DaisyUI Web Programming: Professional

(Integrative-Generative AI Edition)

Author: Student Price Book Center

คำนำ

การพัฒนาเว็บแอปพลิเคชันในยุคปัจจุบันไม่ได้จำกัดอยู่เพียงการสร้างหน้าเว็บที่สวยงามเพียงอย่างเดียว แต่ยังรวมถึงการออกแบบระบบ UI ที่ **สอดคล้อง ยืดหยุ่น และสามารถนำกลับมาใช้ซ้ำได้** ในระดับทีมและโปรเจกต์ขนาดใหญ่ การเลือกเครื่องมือและเฟรมเวิร์กที่เหมาะสมจึงมีความสำคัญอย่างยิ่ง DaisyUI เป็นหนึ่งในเครื่องมือที่ตอบโจทย์เหล่านี้ได้ครบถ้วน ด้วยความสามารถในการทำงานร่วมกับ Tailwind CSS อย่างลื่นไหล พร้อมทั้งมีคอมโพเนนต์ UI ที่พร้อมใช้งาน ทำให้การพัฒนาเว็บเร็วขึ้นและมีมาตรฐาน

หนังสือเล่มนี้มุ่งเน้นการพัฒนา **DaisyUI Web Programming ในระดับ Professional** โดยเนื้อหาจะนำผู้อ่านตั้งแต่การสร้างระบบ UI ขั้นพื้นฐาน จนถึงการสร้าง **Design System แบบมืออาชีพ, การจัดการสถานะ, การจัดการธีมขั้นสูง, การปรับแต่งประสิทธิภาพ, และการประยุกต์ใช้ในโปรเจกต์จริง** ผ่านตัวอย่างและโค้ดตัวอย่างที่ครบถ้วน หนังสือเล่มนี้ไม่เพียงแค่ให้ความรู้ด้านทฤษฎี แต่ยังเป็นแนวทางปฏิบัติสำหรับนักพัฒนาที่ต้องการสร้างแอปพลิเคชันที่ **สวยงาม, ยืดหยุ่น, และพร้อมใช้งานจริงในโปรดักชัน**

ใน **บทที่ 16: Design System with DaisyUI** ผู้อ่านจะได้เรียนรู้การสร้าง **Component Library ภายในทีม**, การกำหนด **custom class** เช่น btn-brand เพื่อตอบโจทย์เอกลักษณ์ของแบรนด์, และการใช้ DaisyUI ร่วมกับ **Storybook** เพื่อจัดทำเอกสารประกอบคอมโพเนนต์แบบ Interactive นอกจากนี้ยังมีตัวอย่างบูรณาการ, Ultimate และ Super Ultimate Project ที่รวมฟีเจอร์หลายด้าน เช่น Team Dashboard, E-Commerce, และ Auth Library เพื่อให้เห็นภาพการใช้งานจริงของระบบ Design System

ต่อมาใน **บทที่ 17: DaisyUI + State Management** หนังสือจะเจาะลึกถึงการผสาน DaisyUI เข้ากับเครื่องมือจัดการสถานะยอดนิยมอย่าง **Redux, Zustand และ Pinia** ตัวอย่าง UI เช่น **Shopping Cart และ Dashboard Analytics** จะช่วยให้เห็นภาพการจัดการสถานะแบบเรียลไทม์ พร้อมทั้งแนวทางสร้าง **Prop-driven DaisyUI Wrapper** เพื่อให้คอมโพเนนต์สามารถนำกลับมาใช้ซ้ำได้ง่าย การออกแบบคอมโพเนนต์ในลักษณะนี้ช่วยให้ทีมสามารถพัฒนา UI ที่ตอบสนองต่อข้อมูลและสถานะได้อย่างมีประสิทธิภาพ บทนี้ยังมีตัวอย่างบูรณาการ, Ultimate และ Super Ultimate Project สำหรับการประยุกต์ใช้จริง

ใน **บทที่ 18: Advanced Theme Engineering** ผู้อ่านจะได้เรียนรู้เทคนิคการจัดการธีมขั้นสูงสำหรับ **โปรเจกต์ขนาดใหญ่และ multi-tenant SaaS** โดยครอบคลุมการสร้าง **Multiple Themes**, การปรับธีมแบบไดนามิกตามผู้ใช้ และการใช้ **Theme Inheritance ร่วมกับ Custom CSS Variables** เทคนิคเหล่านี้ช่วยให้การบริหารธีมในระบบซับซ้อนมีความยืดหยุ่นและง่ายต่อการบำรุงรักษา พร้อมทั้งตัวอย่างบูรณาการ, Ultimate และ Super Ultimate Project ที่แสดงการใช้งานธีมในบริบทจริง

บทที่ 19: Performance Optimization จะสอนวิธีเพิ่มประสิทธิภาพ DaisyUI ให้เว็บโหลดเร็วและใช้งานลื่นไหล โดยครอบคลุมการใช้ **PurgeCSS เพื่อลด unused CSS**, การ **Tree-shaking DaisyUI components**, และการ **โหลดเฉพาะคอมโพเนนต์ที่ใช้งานจริง** ผ่านเทคนิค Lazy Loading และ Tailwind JIT การปรับแต่งเหล่านี้จะช่วยให้เว็บแอปพลิเคชันไม่เพียงแต่สวยงาม แต่ยังตอบสนองได้รวดเร็วและมีประสิทธิภาพสูง บทนี้ยังมีตัวอย่างบูรณาการ, Ultimate และ Super Ultimate Project Overview เพื่อให้ผู้อ่านเห็นภาพการนำไปใช้ใน production

