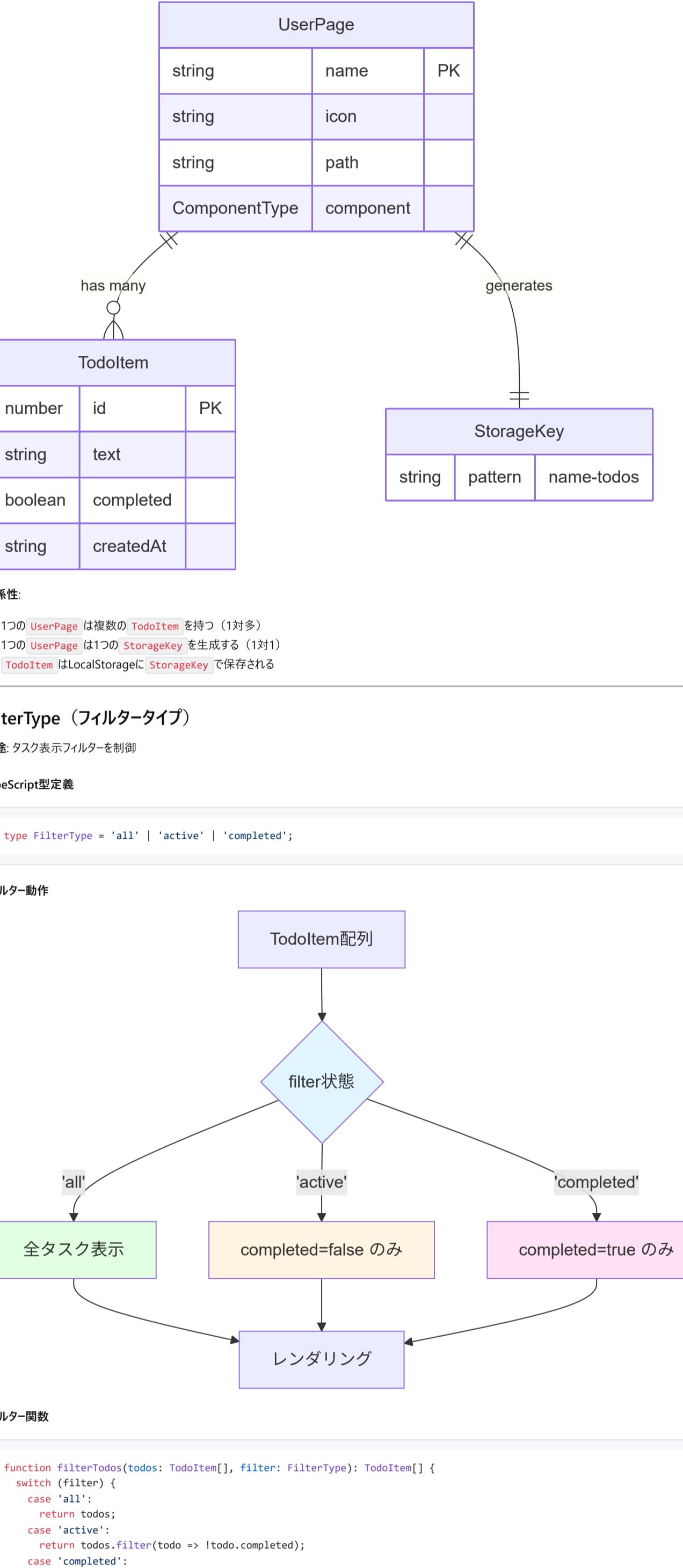
フィールド詳細

フィールド	型	必須	説明	制約
id	number	✓	一意識別子	Date.now()によるタイムスタンプ、正の整数
text	string	✓	タスク内容	1~50文字、trim後空文字列禁止
completed	boolean	✓	完了状態	true (完了) または false (未完了)
createdAt	string	✓	作成日時	ISO 8601形式 (例: "2025-11-13T10:30:00.000Z")

