Tailwind CSS

FRAMEWORK: ADVANCE (INTEGRATIVE-GENERATIVE AI EDITION)

Contents:

Author: Student Price Book Center

คำนำ

ในโลกของการพัฒนาเว็บไซต์ยุคใหม่ ความเร็วในการออกแบบและความสามารถในการสร้างระบบ UI ที่มีประสิทธิภาพ กำลังกลายเป็นปัจจัยสำคัญที่ขับเคลื่อนการแข่งขันทางดิจิทัล Tailwind CSS — หนึ่งใน CSS Framework ที่เติบโตเร็วที่สุดในช่วงไม่กี่ปีที่ผ่านมา — ได้พิสูจน์ตัวเองว่าเป็นเครื่องมือทรงพลังที่ตอบโจทย์การพัฒนาเชิงโมดูลาร์ และสนับสนุนแนวคิดการออกแบบ UI แบบ Atomic อย่างแท้จริง หนังสือ *Tailwind CSS Framework: Advance* เล่มนี้ จึงถูกเรียบเรียงขึ้นเพื่อยกระดับความเข้าใจจากพื้นฐานสู่การประยุกต์ใช้ขั้นสูง ครอบคลุมเทคนิค รูปแบบ และการรวมระบบในระดับมืออาชีพ

หนังสือเล่มนี้ประกอบด้วยเนื้อหาลึกซึ้ง 5 บทหลัก ที่เจาะลึกหัวข้อซับซ้อนในการใช้งาน Tailwind CSS ให้เกิดประสิทธิผลสูงสุด ไม่ว่าจะเป็นการออกแบบ component ที่นำกลับมาใช้ซ้ำได้ (Reusable Component Patterns) การจัดระบบ layout ที่ซับซ้อนแบบ responsive การควบคุมอนิเมชันอย่างมีชีวิตชีวา ไปจนถึงการสร้างประสบการณ์แบบ Dark Mode และการรวมเข้ากับ Framework ต่าง ๆ ได้อย่างราบรื่น ทั้งหมดนี้เรียบเรียงด้วยภาษาวิชาการ อ่านเข้าใจง่าย และสามารถนำไปประยุกต์ใช้ได้จริง

บทที่ 11 ว่าด้วยเรื่อง **Reusable Component Patterns** เป็นหัวใจสำคัญของการพัฒนา UI สมัยใหม่ที่เน้นการเขียน CSS แบบไม่ซ้ำซ้อน (DRY - Don’t Repeat Yourself) โดยผู้อ่านจะได้เรียนรู้การใช้ @apply อย่างมีประสิทธิภาพ เทคนิคการสร้าง custom utility class การผสานโครงสร้าง BEM เข้ากับ Tailwind CSS รวมถึงการใช้ SCSS เพื่อจัดการ style อย่างเป็นระบบ สิ่งเหล่านี้ช่วยให้การออกแบบ UI มีความยืดหยุ่นและดูแลรักษาได้ง่ายขึ้นในระยะยาว

บทที่ 12 เจาะลึก **Layout System ขั้นสูง** สำหรับการจัดวางเนื้อหาและองค์ประกอบบนหน้าเว็บ โดยมีการสาธิตการใช้งาน nested layout ทั้งในรูปแบบ Flex และ Grid รวมถึงการใช้คลาสระดับสูง เช่น aspect-ratio, min-h-screen, max-w-screen-lg และการใช้ Container Query เพื่อสร้าง layout ที่ตอบสนองต่อบริบทของแต่ละองค์ประกอบ ซึ่งเป็นแนวทางใหม่ของ Responsive Design ที่ทรงพลัง

บทที่ 13 ว่าด้วย **Transition & Animation** ที่ช่วยเพิ่มมิติให้กับการนำเสนอ UI ในระดับมืออาชีพ ผ่านการควบคุม transition อย่างละเอียด เช่น transition-all, ease-in-out, duration-300 รวมถึงการใช้คลาส animation ที่มีให้ เช่น animate-bounce, animate-spin, animate-ping ตลอดจนการสร้าง keyframe animation แบบ custom ผ่าน tailwind.config.js เพื่อรองรับการเคลื่อนไหวที่ตรงตามความต้องการอย่างแท้จริง