สุดท้ายใน **บทที่ 20: Real-world Projects** หนังสือจะพาผู้อ่านเข้าสู่การประยุกต์ใช้ DaisyUI ใน **โปรเจกต์จริง** ครอบคลุมทั้ง **E-Commerce UI** (Product Listing, Cart, Checkout), **Admin Dashboard** (Sidebar, Charts, Table) และ **SaaS Application UI** (Auth, Profile, Settings, Dark Mode) โดยเน้นการออกแบบ UI ที่ตอบโจทย์ธุรกิจ พร้อมตัวอย่าง Ultimate และการต่อยอดให้เหมือนแอป SaaS จริง ๆ เพื่อให้ผู้อ่านสามารถปิดท้ายการเรียนรู้ด้วยประสบการณ์ที่ครบวงจร ตั้งแต่การออกแบบ, การพัฒนา, ไปจนถึงการนำไปใช้จริงในโปรเจกต์ระดับมืออาชีพ

คำนำนี้ตั้งใจให้ผู้อ่านเห็นภาพรวมของหนังสือและการเดินทางตั้งแต่การเรียนรู้ DaisyUI แบบพื้นฐานไปจนถึงการประยุกต์ใช้งานจริงในโปรเจกต์ขนาดใหญ่ ด้วยตัวอย่างเชิงลึก, เทคนิคเชิงปฏิบัติ, และ Super Ultimate Projects ที่ครอบคลุมทุกด้าน ผู้อ่านจะได้รับแนวทางครบถ้วนสำหรับการสร้าง **เว็บแอปพลิเคชันที่สวยงาม, ยืดหยุ่น, และพร้อมใช้งานจริงในระดับ Professional**

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 16

Design System with DaisyUI

(Design System with DaisyUI)

เนื้อหา

- Design System with DaisyUI
- Design System with DaisyUI (เชิงลึก)
- การสร้าง Component Library ภายในทีม
- การกำหนด custom class (เช่น btn-brand)
- การใช้ DaisyUI ร่วมกับ Storybook
- ตัวอย่างบูรณาการ
- Ultimate Examples
- Super Ultimate Example – “Team Dashboard + E-Commerce + Auth Library

---

บทนำบทที่ 16: Design System with DaisyUI

ในโลกของการพัฒนาเว็บสมัยใหม่ การออกแบบและพัฒนา UI ไม่ได้มุ่งเน้นเพียงแค่ความสวยงามเท่านั้น แต่ยังรวมถึงความสม่ำเสมอและการใช้งานซ้ำได้อย่างมีประสิทธิภาพ การสร้าง **Design System** คือหัวใจสำคัญที่ช่วยให้ทีมสามารถทำงานร่วมกันได้อย่างเป็นระบบ และหลีกเลี่ยงความซ้ำซ้อนในการออกแบบหน้าต่างๆ ของแอปพลิเคชัน บทนี้จะนำเสนอการประยุกต์ใช้ DaisyUI เพื่อสร้าง Design System ที่สามารถขยายผลได้จริงในระดับทีมพัฒนา

เนื้อหาแรกจะมุ่งไปที่ **การสร้าง Component Library ภายในทีม** ซึ่งถือเป็นรากฐานของ Design System การรวมชุดคอมโพเนนต์ เช่น ปุ่ม ฟอร์ม การ์ด และเมนู ไว้ในรูปแบบที่ทีมทุกคนสามารถเข้าถึงและใช้งานร่วมกันได้ จะช่วยลดปัญหาความไม่สอดคล้องในการออกแบบ อีกทั้งยังทำให้การพัฒนา UI เร็วขึ้นและง่ายต่อการบำรุงรักษาในระยะยาว

ถัดมาคือการเรียนรู้วิธี **กำหนด custom class** เพื่อปรับแต่ง DaisyUI ให้เข้ากับแบรนด์หรือเอกลักษณ์เฉพาะของโครงการ เช่น การสร้างปุ่มใหม่ด้วย class btn-brand ที่สะท้อนโทนสีและสไตล์เฉพาะขององค์กร การกำหนด custom class เหล่านี้ทำให้ทีมสามารถนำ DaisyUI มาต่อยอดได้อย่างยืดหยุ่นและไม่จำกัดอยู่เพียงค่า default ที่ DaisyUI มีให้

อีกหนึ่งหัวข้อที่สำคัญคือการ **ใช้ DaisyUI ร่วมกับ Storybook** ซึ่งเป็นเครื่องมือที่ช่วยให้การจัดการและทดสอบคอมโพเนนต์ UI ทำได้เป็นระบบ Storybook ทำหน้าที่เป็นเอกสารประกอบเชิง

โต้ตอบ (Interactive Documentation) ที่ช่วยให้นักพัฒนาและนักออกแบบสามารถเห็นพฤติกรรมและการใช้งานคอมโพเนนต์ได้ทันที ทำให้การสื่อสารในทีมราบรื่น และลดความผิดพลาดจากความเข้าใจที่คลาดเคลื่อน

สุดท้าย บทนี้จะทำหน้าที่เป็นคู่มือเชิงปฏิบัติสำหรับการสร้าง Design System ที่ขับเคลื่อนด้วย DaisyUI ตั้งแต่ขั้นพื้นฐานของการรวมคอมโพเนนต์ ไปจนถึงการจัดการ class เฉพาะแบรนด์ และการสร้างระบบเอกสารคอมโพเนนต์ด้วย Storybook เป้าหมายคือเพื่อให้ผู้อ่านสามารถสร้าง Design System ที่ครบถ้วน ยืดหยุ่น และพร้อมรองรับการพัฒนาร่วมกันในทีมได้อย่างมืออาชีพ

Design System with DaisyUI

- การสร้าง Component Library ภายในทีม
- กำหนด custom class (เช่น btn-brand)
- ใช้ DaisyUI ร่วมกับ Storybook