バリデーションルール

```
function validateTodoItem(item: unknown): item is TodoItem {  
  if (typeof item !== 'object' || item === null) return false;  
  
  const todo = item as Record<string, unknown>;  
  
  // id: 正の整数  
  if (typeof todo.id !== 'number' || !Number.isInteger(todo.id) || todo.id <= 0) {  
    return false;  
  }  
  
  // text: 1~50文字  
  if (typeof todo.text !== 'string' || todo.text.trim().length === 0) {  
    return false;  
  }  
  
  if (!todo.completed) {  
    return false;  
  }  
  
  // createdAt: ISO 8601形式  
  if (typeof todo.createdAt !== 'string') {  
    return false;  
  }  
  
  const date = new Date(todo.createdAt);  
  if (!isNaN(date.getTime())) {  
    return false;  
  }  
  
  return true;  
}
```

状態遷移



永続化形式

LocalStorageには以下のJSON配列として保存:

```
[  
  {  
    "id": 1699876543210,  
    "text": "Reactの学習",  
    "completed": false,  
    "createdAt": "2025-11-13T01:22:23.210Z"  
  },  
  {  
    "id": 1699876600000,  
    "text": "TypeScriptの学習",  
    "completed": true,  
    "createdAt": "2025-11-13T01:23:20.000Z"  
}
```

2. UserPage (ユーザーページ)

用途: ユーザーごとのTodoページ設定を表現

TypeScript型定義

```
interface UserPage {  
  name: string;  
  icon: string;  
  path: string;  
  component: React.ComponentType;  
}
```

フィールド詳細

フィールド	型	必須	説明	制約
name	string	✓	ページ名 (日本語対応)	1~50文字、重複禁止、trim後空文字列禁止
icon	string	✓	絵文字アイコン	1文字の絵文字、デフォルト "■"
path	string	✓	ルーティングパス	/開始、英数字とハイフン、重複禁止

バリデーションルール

```
function validateUserPage(page: unknown): page is UserPage {  
  if (typeof page !== 'object' || page === null) return false;  
  
  const p = page as Record<string, unknown>;  
  
  // name: 1~50文字  
  if (typeof p.name !== 'string' || p.name.trim().length === 0) {  
    return false;  
  }  
  if (p.name.length > 50) {  
    return false;  
  }  
  
  // icon: 1文字の絵文字  
  if (typeof p.icon !== 'string' || p.icon.length === 0) {  
    return false;  
  }  
  
  // path: /開始、有効なパス  
  if (typeof p.path !== 'string' || !p.path.startsWith('/')) {  
    return false;  
  }  
  if (!/^(a-zA-Z-9-.)+$/i.test(p.path)) {  
    return false;  
  }  
  
  return true;  
}
```

設定ファイル形式

src/config/userPages.ts に以下の形式で定義:

```
import { DynamicTodoPage } from './pages/DynamicTodoPage';  
  
export const userPages: UserPage[] = [  
  {  
    name: '浜崎秀寿',  
    icon: '■',  
    path: '/hamasaki-todo',  
    component: DynamicTodoPage  
  },  
  {  
    name: 'TestUser',  
    icon: '■',  
    path: '/testuser-todo',  
    component: DynamicTodoPage  
  }  
];
```

ページライフサイクル



データ整合性ルール

1. ID一意性保証

用途: LocalStorageのキー命名規則を標準化

命名パターン

```
type StorageKeyPattern = `${string}-todos`;  
  
// 例  
const key1: StorageKeyPattern = '浜崎秀寿-todos';  
const key2: StorageKeyPattern = 'testuser-todos';  
const key3: StorageKeyPattern = 'page-info'; // メタデータ用
```

キー生成関数

```
function genTodoKey(pageName: string): string {  
  // エスケープ処理 (特殊文字削除)  
  const safeName = pageName.trim();  
  return `${safeName}-todos`;  
}  
  
function getPageInfoKey(): string {  
  return 'page-info';  
}
```