บทที่ 14 กล่าวถึง **Dark Mode** ซึ่งกลายเป็นฟีเจอร์มาตรฐานของแอปพลิเคชันสมัยใหม่ หนังสือจะอธิบายการตั้งค่าระบบ dark mode ทั้งแบบ media และแบบ class-based รวมถึงตัวอย่างการสร้างปุ่ม toggle แบบอินเทอร์แอคทีฟ การสลับระหว่างโหมดต่าง ๆ และการจัดการ state ของโหมดภายใน component อย่างเหมาะสม

สุดท้ายในบทที่ 15 ผู้อ่านจะได้เรียนรู้การ **Integration กับ Framework** ชั้นนำของยุค เช่น React, Next.js, Laravel, Svelte และ Astro โดยเน้นวิธีการติดตั้ง Tailwind CSS ร่วมกับระบบ

template engine หรือ component framework เหล่านั้น พร้อมเทคนิคการใช้ร่วมกับ component library อย่าง Headless UI เพื่อสร้าง interface ที่ยืดหยุ่นและขยายได้ง่าย

การอ่านหนังสือเล่มนี้ไม่เพียงช่วยให้ผู้อ่านมีความเข้าใจลึกซึ้งใน Tailwind CSS แต่ยังปลูกฝังแนวคิดเชิงสถาปัตยกรรมที่จำเป็นต่อการสร้างระบบ front-end ขนาดใหญ่ ไม่ว่าคุณจะเป็นนักพัฒนา front-end ที่ต้องดูแลแอปพลิเคชันระดับองค์กร นักออกแบบ UI ที่ต้องการ framework ที่ตอบสนองกับแนวคิด design system หรือ full-stack developer ที่ต้องเขียนโค้ด CSS อย่างมีระบบ หนังสือเล่มนี้จะเป็นแหล่งความรู้ชั้นดี ที่ช่วยเสริมสร้างทักษะและเพิ่มพูนมุมมองใหม่ ๆ ในการใช้ Tailwind CSS อย่างแท้จริง

หวังเป็นอย่างยิ่งว่า *Tailwind CSS Framework: Advance* จะเป็นคู่มือที่ช่วยเติมเต็มองค์ความรู้ของผู้อ่านในการสร้างประสบการณ์ผู้ใช้อันยอดเยี่ยมบนเว็บยุคใหม่ ด้วยพลังของ Utility-first CSS ที่ผสานแนวคิด component-driven design อย่างลงตัว

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 11

Reusable Component Patterns

(Reusable Component Patterns)

เนื้อหา

- Reusable Component Patterns
- Reusable Component Patterns (รายละเอียดเชิงลึก)
- การใช้ @apply ในไฟล์ CSS
- การสร้าง Custom Utility Class ใน Tailwind CSS
- การใช้โครงสร้าง BEM (Block Element Modifier) ร่วมกับ Tailwind CSS
- การใช้ Preprocessor เช่น SCSS ร่วมกับ Tailwind CSS

บทนำ บทที่ 11: Reusable Component Patterns

ในโลกของการพัฒนาเว็บยุคปัจจุบัน ความสามารถในการสร้างและนำกลับมาใช้ซ้ำ (Reusable) ของส่วนประกอบ UI (User Interface Components) ถือเป็นปัจจัยสำคัญที่ส่งผลต่อประสิทธิภาพของการเขียนโค้ด ความยืดหยุ่นในการปรับแต่ง และความสามารถในการบำรุงรักษาระบบในระยะยาว บทที่ 11 นี้จะพาผู้อ่านเข้าสู่แนวคิดและรูปแบบการพัฒนา Reusable Component ด้วยแนวทางที่สอดคล้องกับสถาปัตยกรรมของ Tailwind CSS โดยเน้นการเขียน CSS อย่างมีระบบและสามารถนำกลับมาใช้ซ้ำได้อย่างแท้จริง