บทนี้จะพาคุณจากแนวคิดของระบบดีไซน์ (Design System) ไปสู่การสร้าง **Component Library** ภายในทีมโดยใช้ **DaisyUI + Tailwind** เป็นฐาน — รวมทั้งตัวอย่างการสร้าง btn-brand แบบที่ทีมใช้ร่วมกัน และการนำทุกอย่างไปรันเป็น **Storybook** เพื่อเป็น Living Documentation / QA environment

---

1) แนวคิดสำคัญก่อนลงมือ

- **Design tokens** — สี, ระยะขอบ, ฟ้อนต์, เงา ฯลฯ ควรถูกเก็บเป็นตัวแปร (theme) เพื่อให้ปรับได้จากจุดเดียว
- **Atomic components + Composition** — แยกเป็น primitive (Button, Input, Card) และ compound (Form, Modal)
- **API ของ component ต้องชัดเจน** — props, events, accessibility contract (aria)
- **ธีม & runtime switch** — DaisyUI รองรับ data-theme ทำให้ switch ธีมได้ง่าย
- **Documentation & Visual tests** — Storybook เป็นหัวใจของ process: design ↔ dev ↔ QA

---

2) การสร้าง Component Library ภายในทีม — โครงสร้างและ workflow (ตัวอย่าง)

แนะนำใช้ monorepo (optional) หรือ repo เดียวมีโฟลเดอร์แยกชัด:

```
design-system/
├── package.json
├── tailwind.config.js
├── src/
│   ├── index.css          # tailwind + daisyui + component styles
```

```
│   ├── components/
│   │   ├── Button/
│   │   │   ├── Button.tsx
│   │   │   ├── Button.stories.tsx
│   │   │   └── index.ts
│   │   └── ...
│   └── tokens/
│       └── tokens.css
├── .storybook/
│   ├── main.js
│   └── preview.js
└── README.md
```

Tooling ที่แนะนำ

- TypeScript, ESLint, Prettier
- TailwindCSS + DaisyUI (plugin)
- Storybook (living docs)
- Testing: Jest + React Testing Library (unit) + Playwright / Chromatic (visual/regression)
- Build: Rollup / Vite to bundle ui package (ถ้าต้อง publish)

Process (ทีม)

1. Component RFC (short spec: props, variants, accessibility notes, visual)
2. Dev in branch + Storybook stories (interactive examples + controls/args)
3. Unit tests + accessibility tests (axe) + visual snapshots
4. Review (code + visual) ⟶ merge ⟶ release (semver)

---

3) เก็บ tokens & ธีม — วิธีใช้งานกับ DaisyUI

วิธีที่ดี: เก็บ color tokens ใน tailwind.config.js ผ่าน DaisyUI theme เพื่อให้ DaisyUI เข้าใจ primary/secondary ฯลฯ แล้วใช้ data-theme เพื่อสลับ

ตัวอย่าง tailwind.config.js (ย่อ)

```
module.exports = {
  content: ["./src/**/*.{js,ts,jsx,tsx,html}"],
  theme: {
    extend: {
      // ถ้าต้องการตัวแปรเพิ่มเติมสำหรับ utility ที่ไม่ได้มาจาก daisyUI
```

```
      spacing: { '72': '18rem' },
    }
  },
  plugins: [require('daisyui')],
  daisyui: {
    themes: [
      'light',
      'dark',
      {
        'brand-theme': {
          'primary': '#0ea5a4',
          'secondary': '#7c3aed',
          'accent': '#f97316',
          'neutral': '#111827',
          // ...อื่นๆ ตามต้องการ
        }
      }
    ],
    darkTheme: 'dark'
  }
}
```

ใน runtime:

```
<html data-theme="brand-theme"> ... </html>
```

หรือเปลี่ยน programmatically:

```
document.documentElement.setAttribute('data-theme', 'brand-theme')
```

---

4) กำหนด custom class — ตัวอย่าง btn-brand

มีสองแนวทางหลัก: (A) ใช้ @layer components + @apply เพื่อสร้าง class ใหม่ (B) ใช้ DaisyUI theme แล้วเรียก btn btn-primary (เมื่อ primary ชี้ไปยัง brand color)

A. สร้าง .btn-brand ด้วย Tailwind @apply (ง่ายและควบคุมได้)

src/index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

```
/* Design tokens (optional) */
:root {
  --brand-500: #06b6d4;
  --brand-600: #0891b2;
}

/* Component layer */
@layer components {
  .btn-brand {
    @apply btn; /* daisyUI base button */
    @apply text-white font-medium;
    background-image: linear-gradient(90deg, var(--brand-500), var(--brand-600));
    border: none;
    padding: .5rem 1rem;
    border-radius: .5rem;
  }
  .btn-brand:focus {
    @apply ring-4 ring-offset-2;
    box-shadow: 0 0 0 4px rgba(6,182,212,0.18);
  }
}
```

B. กำหนด theme แล้วใช้ btn btn-primary

ถ้า daisyui.themes กำหนด 'primary' เป็นสีแบรนด์แล้ว:

```
<button class="btn btn-primary">Primary (brand)</button>
```

ข้อดี: เปลี่ยนธีมครั้งเดียว ทุก btn btn-primary เปลี่ยนตามไปด้วย

ตัวอย่าง React Button component (TypeScript)

```
import React from 'react';

type Variant = 'primary' | 'secondary' | 'brand' | 'ghost' | 'outline';

export interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {
  variant?: Variant;
}
```

```
export const Button: React.FC<ButtonProps> = ({ variant = 'primary', className = '', children,
...rest }) => {
  const base = 'btn';
  const variantClass = variant === 'brand' ? 'btn-brand' : `btn btn-${variant}`;
  return (
    <button className={`${variantClass} ${className}`} {...rest}>
      {children}
    </button>
  );
};
```

