

catから始めるUI MCP

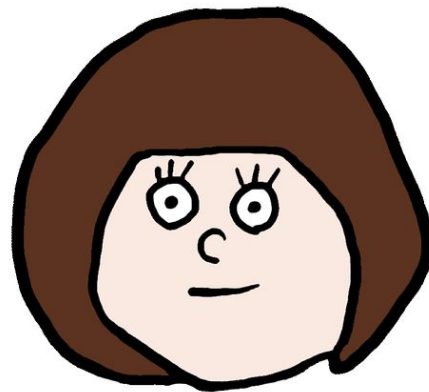
2025/05/12 THE UI EXPERTS - AIとMCPの現在地

@yoshiko

_KNOWLEDGE WORK

よしこ [@yoshiko_pg](#)

株式会社ナレッジワークのフロントエンドエンジニア
AIが好きでZennでAI関連の記事書いてます



<https://zenn.dev/yoshiko>

ナレッジワーク UI MCP の段階的実装記録

やってみたいけどどう始めたらいいかわからない…という方へ
思ったより雑に始められますよ！というのを伝えたい

- 前提
- 実装
 - Step1: catで最小限実装
 - Step2: Props型の提供
 - Step3: Storyの提供
- 知見
- 今後やりたいこと

- **前提**
- **実装**
 - Step1: catで最小限実装
 - Step2: Props型の提供
 - Step3: Storyの提供
- **知見**
- **今後やりたいこと**

- 元々社内でUI Componentがnpm packageとして整備されていた
- storybookのカバー率も高かった
 - 各Componentに最低ひとはStoryがある状態

- FastMCPでの基盤があった
- npm install & nodeで実行するshをエントリポイントに
 - mcp.jsonにそのshを登録

```
import { FastMCP } from 'fastmcp'

const server = new FastMCP({
  name: 'ナレッジワークMCPサーバー',
  version: '1.0.0',
})

server.addTool({
  // 次で解説
})

server.start({ transportType: 'stdio' })
```

- addToolでツール登録

利用フローの想定

- 引数のmodeで一覧/詳細
 - まず一覧を取得
 - デザインと見比べて
 - 使うComponentを判断
 - 使うものの詳細を取得

```
server.addTool({
  name: 'UI Component Catalog',
  description: '標準UIコンポーネントの情報を提供',
  // inputスキーマにzodが使える
  parameters: z.object({
    // 一覧と詳細の区別
    mode: z.enum(['list', 'detail']),
    // detailのときにだけ参照。型サボってる
    componentNames: z.array(z.string()).optional(),
  }),
  execute: async (args, _ctx) => {
    const result = args.mode === 'list'
      ? getList()
      : getDetailMulti(args.componentNames)
    return { content: [{ type: 'text', text: result }] }
  },
})
```

このふたつの
関数を実装

- 前提
- **実装**
 - Step1: catで最小限実装
 - Step2: Props型の提供
 - Step3: Storyの提供
- 知見
- 今後やりたいこと

- 一覧：UI Packageのindex.tsファイルをcatして提供
 - Component名とimport pathが入っている

```
/* component */  
export { Badge } from '@component/Badge'  
export { Breadcrumb } from '@component/Breadcrumb'  
export { BulletsList } from '@component/BulletsList'  
export { Button } from '@component/Button'  
export { ButtonGroup } from '@component/ButtonGroup'  
export { AsyncButton } from '@component/AsyncButton'  
export { ButtonBase } from '@component/ButtonBase'  
export { IconButton } from '@component/IconButton'
```

- 詳細：storybookファイルをcatして提供
 - LLMにComponent file pathの配列を引数として送らせる
 - MCP内部で *.stories.tsx に読み替えて中身catして結合して返却

```
export async function listCatalog() {  
  const indexPath = "../../middleware/kwlib/npm/ui/src/index.ts"  
  const results = await `cat ${indexPath}`.text(); // $ is zx  
  return `${results.join("\n")}`  
}
```

=====

[PROMPT]