หัวใจสำคัญของการสร้าง component ที่นำกลับมาใช้ซ้ำได้ คือการลดการเขียนโค้ดที่ซ้ำซ้อน (Duplication) และเน้นการสร้างโครงสร้างโค้ดที่กระชับ ชัดเจน และดูแลรักษาได้ง่าย ด้วยเหตุนี้ เทคนิคหนึ่งที่เป็นที่นิยมในหมู่นักพัฒนา Tailwind CSS คือการใช้ @apply ซึ่งเป็นคำสั่งใน PostCSS ที่ช่วยรวมกลุ่ม class utility หลายรายการให้อยู่ในรูปแบบของ class เดียวที่กำหนดเอง ช่วยให้โค้ดดูสะอาดและสื่อความหมายได้ชัดเจนขึ้น

นอกจาก @apply แล้ว แนวทางอีกหนึ่งที่สำคัญในการสร้างความเป็นระบบคือการกำหนด **Custom Utility Class** ซึ่งเปิดโอกาสให้นักพัฒนาสร้าง class ใหม่ที่สอดคล้องกับบริบทของโปรเจกต์ โดยการผสมผสาน class พื้นฐานจาก Tailwind CSS และปรับแต่งให้เหมาะกับรูปแบบของแบรนด์หรือ UI ที่ต้องการสื่อสาร วิธีนี้ทำให้ทีมสามารถแชร์และใช้ component แบบเดียวกันได้อย่างมีประสิทธิภาพมากขึ้น

อีกหนึ่งหัวข้อสำคัญในบทนี้คือการประยุกต์ใช้แนวทาง **BEM (Block Element Modifier)** ร่วมกับ Tailwind CSS แม้ Tailwind จะเน้น utility-first แต่การใช้โครงสร้างของ BEM ร่วมด้วยจะช่วย

ให้โครงสร้าง class มีระบบแบบลำดับชั้นที่ชัดเจน โดยเฉพาะในระบบขนาดใหญ่ที่มีหลาย component ซ้อนกัน การตั้งชื่อ class ตามรูปแบบ BEM ช่วยลดความสับสนและเพิ่มความสามารถในการจัดการ component ได้อย่างมีประสิทธิภาพ

นอกจากนั้น บทนี้ยังกล่าวถึงแนวทางการเขียน CSS แบบ **Component-based** โดยเน้นหลักการ DRY (Don't Repeat Yourself) ซึ่งส่งผลให้การพัฒนา UI มีความยั่งยืน เมื่อ UI component ใด ๆ เปลี่ยนแปลง นักพัฒนาสามารถปรับที่จุดเดียวและสะท้อนผลได้ในทุกที่ที่ใช้งาน ส่งผลให้ทีมสามารถทำงานร่วมกันได้ดีขึ้น และลดเวลาที่ใช้ในการบำรุงรักษาโค้ดในอนาคต

เพื่อเสริมประสิทธิภาพการจัดการ CSS บทนี้ยังแนะนำการใช้ **Preprocessor อย่าง SCSS** ร่วมกับ Tailwind CSS ซึ่งเปิดทางให้สามารถใช้ mixins, variables, nesting และฟังก์ชันต่างๆ ที่ SCSS มีให้ โดยไม่ละทิ้งแนวทาง utility-first ของ Tailwind ทั้งนี้ การใช้ SCSS อย่างเหมาะสมจะช่วยเพิ่มความยืดหยุ่นในการเขียนโค้ดและขยายความสามารถของระบบ component ให้ทรงพลังมากยิ่งขึ้น