---

5) ใช้ DaisyUI ร่วมกับ Storybook — ขั้นตอน & config ตัวอย่าง

เป้าหมาย: ให้ Storybook โหลด Tailwind + DaisyUI และให้ user สลับธีมจาก toolbar ได้

ติดตั้ง (ตัวอย่างแบบย่อ)

```
# ตัวอย่าง packages (project ที่มี React + tailwind)
npm i -D @storybook/react @storybook/addon-essentials
# tailwind, daisyui, postcss ควรติดตั้งอยู่แล้ว
```

.storybook/main.js

```
module.exports = {
  stories: ['../src/**/*.stories.@(js|jsx|ts|tsx)'],
  addons: ['@storybook/addon-essentials'],
  framework: '@storybook/react',
};
```

.storybook/preview.js

```
import '../src/index.css'; // รวม tailwind + daisyui + custom classes
import React from 'react';

export const parameters = {
  actions: { argTypesRegex: '^on[A-Z].*' },
  controls: { expanded: true },
};

export const globalTypes = {
```

```
  theme: {
    name: 'Theme',
    description: 'DaisyUI theme',
    defaultValue: 'light',
    toolbar: { icon: 'circlehollow', items: ['light', 'dark', 'brand-theme', 'cupcake'] }
  }
};

export const decorators = [
  (Story, context) => {
    // apply theme to document
    const theme = context.globals.theme || 'light';
    document.documentElement.setAttribute('data-theme', theme);
    return <Story />;
  }
];
```

ตัวอย่าง Story ของ Button (Button.stories.tsx)

```
import React from 'react';
import { Button } from './Button';

export default {
  title: 'Components/Button',
  component: Button,
  argTypes: {
    variant: {
      control: { type: 'select' },
      options: ['primary', 'secondary', 'brand', 'outline', 'ghost']
    },
    onClick: { action: 'clicked' }
  }
};

const Template = (args) => <Button {...args}>Click me</Button>;
```

```
export const Primary = Template.bind({});
Primary.args = { variant: 'primary' };

export const Brand = Template.bind({});
Brand.args = { variant: 'brand' };
```

ผลลัพธ์: ใน Storybook คุณจะได้

- Controls เพื่อเปลี่ยน variant แบบ realtime
- Toolbar เพื่อสลับ data-theme (DaisyUI) — ทำให้เห็น btn btn-primary ภายใต้ธีมต่างๆ
- Living documentation ที่นักออกแบบ/นักพัฒนาสามารถตรวจสอบและทดสอบได้

---

6) แนวปฏิบัติ (Best practices) — ทำให้ทีมใช้ได้จริง

- **One source of truth for colors** — เก็บใน daisyui.themes หรือ CSS vars (:root)
- **Use small, focused components** — Button ทำหน้าที่ปุ่มเท่านั้น ไม่ผูกกับ modal logic
- **Accessibility first** — ทุก component ต้องรองรับ keyboard, aria-* และ focus states
- **Story-first development** — สร้าง story ก่อนหรือพร้อมกับ component เพื่อเป็น acceptance criteria
- **Docs + Examples** — ให้แต่ละ component มี stories: default, variants, disabled, loading, accessibility demo
- **Visual regression** — เพิ่มการทดสอบภาพ (Chromatic / Playwright snapshots) ใน CI
- **Versioning & Changelog** — semantic versioning + conventional commits ⟶ ช่วย rollback และ communicate changes
- **Token migration plan** — ถ้าต้องเปลี่ยนสีหลัก ให้มี migration plan (deprecation notices / dual support)

---

7) Checklist ก่อน merge component ใหม่

- ☐ Storybook story ครบ (default + variants + edge cases)
- ☐ Unit test (render, interaction) + accessibility audit (axe)
- ☐ Visual snapshot หรือ screenshot comparison
- ☐ Docs (usage, props, accessibility)
- ☐ Export ใน index ของ library (เพื่อให้ import ง่าย)
- ☐ Changelog entry / release note (ถ้าเป็น breaking change)

---

8) ตัวอย่าง flow สั้น ๆ — จากออกแบบ ⟶ ใช้งานจริง

1. Designer ส่งสีและตัวอย่างใน Figma ⟶ map เป็น design tokens (--brand, primary)
2. Dev เพิ่ม theme ใน tailwind.config.js (daisyUI) และสร้าง .btn-brand ใน @layer components
3. สร้าง Button component (TS + tests) และ Button.stories.tsx
4. เปิด PR ⟶ reviewer ดูโค้ด + เปิด Storybook เพื่อตรวจ visual ⟶ merge
5. Release package / update monorepo ⟶ แอปอื่นในบริษัทสามารถนำไปใช้ด้วย npm i @company/design-system หรือ monorepo dependency

---

Design System with DaisyUI (เชิงลึก)

สรุปแนวคิดระดับสถาปัตยกรรม (Core principles)

- **Design tokens เป็น "แหล่งความจริงเดียว"**: สี, ระยะ, ฟอนต์, เงา ควรถูกเก็บเป็น tokens เพื่อให้ทั้งเว็บ แอปมือถือ และไฟล์ออกแบบ (Figma) ซิงก์กันได้ง่าย (การเก็บด้วย CSS variables หรือไฟล์ JSON ที่ generate เป็นค่าใน tailwind.config.js) ๐ (contentful.com)
- **Primitive ⟶ Composition**: แยกเป็น primitive components (Button, Input, Icon) ที่เล็กและชัดเจน แล้วประกอบเป็น compound (Form, Modal, Card) เพื่อให้ reuse ง่ายและทดสอบได้
- **Theme-first**: กำหนดธีม (light/dark/brand) ในระดับ tokens แล้วใช้ DaisyUI/Tailwind map ให้ primary/accent ฯลฯ ชี้ไปยัง token เหล่านั้น — จะทำให้เปลี่ยนสีระบบได้จากจุดเดียว (single source of truth). (daisyui.com)
- **Docs = Contract**: Storybook ทำหน้าที่เป็น living documentation + visual QA + playground (controls) สำหรับทีม product/designer/dev/QA