LocalStorage構造

```
{  
  "浜崎秀寿-todos": "[{"id": 1699876543210, "text": "Reactの学習", "completed": false, "createdAt": "2025-11-13T01:22:23.210Z"}, {"id": 1699876600000, "text": "TypeScriptの学習", "completed": true, "createdAt": "2025-11-13T01:23:20.000Z"}]",  
  "testuser-todos": "[{"id": 1699876600000, "text": "TypeScriptの学習", "completed": true, "createdAt": "2025-11-13T01:23:20.000Z"}]",  
  "page-info": "{"version": "1.0", "lastUpdated": "2025-11-13T10:00:00.000Z"}"
}
```

クリーンアップルール

```
function cleanupOldData(pageName: string): void {  
  const key = getTodoKey(pageName);  
  
  // LocalStorageから一時削除  
  localStorage.removeItem(key);  
  
  // userPages.tsから手動削除 (サーバー再起動必要)  
  // ※ 自動化は将来的な課題  
}
```

エンティティ関係図

関係性

- 1つのUserPage は複数のTodoItem を持つ (1対多)
- 1つのUserPage は1つのStorageKey を生成する (1対1)
- TodoItem はLocalStorage のStorageKey で保存される

FilterType (フィルタータイプ)

用途: タスク表示フィルタを制御

TypeScript型定義

```
type FilterType = 'all' | 'active' | 'completed';
```

フィルタ動作

フィルタ関数

```
function filterTodos(todos: TodoItem[], filter: FilterType): TodoItem[] {  
  switch (filter) {  
    case 'all':  
      return todos;  
    case 'active':  
      return todos.filter(todo => !todo.completed);  
    case 'completed':  
      return todos.filter(todo => todo.completed);  
    default:  
      return todos;  
  }
}
```

データ整合性ルール

1. ID一意性保証

用途: LocalStorageのキー命名規則を標準化

命名パターン

```
type StorageKeyPattern = `${string}-todos`;  
  
// 例  
const key1: StorageKeyPattern = '浜崎秀寿-todos';  
const key2: StorageKeyPattern = 'testuser-todos';  
const key3: StorageKeyPattern = 'page-info'; // メタデータ用
```

設定ファイル形式

```
import { DynamicTodoPage } from './pages/DynamicTodoPage';  
  
export const userPages: UserPage[] = [  
  {  
    name: '浜崎秀寿',  
    icon: '■',  
    path: '/hamasaki-todo',  
    component: DynamicTodoPage  
  },  
  {  
    name: 'TestUser',  
    icon: '■',  
    path: '/testuser-todo',  
    component: DynamicTodoPage  
  }  
];
```

ページライフサイクル

データ整合性ルール

2. LocalStorage同期

用途: ページ間でデータ同期

```
// ページ間でIDが重複する場合、1を  
useEffect(() => {  
  const key = getTodoKey(pageName);  
  localStorage.setItem(key, JSON.stringify(todos));  
  
  const saved = localStorage.getItem(key);  
  if (saved) {  
    const parsed = JSON.parse(saved);  
    if (Array.isArray(parsed)) {  
      setTodos(parsed);  
    }  
  }
}, [pageName]);
```

3. ページ名重複チェック

```
function isPageNameUnique(name: string, existingPages: UserPage[]): boolean {  
  return !existingPages.some(page => page.name === name);  
}
```

パフォーマンス考慮事項

1. LocalStorage容量管理

```
function getSizeLocalStorage(): number {  
  let total = 0;  
  for (const key in localStorage) {  
    if (localStorage.hasOwnProperty(key)) {  
      total += localStorage[key].length + key.length;  
    }
}
```

2. 大量タスク対策

```
// 大量タスク表示で1秒も遅延しないよう実装  
useEffect(() => {  
  const key = getTodoKey(pageName);  
  localStorage.setItem(key, JSON.stringify(todos));  
  
  const saved = localStorage.getItem(key);  
  if (saved) {  
    const parsed = JSON.parse(saved);  
    if (Array.isArray(parsed)) {  
      setTodos(parsed);  
    }  
  }
}, [pageName]);
```

セキュリティ考慮事項

1. XSS対策