บทที่ 11 นี้จึงเหมาะอย่างยิ่งสำหรับนักพัฒนาเว็บที่ต้องการยกระดับแนวทางการจัดการ CSS ให้มีความสามารถในการขยายตัว รองรับการทำงานเป็นทีม และปรับเปลี่ยนได้ง่ายในระยะยาว โดยอาศัยแนวคิดเชิงระบบ การออกแบบ component อย่างชาญฉลาด และการผสานพลังของ Tailwind CSS กับเครื่องมือทันสมัยอย่าง SCSS เพื่อสร้างระบบที่ทั้งยืดหยุ่นและยั่งยืนในโลกแห่งการพัฒนาเว็บยุคใหม่

Reusable Component Patterns

□ **หัวข้อหลัก**

1. **การใช้ @apply ในไฟล์ CSS**
2. **สร้าง class แบบ Custom Utility**
3. **โครงสร้าง BEM + Tailwind**
4. **แนวทางเขียน component-based CSS แบบ DRY**
5. **การใช้ Preprocessor เช่น SCSS + Tailwind**

---

1. □ การใช้ @apply ในไฟล์ CSS

@apply เป็นฟีเจอร์ของ Tailwind ที่ช่วยให้เรานำ utility class หลายๆ ตัวมารวมเป็นหนึ่งเดียว แล้วนำไปใช้ซ้ำได้

□ **ตัวอย่าง**

```
/* styles.css */
.btn {
  @apply bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600;
```

```
}
<button class="btn">Click Me</button>
```

- □ **ข้อดี:** ลดซ้ำซ้อนใน HTML
- ⚠□ **ข้อควรระวัง:** อย่าใช้กับ pseudo เช่น focus: โดยตรงใน CSS (ใช้ใน HTML เท่านั้น)

---

2. □ สร้าง Custom Utility Class

สามารถสร้าง utility class ใหม่ใน tailwind.config.js เพื่อให้โค้ดอ่านง่ายขึ้น และควบคุมได้จากศูนย์กลาง

□ ตัวอย่าง

```
// tailwind.config.js
module.exports = {
  content: ['./index.html'],
  theme: {
    extend: {
      spacing: {
        'fluid': 'clamp(1rem, 2vw, 3rem)'
      },
      colors: {
        'brand-blue': '#2563eb',
      },
    },
  },
  plugins: [],
}
<div class="p-fluid text-brand-blue">Custom Utility</div>
```

---

3. □ โครงสร้าง BEM + Tailwind

BEM (Block Element Modifier) เป็นโครงสร้างการตั้งชื่อ class ที่ช่วยให้จัดการ CSS ได้ง่ายขึ้นในระบบ component-based

□ ตัวอย่าง

```
<div class="card">
  <h2 class="card__title">Title</h2>
  <p class="card__content">Lorem ipsum dolor sit amet...</p>
</div>
```

```
/* styles.css */
.card {
  @apply bg-white shadow-md p-4 rounded-lg;
}
.card__title {
  @apply text-xl font-bold text-gray-800;
}
.card__content {
  @apply text-sm text-gray-600;
}
```

- □ **ประโยชน์:** ใช้ร่วมกับ Tailwind เพื่อความชัดเจนของ component และจัดการซับซ้อนได้ดี

---

4. ♻□ เขียน Component-based CSS แบบ DRY (Don't Repeat Yourself)

□ **เทคนิค:**

- สร้าง partial component เช่น .btn, .input, .card
- ใช้ @apply รวม class เด่น ๆ เข้าไว้
- ใช้ @layer components เพื่อแยกโซนใน CSS ให้ Tailwind จัดเรียงถูกต้อง

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

@layer components {
  .btn {
    @apply px-4 py-2 font-semibold rounded text-white bg-blue-500 hover:bg-blue-600;
  }
  .input {
    @apply border px-3 py-2 rounded w-full focus:outline-none focus:ring-2 focus:ring-blue-300;
  }
}
<button class="btn">Submit</button>
<input type="text" class="input" placeholder="Enter something">
```

---

5. ⚙□ การใช้ Preprocessor เช่น SCSS + Tailwind

SCSS ช่วยให้เราใช้ตัวแปร, nesting, mixin, และ loop ได้ ทำงานร่วมกับ Tailwind ได้โดยใช้โครงสร้างแบบ hybrid

□ ตัวอย่างการรวม SCSS กับ Tailwind