Theming กับ DaisyUI — วิธีคิดและการใช้งานจริง

1. **DaisyUI ใช้ data-theme เพื่อสลับธีมบน DOM** — คุณสามารถใส่ data-theme="brand-theme" บน <html> หรือ section ใดก็ได้ (nested themes ก็ได้) และทุกองค์ประกอบภายในจะใช้ค่าธีมนั้น. (daisyui.com)
2. **กำหนดธีมใน tailwind.config.js** — DaisyUI ยอมให้เพิ่ม custom themes หรือเลือกใช้ธีม built-in; ถ้ากำหนด primary ใน theme แล้วทุก component ที่อ้าง btn btn-primary จะเปลี่ยนตาม. (daisyui.com)

ตัวอย่าง tailwind.config.js (เชิงปฏิบัติ)

```
// tailwind.config.js
module.exports = {
  content: ["./src/**/*.{js,ts,jsx,tsx,html}"],
```

```
  theme: { extend: {} },
  plugins: [require("daisyui")],
  daisyui: {
    themes: [
      "light",
      "dark",
      {
        "brand-theme": {
          "primary": "#0ea5a4",
          "primary-focus": "#0891b2",
          "secondary": "#7c3aed",
          "accent": "#f97316",
          "neutral": "#111827",
          "base-100": "#ffffff"
        }
      }
    ],
    darkTheme: "dark"
  }
}
```

(ใช้ data-theme="brand-theme" ที่ runtime เพื่อสลับ) (daisyui.com)

สร้าง custom class เช่น btn-brand — 2 แนวทาง (ข้อดี/ข้อเสีย)

แนวทาง A — Theme-driven (แนะนำถ้าเปลี่ยนค่าสีบ่อย)

- ให้ primary ใน daisyUI map เป็นสีแบรนด์ แล้วใช้ btn btn-primary ⟶ ข้อดี: เปลี่ยนธีมครั้งเดียว ทุกปุ่มเปลี่ยนตาม, สอดคล้องกับ DaisyUI philosophy. (daisyui.com)

แนวทาง B — @layer components + @apply (ควบคุมละเอียด)

- สร้าง .btn-brand ใน CSS ของคุณโดยใช้ @layer components + @apply เพื่อรวม utility classes และเพิ่ม styles เฉพาะ เช่น gradient หรือ shadow

ตัวอย่างไฟล์ src/index.css:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

```
/* design tokens (optional) */
:root {
  --brand-500: #06b6d4;
  --brand-600: #0891b2;
}

/* component layer (ใช้ @layer เพื่อหลีกเลี่ยงปัญหา specificity และให้ Tailwind จัดการลำดับ) */
@layer components {
  .btn-brand {
    @apply btn text-white font-medium;
    background-image: linear-gradient(90deg, var(--brand-500), var(--brand-600));
    border: none;
    padding: .5rem 1rem;
    border-radius: .5rem;
  }
  .btn-brand:focus {
    @apply ring-4 ring-offset-2;
    box-shadow: 0 0 0 4px rgba(6,182,212,0.18);
  }
}
```

เหตุผลใช้ @layer components + @apply: ป้องกันปัญหา order/specifity และช่วยให้ Tailwind tree-shaking ทำงานได้ดี. (Tailwind CSS)

ตัวอย่าง React + TypeScript Button component (พร้อม accessibility)

```
// src/components/Button/Button.tsx
import React from 'react';
import cn from 'classnames';

type Variant = 'primary'|'secondary'|'brand'|'ghost'|'outline';
type Size = 'sm'|'md'|'lg';

export interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {
  variant?: Variant;
  size?: Size;
```

```
  loading?: boolean;
  children?: React.ReactNode;
}

export const Button: React.FC<ButtonProps> = ({
  variant = 'primary',
  size = 'md',
  loading = false,
  disabled,
  className,
  children,
  ...rest
}) => {
  const sizeClass = size === 'sm' ? 'btn-sm' : size === 'lg' ? 'btn-lg' : 'btn-md';
  const variantClass = variant === 'brand' ? 'btn-brand' : `btn btn-${variant}`;
  return (
    <button
      className={cn(variantClass, sizeClass, className, { 'opacity-70 cursor-not-allowed':
disabled || loading })}
      aria-disabled={disabled || loading}
      {...rest}
    >
      {loading && (
        <svg className="animate-spin -ml-1 mr-2 h-4 w-4" xmlns="http://www.w3.org/2000/svg"
fill="none" viewBox="0 0 24 24" aria-hidden>
          <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor"
strokeWidth="4"></circle>
          <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8v4a4 4 0 00-4
4H4z"></path>
        </svg>
      )}
      <span>{children}</span>
    </button>
  );
```

```
};
```

- จุดสำคัญ: ให้ aria-disabled, keyboard-focus, และแสดง loading spinner ภายในปุ่ม — เป็นมาตรฐาน accessibility ที่ต้องมี

Storybook — ทำให้เป็น Living Docs + Theme Switcher

แนวทางพื้นฐาน

1. โหลด CSS (Tailwind + DaisyUI + custom) เข้าใน Storybook preview
2. เพิ่ม toolbar/global to switch data-theme (Storybook globalTypes) หรือใช้ addon ที่มี (เช่น storybook-addon-data-theme-switcher / storybook-addon-theme-changer) เพื่อสลับ data-theme ใน iframe ของ Storybook. (Storybook)

ตัวอย่าง .storybook/main.js

```
module.exports = {
  stories: ['../src/**/*.stories.@(js|jsx|ts|tsx)'],
  addons: ['@storybook/addon-essentials'],
  framework: '@storybook/react',
};
```