- 今の文脈で利用すべきComponentはどれですか？

必要なComponent名を全て出力してから次のステップを考えてください。

- Componentを使って実装する前に、MCPで詳細情報を取得してください。

- 複数一度に取得できます

`;
}

呼び出しの連鎖が上手くいくように
返却するテキストの最後に
プロンプトを仕込んでおいた

```
export async function getDetailMulti(componentPaths: string[]) {
  const results = await Promise.all(
    componentPaths.map(async (componentPath) => {
      const storyPath = componentPath.replace(".tsx", ".stories.tsx");
      const cat = await `cat ${storyPath}`.text();
      return `
=====
target: ${componentPath}
以下のstorybookのコードを参考にしてください。
---
${cat}`;
    })
  );
  return results.join("\n");
}
```

これだけでも驚く成果が出た。スクショからほぼそれ通りの実装ができる。

株式会社ナレッジワーク

メールアドレス

パスワード

ログイン

パスワードの設定はこちら

[利用規約](#) [プライバシーポリシー](#) に
同意の上ログインしてください。

[ログインでお困りの方はこちら](#)



_KNOWLEDGE WORK

メールアドレス

パスワード

ログイン

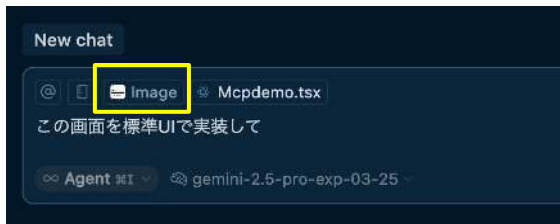
パスワードの設定はこちら

[利用規約](#) [プライバシーポリシー](#) に同意の上ログインしてください。

[ログインでお困りの方はこちら](#)

Cursorからの呼び出しイメージ。Gemini 2.5 ProがComponentの選択が上手い

一覧：



承知しました。標準UIコンポーネントを使用して、提示された画像のデザインを `Mcpdemo.tsx` に実装します。

まず、どのような標準UIコンポーネントが利用可能か確認します。次に、必要なコンポーネントの詳細情報を取得して実装を進めます。

標準UIコンポーネントのリストを取得します。

> Called MCP tool UI_Component_Catalog ✓

リストを確認しました。以下のコンポーネントが必要そうです。

- `@kwork/ui`
 - `Stack`: 要素を縦または横に並べる
 - `Heading`: タイトル表示 ("株式会社ナレッジワーク")
 - `Text`: ラベルや説明文の表示
 - `Input`: メールアドレス入力
 - `Button`: ログインボタン、パスワード設定ボタン
 - `Link`: 利用規約、プライバシーポリシー、困ったときのリンク
 - `Icon`: パスワード表示/非表示アイコン
- `@kwork/ui-raw`
 - `FormField`: ラベルと入力フィールドを組み合わせる (`FormFieldV2` が `@kwork/ui` にあるようなのでそちらを優先)
 - パスワード入力フィールド (`PasswordTextField` のような専用コンポーネントが見当たらないため、`Input` と `IconButton` を組み合わせる必要があります)



imageとしてデザインの
スクリーンショットを添付

Cursorからの呼び出しイメージ。Gemini 2.5 ProがComponentの選択が上手い

詳細：

これらのコンポーネントの詳細情報を取得します。

`FormFieldV2`, `Input`, `IconButton` も必要になる可能性があるので、合わせて取得します。

✓ Called MCP tool `UI_Component_Catalog` ✓

Parameters:

```
{
  "mode": "detail",
  "explanation": "必要な標準UIコンポーネント (Stack, Heading, Text, :
  \"detailPaths\": [
    \"middleware/kwlib/npm/ui/src/utility/display/Stack/Stack.tsx\",
    \"middleware/kwlib/npm/ui/src/component/Heading/Heading.tsx\",
    \"middleware/kwlib/npm/ui/src/component/Text/Text.tsx\",
    \"middleware/kwlib/npm/ui/src/component/Input/Input.tsx\",
    \"middleware/kwlib/npm/ui/src/component/Button/Button.tsx\",
```

取得した情報をもとに、`Mcpdemo.tsx` と `Mcpdemo.module.css` を編集してログイン画面を実装します。

`Mcpdemo.module.css` が存在しない場合は新規作成します。

```
<div className={styles.container}>
  <Stack as="form" className={styles.loginForm} direction="vertical" gap={24}>
    <Heading level="group">株式会社ナレッジワーク</Heading>

    <Stack direction="vertical" gap={16} className={styles.formFields}>
      <FormFieldV2 size="m">
        <FormFieldV2.Label>メールアドレス</FormFieldV2.Label>
        <FormFieldV2.Content>
          <Input
            type="email"
            placeholder="name1@sample.com"
            defaultValue="name1@sample.com"
          />
        </FormFieldV2.Content>
      </FormFieldV2>

      <FormFieldV2 size="m">
        <FormFieldV2.Label>パスワード</FormFieldV2.Label>
        <FormFieldV2.Content>
          <Input
            type={passwordVisible ? 'text' : 'password'}
            placeholder="*****"
            defaultValue="*****"
            trailingAction={
              <IconButton
```