```
// styles.scss
@tailwind base;
@tailwind components;
@tailwind utilities;

$primary: #4f46e5;

.btn {
  @apply px-4 py-2 rounded text-white font-medium;
  background-color: $primary;

  &:hover {
    background-color: darken($primary, 10%);
  }
}
```

ใช้งาน SCSS ต้องคอมไพล์ผ่าน PostCSS หรือ SCSS compiler เช่น sass styles.scss styles.css

---

□ สรุปเทคนิคสำคัญ

เทคนิค	จุดเด่น	เหมาะกับ
@apply	รวม utility class เพื่อใช้งานซ้ำ	ปรับปรุงความสะอาดของ HTML
Custom Utility	เพิ่ม class เฉพาะองค์กร	ใช้ในหลาย component
BEM + Tailwind	สื่อความหมาย component ชัด	ทีม dev ที่ทำงานร่วมกัน
@layer components	สร้าง reusable component	ใช้ในโปรเจกต์ขนาดกลาง-ใหญ่
SCSS Integration	ใช้ logic + style ร่วมกัน	ทีมที่ใช้ SCSS เป็นหลัก

Reusable Component Patterns (รายละเอียดเชิงลึก)

---

1. □ การใช้ @apply ในไฟล์ CSS

□ **หลักการ:**

- Tailwind CSS ช่วยให้เขียน HTML ได้รวดเร็ว แต่เมื่อ component มี class ซ้ำกันหลายครั้ง (เช่น bg-blue-500 text-white py-2 px-4 rounded) โค้ดจะซ้ำซ้อนและดูรก
- @apply ช่วยรวบรวม class หลายตัวให้กลายเป็นหนึ่ง class เพื่อใช้ซ้ำใน component อื่น

□ ตัวอย่างโค้ด:

```
/* styles.css */
.btn-primary {
  @apply bg-blue-600 text-white font-medium px-4 py-2 rounded hover:bg-blue-700;
}
.btn-secondary {
  @apply bg-gray-200 text-gray-800 font-medium px-4 py-2 rounded hover:bg-gray-300;
}
<button class="btn-primary">Submit</button>
<button class="btn-secondary">Cancel</button>
```

□ ข้อดี:

- ลดความซ้ำซ้อน
- โค้ด HTML อ่านง่าย
- แก้ class ได้จากศูนย์กลาง

⚠□ ข้อควรระวัง:

- ใช้ @apply ได้เฉพาะในไฟล์ CSS ที่รองรับ PostCSS (Tailwind จัดการให้)
- ไม่สามารถใช้กับ class ที่มี pseudo-element เช่น hover:bg-blue-600 นอก context @layer

---

2. □ สร้าง Custom Utility Class

□ หลักการ:

สามารถกำหนด custom utility class ของเราเองผ่าน extend ใน tailwind.config.js เช่น padding, margin, color, spacing, หรือ breakpoint ใหม่ ๆ

□ ตัวอย่างโค้ด:

```
// tailwind.config.js
module.exports = {
  theme: {
    extend: {
      spacing: {
        'fluid': 'clamp(1rem, 5vw, 4rem)', // responsive padding
      },
      colors: {
```

```
        brand: {
          light: '#93c5fd',
          DEFAULT: '#3b82f6',
          dark: '#1e40af',
        },
      },
    },
  },
}
<div class="p-fluid bg-brand-dark text-white">Custom Utility</div>
```

ประโยชน์:

- ใช้ชื่อ class ที่อ่านง่าย
- มีการควบคุม consistent ของดีไซน์ทั้งระบบ
- เพิ่ม utility ใหม่ที่ไม่มีใน Tailwind ได้

---

3. โครงสร้าง BEM + Tailwind

หลักการ:

- BEM = Block__Element--Modifier
- เมื่อใช้ Tailwind ควบคู่ BEM จะได้ชื่อ class ที่สื่อถึงโครงสร้าง component ชัดเจน

ตัวอย่าง:

```
<div class="card">
  <h2 class="card__title">Product Title</h2>
  <p class="card__description">Lorem ipsum dolor sit amet</p>
  <button class="card__btn">Buy</button>
</div>
/* styles.css */
.card {
  @apply p-6 bg-white rounded-lg shadow-md;
}
.card__title {
  @apply text-xl font-bold text-gray-800;
}
.card__description {
  @apply text-gray-600;
```

```
}
.card__btn {
  @apply mt-4 bg-blue-500 text-white py-2 px-4 rounded hover:bg-blue-600;
}
```

ประโยชน์:

- เพิ่ม modularity และ readability
- เหมาะกับทีมพัฒนาแบบแยกหน้าที่ Frontend/Design
- สามารถขยาย component แบบไม่สับสน

---

4. ♻ เขียน CSS แบบ DRY (Don't Repeat Yourself)

แนวคิด:

- อย่าเขียน class ซ้ำ ๆ หลายจุดใน HTML
- สร้าง class reusable เช่น .btn, .input, .card, .avatar

ตัวอย่าง:

```
@layer components {
  .btn {
    @apply inline-block px-4 py-2 text-sm font-medium text-white bg-blue-500 rounded hover:bg-blue-600;
  }

  .input {
    @apply w-full px-3 py-2 border border-gray-300 rounded focus:outline-none focus:ring-2 focus:ring-blue-400;
  }
}
<input class="input" placeholder="Enter your email" />
<button class="btn">Submit</button>
```

เทคนิค:

- จัดกลุ่ม reusable class ไว้ใน @layer components เพื่อให้ Tailwind ควบคุม priority ได้
- อย่าใช้ @apply กับ @responsive, @variants หรือ @screen ซ้อนกันโดยตรง

---

5. ⚙ การใช้ SCSS + Tailwind

ทำไมต้อง SCSS:

- ใช้ตัวแปร ($primary-color)

- Mixin และฟังก์ชันซ้ำได้
- Nesting ที่ Tailwind ไม่รองรับโดยตรง

☐ **ตัวอย่าง SCSS + Tailwind:**

```
// styles.scss
@tailwind base;
@tailwind components;
@tailwind utilities;

$primary: #4f46e5;

.card {
  @apply p-6 rounded shadow-lg bg-white;

  &__title {
    @apply text-lg font-bold text-gray-900;
  }

  &__button {
    @apply mt-4 px-4 py-2 rounded text-white;
    background-color: $primary;

    &:hover {
      background-color: darken($primary, 10%);
    }
  }
}
```

ต้องคอมไพล์ SCSS เป็น CSS ด้วย PostCSS หรือใช้ sass CLI เช่น:

```
sass styles.scss styles.css
```

☐ **ข้อดี:**

- เขียน CSS ที่ยืดหยุ่นและ maintain ได้ดีขึ้น
- รวมพลัง SCSS + Tailwind เพื่อใช้ logic เช่น loop, condition, nesting

---

☐ **สรุป Best Practices**

เทคนิค	ใช้เมื่อ	สิ่งที่ควรระวัง

เทคนิค	ใช้เมื่อ	สิ่งที่ควรระวัง
@apply	class utility ซ้ำบ่อย	หลีกเลี่ยง pseudo ใน @apply
Custom Utility	ต้องการใช้ class เฉพาะองค์กร	อย่าขัดกับ core ของ Tailwind
BEM	เขียน CSS แบบ component-based	ต้องตั้งชื่อให้ชัดเจน
DRY via @layer	ลดความซ้ำในโค้ด HTML	อย่าเขียน utility ซ้อน
SCSS + Tailwind	ต้องการ logic และ nesting	ใช้ build tools เพิ่มเติม

ตัวอย่าง **โครงสร้างโปรเจกต์จริง**, **โปรแกรมแนวประยุกต์** หรือ **โปรแกรมบูรณาการ** ที่ใช้แนวทางในบทนี้ เช่น:

- **ระบบปุ่ม + ฟอร์ม reusable ทั้งชุด**
- **Reusable Card component ที่ปรับขนาดและ theme ได้**
- **ระบบ SCSS component แบบ module-based**

ต่อไปนี้คือรายละเอียดเจาะลึกเกี่ยวกับ **การใช้ @apply ในไฟล์ CSS** พร้อมคำอธิบายแบบเป็นระบบ ตัวอย่างโค้ด โครงสร้างไฟล์ และผลการรัน:

การใช้ @apply ในไฟล์ CSS

□ คืออะไร?

@apply คือคำสั่งของ Tailwind CSS ที่ใช้รวมหลาย utility class เข้าด้วยกันภายใน custom class ที่กำหนดไว้ใน CSS เพื่อทำให้โค้ด HTML อ่านง่ายและไม่ซ้ำซ้อน

□ เหมาะกับสถานการณ์ใด?

- เมื่อมี class ซ้ำกันในหลาย component เช่น ปุ่ม, การ์ด, อินพุต ฯลฯ
- ต้องการเขียน CSS แบบ component-based
- ต้องการควบคุมจากศูนย์กลางแบบ DRY (Don't Repeat Yourself)

□ โครงสร้างโปรเจกต์พื้นฐาน

```
tailwind-apply-example/
├── index.html
├── styles/
│   └── main.css
└── tailwind.config.js
```

```
├── postcss.config.js
└── package.json
```

ไฟล์ที่เกี่ยวข้อง

1. tailwind.config.js

```
module.exports = {
  content: ["./index.html"],
  theme: {
    extend: {},
  },
  plugins: [],
};
```

---

2. postcss.config.js

```
module.exports = {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
};
```

---

3. styles/main.css

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

/* Reusable components using @apply */
.btn {
  @apply px-4 py-2 rounded text-white font-semibold;
}

.btn-primary {
  @apply btn bg-blue-600 hover:bg-blue-700;
}
```

```
.btn-secondary {
  @apply btn bg-gray-300 text-gray-800 hover:bg-gray-400;
}
```

---

4. index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@apply Example</title>
  <link href="./styles/main.css" rel="stylesheet" />
</head>
<body class="p-10 bg-gray-50 space-y-4">
  <button class="btn-primary">Primary Button</button>
  <button class="btn-secondary">Secondary Button</button>
</body>
</html>
```

ผลการรัน

เมื่อเปิด index.html:

- ปุ่ม “Primary Button” เป็นสีฟ้าเข้ม (bg-blue-600) เมื่อ hover จะเปลี่ยนเป็นฟ้าเข้มกว่า (bg-blue-700)
- ปุ่ม “Secondary Button” เป็นสีเทา (bg-gray-300) พร้อม text สีเทาเข้ม (text-gray-800) และเปลี่ยนสีเมื่อ hover

---

อธิบายแนวคิด