ตัวอย่าง .storybook/preview.js

```
import '../src/index.css'; // tailwind + daisyui + custom styles

export const parameters = {
  actions: { argTypesRegex: '^on[A-Z].*' },
};

export const globalTypes = {
  theme: {
    name: 'Theme',
    defaultValue: 'light',
    toolbar: {
      icon: 'circlehollow',
      items: ['light', 'dark', 'brand-theme', 'cupcake'],
    },
  },
};
```

```
export const decorators = [
  (Story, context) => {
    const theme = context.globals.theme || 'light';
    // note: Storybook renders stories inside an iframe; setting on documentElement works for most setups
    document.documentElement.setAttribute('data-theme', theme);
    return <Story />;
  },
];
```

- ถ้าต้องการการสลับ data-theme ที่แม่นยำกับ iframe ให้ใช้ storybook-addon-data-theme-switcher (จะเซ็ต data-theme บน iframe html) — addon นี้เจาะจงงานแบบ DaisyUI ได้ดี. (Storybook)

(มีบทความสอนการตั้งค่า step-by-step ที่ใช้งานจริงเพื่อหลีกเลี่ยง pitfalls ของ PostCSS/Storybook config). (Tailkits)

Testing & CI (เพื่อคงคุณภาพของ UI Library)

- **Unit tests**: React Testing Library + Jest — test interaction contract (onClick, keyboard)
- **Accessibility tests**: axe-core (jest-axe) ใน unit test และ axe + Playwright ใน E2E
- **Visual regression**: Chromatic / Playwright screenshots / Percy — สร้าง baseline snapshot ของทุก story แล้วล็อกไว้ใน CI เพื่อจับ regressions
- **CI flow (ตัวอย่าง)**:
  1. npm ci / install
  2. build tailwind css (postcss) + build storybook (static)
  3. run unit tests + accessibility tests
  4. run visual tests (Chromatic / Playwright)
  5. publish package (เมื่อผ่านทุกด่าน)

Packaging & Distribution (monorepo vs single package)

- **Monorepo (Nx / Turborepo / pnpm workspaces)** — ดีเมื่อมีหลายแพ็กเกจ (ui, tokens, docs). ช่วย sync version และ local dev.
- **Single package** — ง่ายกว่า ถ้าทีมเล็ก
- **CSS handling**: แนะนำให้ ui package เป็น *source* ที่ export compiled CSS (เช่น dist/styles.css) และให้ consumer เลือกนำเข้า (หรือแจ้งใน README ว่าให้ติดตั้ง

DaisyUI/Tailwind เป็น peerDependency) — เพื่อหลีกเลี่ยงการมี Tailwind/DaisyUI หลายสำเนาในการใช้งานจริง (duplicate CSS).

- **Bundler**: Rollup หรือ Vite (library mode) เพื่อ bundle JS/TS; สำหรับ CSS ให้ build via PostCSS (tailwind) เป็นไฟล์แยก.

Design tokens — sync ข้ามระบบ (Figma ↔ code)

- **Workflow แนะนำ**: Designer export tokens (JSON) ⟶ Style Dictionary หรือ Token Transform ⟶ สร้าง
  - tokens.json -> generate :root CSS vars (for non-Tailwind/native)
  - generate tailwind.config.js extension (map color names to Tailwind tokens)
- การใช้ tokens ช่วยลด friction เมื่อเปลี่ยนแบรนด์สี/spacing และทำให้ native apps ใช้ค่าเดียวกับเว็บได้. (contentful.com)

Advanced patterns & Pitfalls ที่ต้องระวัง

- **Avoid styling in markup**: หลีกเลี่ยงการผสม styles ที่ซ้ำในหลายไฟล์ — เก็บใน component class หรือ token
- **Specificity traps**: ถ้าจำเป็นต้อง override DaisyUI styles ให้เขียนใน @layer components เพื่อจัดลำดับอย่างถูกต้อง — อย่าใช้ !important เป็นทางออกแรก. (pieces.app)
- **Tree-shaking**: Tailwind คำนวณจาก content — ตรวจสอบให้แน่ใจว่า folder stories/components ถูกประกาศใน content เพื่อไม่ให้เกิด missing styles ใน Storybook/CI
- **DaisyUI themes & build**: ถาจะให้หลายธีมพร้อมกันให้แน่ใจว่าคอนฟิกของ DaisyUI ถูกเซ็ตอย่างถูกต้อง (custom themes / disabling built-in ถ้าต้องการ) 。 (daisyui.com)

ตัวอย่าง workflow สั้น — จาก idea ⟶ release

1. Designer สร้าง token (JSON) ⟶ commit to tokens/
2. Dev เพิ่ม theme ใน tailwind.config.js (map ไปยัง token) + สร้าง .btn-brand ใน @layer components
3. สร้าง component (Button) + story (Storybook) ก่อน PR
4. CI: run tests + accessibility + visual snapshot ⟶ reviewer ตรวจ storybook ⟶ merge
5. Release (semantic version) ⟶ update consumers (dependabot / internal registry)

แหล่งอ้างอิงสำคัญ (เพื่ออ่านต่อและตรวจสอบ)

- DaisyUI — themes, data-theme usage, config docs. (daisyui.com)

- DaisyUI blog — วิธีเพิ่มสีใหม่ลงใน themes. (daisyui.com)
- Tailwind docs — @apply, @layer, การเพิ่ม custom styles และแนวทาง utility-first. (Tailwind CSS)
- Storybook — Tailwind recipe และการเพิ่ม toolbar/theme switcher. (Storybook)
- บทความแนะนำการตั้งค่า Storybook + DaisyUI + Tailwind (tutorial + pitfalls). (Tailkits)
- Design tokens (เหตุผลเชิงปฏิบัติและ workflow) — Contentful / บทความ tokens. (contentful.com)

