kintone UI Designer - 実装仕様書

プロジェクト概要

kintone JavaScriptカスタマイズ用のビジュアルUIデザインツールを開発してください。プログラミング知識がないユーザーでも、ドラッグ&ドロップでUIコンポーネントを配置し、kintone用のJavaScriptコードを自動生成できるツールです。

技術スタック

必須技術

- **フロントエンド**: React 18+ with TypeScript
- **スタイリング**: Tailwind CSS
- **状態管理**: Zustand
- **ドラッグ&ドロップ**: @dnd-kit/sortable
- **コードフォーマット**: Prettier API
- **ビルドツール**: Vite

プロジェクト構造

```
kintone-ui-designer/
├── src/
│   ├── components/
│   │   ├── Editor/
│   │   │   ├── Canvas.tsx
│   │   │   ├── ComponentPalette.tsx
│   │   │   ├── PropertyPanel.tsx
│   │   │   └── PreviewMode.tsx
│   │   ├── UIComponents/
│   │   │   ├── Modal/
│   │   │   ├── Button/
│   │   │   ├── Form/
│   │   │   └── Layout/
│   │   └── CodeGenerator/
│   │       ├── JavaScriptGenerator.tsx
│   │       └── CSSGenerator.tsx
│   ├── store/
│   │   ├── editorStore.ts
│   │   └── componentStore.ts
│   ├── types/
│   │   ├── component.types.ts
│   │   └── kintone.types.ts
│   ├── utils/
│   │   ├── codeGeneration.ts
│   │   └── kintoneHelpers.ts
│   └── templates/
│       └── componentTemplates.ts
└── public/
```

実装する機能

1. エディター基本機能

Canvas コンポーネント

typescript

```
// 実装すべき機能:
- ドラッグ&ドロップによるコンポーネント配置
- グリッドスナップ（8px単位）
- 選択したコンポーネントのハイライト表示
- 右クリックコンテキストメニュー（複製、削除、整列）
- Undo/Redo機能（Ctrl+Z/Ctrl+Y）
- コンポーネントのリサイズハンドル
```

ComponentPalette

typescript

```typescript
// 実装するコンポーネント一覧:
const components = {
  modal: {
    name: 'モーダル',
    icon: 'Modal',
    defaultProps: {
      width: 600,
      height: 400,
      title: '新規モーダル',
      showCloseButton: true,
      overlay: true,
      animation: 'fade'
    }
  },
  button: {
    name: 'ボタン',
    icon: 'Button',
    variants: ['primary', 'secondary', 'danger', 'success'],
    sizes: ['small', 'medium', 'large']
  },
  form: {
    input: { type: 'text', label: 'テキスト入力' },
    select: { type: 'select', options: [] },
    checkbox: { type: 'checkbox', label: 'チェックボックス' },
    radio: { type: 'radio', options: [] },
    textarea: { type: 'textarea', rows: 4 }
  },
  layout: {
    container: { padding: 16, background: '#ffffff' },
    grid: { columns: 12, gap: 16 },
    tabs: { tabs: ['Tab1', 'Tab2'] },
    accordion: { panels: [] }
  }
}
```

2. PropertyPanel の実装

typescript

```
// プロパティパネルの構造
interface PropertyPanelProps {
  selectedComponent: Component | null
  onUpdate: (id: string, updates: Partial<Component>) => void
}

// 実装する設定項目:
- スタイルタブ:
  - 背景色（カラーピッカー）
  - ボーダー（幅、色、スタイル）
  - 影（box-shadow設定）
  - サイズ（width, height, padding, margin）
  - フォント（family, size, weight, color）

- 動作タブ:
  - kintoneイベント選択（ドロップダウン）
  - カスタムイベント設定
  - 表示条件設定
  - バリデーションルール

- コンテンツタブ:
  - テキスト編集
  - アイコン選択
  - 画像アップロード（Base64変換）
```

3. コード生成機能

JavaScriptGenerator

```
javascript
```

```
// 生成するコードの例:
(function() {
  'use strict';

  // kintone イベントハンドラー
  kintone.events.on(['app.record.detail.show'], function(event) {
    // UIコンポーネント初期化
    const uiComponents = new KintoneUIComponents();

    // モーダルの作成
    const modal1 = uiComponents.createModal({
      id: 'modal_001',
      title: 'データ確認',
      width: 600,
      height: 400,
      content: '<div>...</div>',
      buttons: [
        { text: '確認', action: 'confirm', style: 'primary' },
        { text: 'キャンセル', action: 'cancel', style: 'secondary' }
      ]
    });

    // ボタンの追加
    const button1 = uiComponents.createButton({
      id: 'btn_001',
      text: 'モーダルを開く',
      style: 'primary',
      size: 'medium',
      onClick: () => modal1.show()
    });

    // DOM に追加
    kintone.app.record.getHeaderMenuSpaceElement().appendChild(button1.element);

    return event;
  });

  // ヘルパークラス
  class KintoneUIComponents {
    createModal(config) { /* 実装 */ }
    createButton(config) { /* 実装 */ }
    // 他のコンポーネント作成メソッド
  }
})();
```

CSSGenerator

```css
/* 生成するCSSの例 */
.kui-modal {
  --kui-modal-bg: #ffffff;
  --kui-modal-shadow: 0 10px 25px rgba(0, 0, 0, 0.2);

  position: fixed;
  background: var(--kui-modal-bg);
  box-shadow: var(--kui-modal-shadow);
  border-radius: 8px;
  z-index: 10000;
  animation: kui-fade-in 0.3s ease-out;
}

.kui-button {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', sans-serif;
  border: none;
  cursor: pointer;
  transition: all 0.2s ease;
}

.kui-button--primary {
  background-color: #3498db;
  color: white;
}

.kui-button--primary:hover {
  background-color: #2980b9;
  transform: translateY(-1px);
}

@keyframes kui-fade-in {
  from { opacity: 0; transform: scale(0.95); }
  to { opacity: 1; transform: scale(1); }
}
```