続いて、Component 詳細のcall

生成されたコード

課題：

- storybookが充実していないComponentの場合、使い方がわからない
- PropsのUnion型情報がとれないのでpropsに渡す値がわからない
 - IconのTypeなど
 - storyでも外部ファイルの定数配列を回していたので直接書いてなかった
- index.tsのexport pathやstorybookのstory以外の部分など、tokenが無駄

- 前提
- **実装**
 - Step1: catで最小限実装
 - Step2: Props型の提供
 - Step3: Storyの提供
- 知見
- 今後やりたいこと

- [ts-morph](#) で真面目にファイル解析
 - [TypeScriptのコード分析を楽しむ ts-morph入門](#)
 - ts-morph、実はAIがかなり書けます。Vibe Codingしどころ
- listではComponent名だけを返す
- detailではComponent fileからexportされているProps型を返す
 - Compound Componentも検出してComponent名とProps型を返す

うちのデザインエンジニアがやってくれました！

解決：

- Props型がとれる！
Propsの型が合わないエラーが激減
- token最適化！

課題：

- 意外とStep1よりもコーディング精度が落ちるケースもあった
 - storybookはサンプルコードとして優秀なデータらしい
 - 確かにfew-shot prompting（いくつかの例を示すプロンプト）になっている

```
### Multiple Component Details ###
#### Component Detail: Breadcrumb ####
- Name: Breadcrumb
- Import: import { Breadcrumb } from '@kwork/ui'
- Main Component <Breadcrumb> Props:
  ```ts
 {
 className?: undefined | string
 items: { label: string; path: string }[]
 }
  ```
```

実装 Step3: Storyの提供

- ts-morphを使い、StorybookファイルからexportされているStory部分だけを提供
- token数は増えるが、動作には問題なかった

いいところ取りして一旦完成！

最終的なDetailの出力 →

md形式、Props/Storyはtsコード埋め込み形式

Multiple Component Details

Component Detail: Breadcrumb

```
- Name: Breadcrumb
- Import: import { Breadcrumb } from '@kwork/ui'
- Main Component <Breadcrumb> Props:
  ```ts
 {
 className?: undefined | string
 items: { label: string; path: string }[]
 }
  ```
```

- Code Example (Storybook):

```
```tsx
export const Default: Story = {
 args: {
 items: [
 { label: 'ホーム', path: '/' },
 { label: '前のページ', path: '/dummy' },
],
 },
}
```
```

Component Detail: Heading

- 前提
- 実装
 - Step1: catで最小限実装
 - Step2: Props型の提供
 - Step3: Storyの提供
- 知見
- 今後やりたいこと

- 人間が読みやすいフォーマットで返すようにしている
 - 機械/プログラムとして捉えてついjsonで返したくなるが、普通に人間が読みやすい平文テキストで返したほうがtoken効率もいいしログ上での確認もしやすい
 - 構造化したければ[xmlがよさそう](#) (jsxコードが交じるときはややこしいかも)
- ドキュメントではなくMCPを使う理由
 - MCPにすることで、ソースコードという確実なSSoTを動的に参照できる
 - 必要な情報はすべてそこにある
 - 追加情報が必要なら、それは人間にとっても必要なのでコードに書くべし

- 前提
- 実装
 - Step1: catで最小限実装
 - Step2: Props型の提供
 - Step3: Storyの提供
- 知見
- 今後やりたいこと

- インプットをスクリーンショットからFigmaにしたい
 - Figma MCP (Figma Configで出るかと思ったら出なかったですね)
 - Figma ComponentとReact Componentの命名を一致させているので、何をどう使うかが明確になりそう
- 実装した画面をPlaywright MCPのvisual modeで見に行かせて自己改善させる…
ができればすごそう

できる喜びが巡る日々を届ける

Deliver the joy of enablement