การสร้าง Component Library ภายในทีม

□ แนวคิดหลัก

การสร้าง **Component Library ภายในทีม** หมายถึงการสร้างชุด UI Components ที่เป็น **มาตรฐานเดียวกัน** ให้ทุกคนในทีมสามารถ reuse ได้ ซึ่งจะช่วยแก้ปัญหา

- ความไม่สม่ำเสมอของ UI (UI inconsistency)
- เวลาที่เสียไปกับการเขียน component เดิมซ้ำ
- ปัญหาความแตกต่างในการตีความดีไซน์

DaisyUI เข้ามาช่วยได้เพราะมัน **มี component base ให้เริ่มต้น** และสามารถ **customize** ตาม Design System ของทีมได้

□ โครงสร้างโปรเจกต์ Component Library

ตัวอย่างกรณีใช้ React + Vite + DaisyUI

design-system/

```
├── src/
│   ├── components/
│   │   ├── Button.jsx
│   │   ├── Card.jsx
│   │   ├── Input.jsx
│   │   └── index.js
│   ├── styles/
│   │   └── tailwind.css
│   └── main.jsx
├── tailwind.config.js
└── package.json
```

- components/ = เก็บทุก component ของทีม
- index.js = export รวมให้ทีมอื่น import ได้สะดวก
- tailwind.config.js = เก็บ custom theme/class ที่ใช้ร่วมกัน

□ การสร้างปุ่มมาตรฐาน (Custom Button)

ตัวอย่าง: ปุ่มหลักของแบรนด์ (btn-brand)

1. กำหนดใน tailwind.config.js

```
module.exports = {
  plugins: [require("daisyui")],
  daisyui: {
    themes: [
      {
        mytheme: {
          primary: "#1e40af", // brand blue
          secondary: "#9333ea",
          accent: "#f59e0b",
          neutral: "#3d4451",
          "base-100": "#ffffff",
        },
      },
    ],
  },
}
```

2. สร้าง Button.jsx

```
export default function Button({ children, variant = "primary", ...props }) {
  const baseClass = "btn rounded-lg px-6";
  const variantClass = {
    primary: "btn-primary",
    secondary: "btn-secondary",
    accent: "btn-accent",
    brand: "bg-blue-700 text-white hover:bg-blue-800",
  };

  return (
```

```
    <button className={`${baseClass} ${variantClass[variant]}`} {...props}>
      {children}
    </button>
  );
}
```

3. ใช้งาน

```
import Button from "./components/Button";

export default function App() {
  return (
    <div className="p-8 space-x-4">
      <Button>Default Primary</Button>
      <Button variant="secondary">Secondary</Button>
      <Button variant="brand">Brand</Button>
    </div>
  );
}
```

☐ **วิธีแชร์ให้ทั้งทีมใช้**

- **Option 1**: แยกเป็น private npm package ⟶ publish ขึ้น GitHub Packages หรือ npm private registry
- **Option 2**: ใช้ Monorepo (เช่น Nx, Turborepo) ⟶ แยก design system เป็นหนึ่ง workspace
- **Option 3**: ใช้ Storybook + Chromatic ⟶ แจก document + playground ของ component

---

☐ **Best Practices**

1. **Atomic Design Principle**
   - เริ่มจาก atom (Button, Input) ⟶ molecule (Form, Card) ⟶ organism (Navbar, Dashboard)
2. **ตั้งชื่อมาตรฐาน**
   - ใช้ prefix เช่น btn-brand, card-brand, input-brand เพื่อกันชนกันกับ DaisyUI class
3. **แยก Business Logic ออกจาก UI**
   - Component Library ควรมีแต่ UI + styling ไม่ใส่ state logic ธุรกิจ
4. **Document ให้ครบ**

- ทุก component ควรมีตัวอย่างการใช้งานและ props ที่รองรับ

โปรแกรมพื้นฐาน (3 ตัวอย่าง) และ **โปรแกรมแนวประยุกต์ (3 ตัวอย่าง)** สำหรับหัวข้อ **"การสร้าง Component Library ภายในทีม"** โดยแต่ละตัวอย่างต้องมี

- □ โครงสร้างโปรเจกต์ (Project structure)
- □ ไฟล์โค้ดเต็ม (Full file)
- □ คำอธิบายโค้ด (Code explanation)
- □ ผลการรัน (Expected result)

ผมจะจัดให้เป็น 2 ชุด

□ ชุดที่ 1: โปรแกรมพื้นฐาน (3 ตัวอย่าง)

□ ตัวอย่างที่ 1 – ปุ่มมาตรฐานใน Component Library

โครงสร้าง

```
component-lib-basic-1/
├── src/
│   ├── components/
│   │   └── Button.jsx
│   └── App.jsx
├── tailwind.config.js
└── index.html
```

ไฟล์โค้ด

tailwind.config.js

```
module.exports = {
  content: ["./src/**/*.{html,js,jsx}"],
  plugins: [require("daisyui")],
  daisyui: {
    themes: ["light"],
  },
}
```

src/components/Button.jsx

```
export default function Button({ children, variant = "primary" }) {
  const baseClass = "btn";
  const variantClass = {
```

```
    primary: "btn-primary",
    secondary: "btn-secondary",
  };

  return <button className={`${baseClass} ${variantClass[variant]}`}>{children}</button>;
}
```

src/App.jsx

```
import Button from "./components/Button";

export default function App() {
  return (
    <div className="p-10 space-x-4">
      <Button>Primary</Button>
      <Button variant="secondary">Secondary</Button>
    </div>
  );
}
```