4. 状態管理（Zustand Store）

```typescript
```

```typescript
// editorStore.ts
interface EditorState {
  components: Component[]
  selectedComponentId: string | null
  canvasSize: { width: number; height: number }
  zoom: number
  gridEnabled: boolean

  // Actions
  addComponent: (component: Component) => void
  updateComponent: (id: string, updates: Partial<Component>) => void
  deleteComponent: (id: string) => void
  selectComponent: (id: string | null) => void
  duplicateComponent: (id: string) => void
  reorderComponents: (newOrder: string[]) => void

  // History
  history: HistoryState[]
  historyIndex: number
  undo: () => void
  redo: () => void
}

// componentStore.ts
interface ComponentState {
  templates: ComponentTemplate[]
  customComponents: CustomComponent[]
  recentlyUsed: string[]

  saveAsTemplate: (component: Component, name: string) => void
  loadTemplate: (templateId: string) => Component
}
```

5. 追加機能の実装

テンプレート機能

typescript

```
// 事前定義テンプレート
const templates = {
  confirmDialog: {
    name: '確認ダイアログ',
    description: '削除や更新時の確認用',
    components: [/* モーダル + ボタン構成 */]
  },
  formDialog: {
    name: 'フォーム入力ダイアログ',
    description: 'データ入力用のフォーム',
    components: [/* モーダル + フォーム要素 */]
  },
  notification: {
    name: '通知ポップアップ',
    description: '成功・エラー通知用',
    components: [/* ポップアップ構成 */]
  }
}
```

プレビュー機能

typescript

```
// PreviewMode.tsx
- iframe内でのリアルタイムプレビュー
- kintone風のUIテーマ適用
- インタラクション動作確認
- レスポンシブ表示確認（PC/タブレット/スマホ）
```

エクスポート機能

typescript

```typescript
interface ExportOptions {
  format: 'single' | 'separate' // 単一ファイル or JS/CSS分離
  minify: boolean
  includeComments: boolean
  kintoneVersion: string
  targetAppId?: string
}

// エクスポート時に生成するファイル:
// 1. custom.js - メインのJavaScriptファイル
// 2. custom.css - スタイルシート
// 3. config.json - 設定情報（再インポート用）
// 4. README.md - 使用方法の説明
```

6. kintone 固有の考慮事項

typescript

```typescript
// kintone イベント一覧
const kintoneEvents = {
  record: {
    create: ['app.record.create.show', 'app.record.create.submit'],
    edit: ['app.record.edit.show', 'app.record.edit.submit'],
    detail: ['app.record.detail.show'],
    index: ['app.record.index.show']
  },
  mobile: {
    // モバイル用イベント
  }
}

// kintone フィールドタイプ
const fieldTypes = [
  'SINGLE_LINE_TEXT',
  'MULTI_LINE_TEXT',
  'NUMBER',
  'CALC',
  'RADIO_BUTTON',
  'CHECK_BOX',
  'MULTI_SELECT',
  'DROP_DOWN',
  'DATE',
  'DATETIME',
  'ATTACHMENT',
  'LINK',
  'USER_SELECT'
]

// DOM 挿入位置
const insertPositions = [
  'HeaderMenuSpace',
  'HeaderSpace',
  'RecordSpace'
]
```

7. エラーハンドリングとバリデーション

typescript

```
// コンポーネント配置時のバリデーション
- 重複IDチェック
- 親子関係の妥当性確認
- サイズ制限チェック

// コード生成時のバリデーション
- kintone API互換性チェック
- 必須プロパティの確認
- イベント名の妥当性確認

// エラー表示
- トースト通知でのエラー表示
- 詳細なエラーログの保持
- 修正提案の表示
```

8. パフォーマンス最適化

```typescript
// 実装すべき最適化:
- React.memo による不要な再レンダリング防止
- useMemo/useCallback の適切な使用
- 仮想スクロール（多数のコンポーネント時）
- デバウンス処理（プロパティ変更時）
- Web Worker でのコード生成（大規模プロジェクト時）
```

9. テスト要件

```typescript
// 単体テスト（Vitest）
- コンポーネントの描画テスト
- ストアの動作テスト
- コード生成の正確性テスト

// 統合テスト
- ドラッグ&ドロップ操作
- コード生成からkintoneでの動作確認
- エクスポート/インポート機能

// E2Eテスト（Playwright）
- 完全なワークフローテスト
- ブラウザ互換性テスト
```

10. 初期実装の優先順位

1. **Phase 1 (MVP)**:
   - 基本的なドラッグ&ドロップエディター
   - モーダルとボタンコンポーネント
   - 基本的なコード生成
2. **Phase 2**:
   - 全UIコンポーネントの実装
   - プロパティパネルの完全実装
   - テンプレート機能
3. **Phase 3**:
   - 高度なスタイリング機能
   - プレビューモード
   - エクスポート/インポート機能

開発開始時の指示

このツールを開発する際は、まず以下の順序で実装してください：

1. Vite + React + TypeScript のプロジェクトセットアップ
2. 基本的なレイアウト（3ペインレイアウト）の実装
3. Zustand ストアの基本構造実装
4. ドラッグ&ドロップ機能の実装（@dnd-kit）
5. 最初のコンポーネント（ボタン）の実装
6. 基本的なコード生成機能
7. 順次機能を追加

各機能は小さく分割して実装し、動作確認しながら進めてください。