คำอธิบาย

- เราสร้าง Button.jsx เป็น component กลาง
- ใช้ DaisyUI class (btn, btn-primary, btn-secondary)
- สามารถ reuse ได้ในทุกหน้าของทีม

ผลการรัน

แสดงปุ่ม 2 ปุ่ม: ปุ่มน้ำเงิน (Primary) และปุ่มเทา (Secondary)

☐ **ตัวอย่างที่ 2 – Card Component Library**

โครงสร้าง

```
component-lib-basic-2/
├── src/
│   ├── components/
│   │   └── Card.jsx
│   └── App.jsx
├── tailwind.config.js
└── index.html
```

ไฟล์โค้ด

src/components/Card.jsx

```
export default function Card({ title, children }) {
  return (
    <div className="card w-96 bg-base-100 shadow-xl">
      <div className="card-body">
        <h2 className="card-title">{title}</h2>
        <p>{children}</p>
      </div>
    </div>
  );
}
```

src/App.jsx

```
import Card from "./components/Card";

export default function App() {
  return (
    <div className="p-10 flex gap-4">
      <Card title="Card 1">This is the first card</Card>
      <Card title="Card 2">This is the second card</Card>
    </div>
  );
}
```

คำอธิบาย

- Card.jsx ใช้ DaisyUI card class
- ทีมสามารถส่ง title และ children มาใช้งานได้
- โครงสร้าง standardized

ผลการรัน

แสดง card 2 ใบ พร้อม title และข้อความภายใน

☐ **ตัวอย่างที่ 3 – Input Component Library**

โครงสร้าง

```
component-lib-basic-3/
├── src/
│   ├── components/
```

```
│   │   └── Input.jsx
│   └── App.jsx
├── tailwind.config.js
└── index.html
```

ไฟล์โค้ด

src/components/Input.jsx

```
export default function Input({ label, placeholder }) {
  return (
    <div className="form-control w-full max-w-xs">
      <label className="label">
        <span className="label-text">{label}</span>
      </label>
      <input type="text" placeholder={placeholder} className="input input-bordered w-full max-w-xs" />
    </div>
  );
}
```

src/App.jsx

```
import Input from "./components/Input";

export default function App() {
  return (
    <div className="p-10 space-y-4">
      <Input label="Username" placeholder="Enter your username" />
      <Input label="Email" placeholder="Enter your email" />
    </div>
  );
}
```

คำอธิบาย

- Input.jsx เป็น input component กลาง
- ใช้ DaisyUI class input input-bordered
- ลดซ้ำซ้อนเวลา dev ทีมต้องสร้างฟอร์ม

ผลการรัน

แสดงฟิลด์กรอก Username และ Email ที่มี label มาตรฐาน

☐ ชุดที่ 2: โปรแกรมแนวประยุกต์ (3 ตัวอย่าง)

☐ ตัวอย่างที่ 4 – Authentication Form (รวม Button + Input)

โครงสร้าง

```
component-lib-adv-1/
├── src/
│   ├── components/
│   │   ├── Button.jsx
│   │   └── Input.jsx
│   └── App.jsx
├── tailwind.config.js
└── index.html
```

ไฟล์โค้ด

src/App.jsx

```
import Button from "./components/Button";
import Input from "./components/Input";

export default function App() {
  return (
    <div className="flex items-center justify-center h-screen bg-base-200">
      <div className="card w-96 bg-base-100 shadow-xl p-6 space-y-4">
        <h2 className="text-2xl font-bold text-center">Login</h2>
        <Input label="Email" placeholder="Enter email" />
        <Input label="Password" placeholder="Enter password" />
        <Button variant="primary">Login</Button>
      </div>
    </div>
  );
}
```

ผลการรัน

ฟอร์ม Login สวยงาม มีกล่อง card + input + ปุ่ม Login

☐ **ตัวอย่างที่ 5 – Product Card Grid (E-Commerce Style)**

โครงสร้าง

```
component-lib-adv-2/
├── src/
│   ├── components/
│   │   ├── Card.jsx
│   │   └── Button.jsx
│   └── App.jsx
├── tailwind.config.js
└── index.html
```

ไฟล์โค้ด

src/App.jsx

```
import Card from "./components/Card";
import Button from "./components/Button";

export default function App() {
  const products = [
    { id: 1, name: "Laptop", desc: "High performance laptop" },
    { id: 2, name: "Phone", desc: "Smartphone with great camera" },
    { id: 3, name: "Headphones", desc: "Noise cancelling headphones" },
  ];

  return (
    <div className="grid grid-cols-3 gap-6 p-10">
      {products.map((p) => (
        <Card key={p.id} title={p.name}>
          {p.desc}
          <Button variant="primary" className="mt-2">Buy Now</Button>
        </Card>
      ))}
    </div>
  );
}
```

ผลการรัน

Grid 3 ใบ แสดงสินค้าพร้อมปุ่ม **Buy Now**

☐ **ตัวอย่างที่ 6 – Dashboard Page (รวม Navbar + Card)**

โครงสร้าง

component-lib-adv-3/

```
├── src/
│   ├── components/
│   │   ├── Card.jsx
│   │   ├── Button.jsx
│   │   └── Navbar.jsx
│   └── App.jsx
├── tailwind.config.js
└── index.html
```

ไฟล์โค้ด

src/components/Navbar.jsx

```
export default function Navbar() {
  return (
    <div className="navbar bg-base-300 px-4">
      <a className="btn btn-ghost normal-case text-xl">Team Dashboard</a>
    </div>
  );
}
```

src/App.jsx

```
import Navbar from "./components/Navbar";
import Card from "./components/Card";

export default function App() {
  return (
    <div className="h-screen flex flex-col">
      <Navbar />
      <div className="grid grid-cols-3 gap-6 p-6">
        <Card title="Users">120 Active Users</Card>
        <Card title="Revenue">$12,300 this month</Card>
```