```
.btn {
  @apply px-4 py-2 rounded text-white font-semibold;
}
```

- .btn เป็นพื้นฐานสำหรับปุ่มทุกรูปแบบ
- .btn-primary, .btn-secondary ใช้ .btn และเติมสีหรือ effect เพิ่มเข้าไป
- ช่วยลดความซ้ำของ class utility และทำให้ style เปลี่ยนง่ายในภายหลัง

☐ ข้อดีของการใช้ @apply

ข้อดี	รายละเอียด
ลดความซ้ำซ้อน	ไม่ต้องเขียน class ยาว ๆ ซ้ำไปซ้ำมา
ง่ายต่อการปรับแก้	เปลี่ยน style ได้จากไฟล์เดียว
สื่อถึงความตั้งใจของ component	เช่น .btn-primary vs .btn-secondary ชัดเจนว่าใช้กับปุ่มใด

⚠☐ ข้อควรระวัง

- ใช้ได้เฉพาะในไฟล์ CSS ที่ Tailwind ประมวลผล (เช่น @tailwind, PostCSS)
- ใช้กับ pseudo-class เช่น hover: ได้ แต่ต้องอยู่ใน @layer components
- อย่าใช้กับ responsive class (md:) ตรง ๆ ภายใน @apply

☐ สรุป

- ใช้ @apply เพื่อเขียน component CSS แบบ reusable
- เหมาะกับการสร้าง UI แบบ modular
- ช่วยให้โค้ดสะอาดและขยายได้ง่าย

ต่อไปนี้คือตัวอย่างโปรแกรม **แบบเต็มไฟล์** พร้อม **โครงสร้าง + คำอธิบายโค้ด + ผลการรัน** จำนวน 3 ตัวอย่าง สำหรับหัวข้อ:

การใช้ @apply ในไฟล์ CSS (Basic Examples)

☐ ตัวอย่างที่ 1: ปุ่มแบบ Reusable ด้วย @apply

☐ โครงสร้างไฟล์

```
tailwind-apply-button/
├── index.html
├── styles/
│   └── main.css
├── tailwind.config.js
└── postcss.config.js
```

☐ styles/main.css

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

```
.btn {
  @apply bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded;
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <link href="./styles/main.css" rel="stylesheet">
  <title>Tailwind Apply Button</title>
</head>
<body class="flex items-center justify-center min-h-screen bg-gray-100">
  <button class="btn">Click Me</button>
</body>
</html>
```

ผลการรัน

ปุ่ม “Click Me” สีฟ้า (bg-blue-500), มี hover สีเข้มขึ้น (bg-blue-700) ใช้ .btn class ที่ reuse ได้

ตัวอย่างที่ 2: Card แบบ Custom Class ด้วย @apply

โครงสร้างไฟล์

```
tailwind-apply-card/
├── index.html
├── styles/
│   └── main.css
├── tailwind.config.js
└── postcss.config.js
```

styles/main.css

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

.card {
```

```
  @apply bg-white rounded-lg shadow-md p-6 max-w-sm;
}
.card-title {
  @apply text-xl font-semibold text-gray-800 mb-2;
}
.card-desc {
  @apply text-gray-600;
}
```

index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="./styles/main.css">
</head>
<body class="flex items-center justify-center min-h-screen bg-gray-200">
  <div class="card">
    <div class="card-title">Card Title</div>
    <div class="card-desc">This is a reusable card component using @apply.</div>
  </div>
</body>
</html>
```

ผลการรัน

Card สีขาว ขอบมน เงาเบา มีหัวข้อและคำอธิบาย ใช้ class .card, .card-title, .card-desc ที่จัดกลุ่ม style ด้วย @apply

---

ตัวอย่างที่ 3: Form Layout Reusable ด้วย @apply

โครงสร้างไฟล์

```
tailwind-apply-form/
├── index.html
├── styles/
│   └── main.css
├── tailwind.config.js
└── postcss.config.js
```

styles/main.css

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

.input-field {
  @apply border border-gray-300 rounded px-3 py-2 w-full focus:outline-none focus:ring-2
focus:ring-blue-500;
}
.label {
  @apply block text-gray-700 font-medium mb-1;
}
.form-group {
  @apply mb-4;
}
```

index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="./styles/main.css">
</head>
<body class="flex items-center justify-center min-h-screen bg-gray-100">
  <form class="bg-white p-6 rounded shadow-md w-96">
    <div class="form-group">
      <label class="label" for="email">Email</label>
      <input class="input-field" type="email" id="email" />
    </div>
    <div class="form-group">
      <label class="label" for="password">Password</label>
      <input class="input-field" type="password" id="password" />
    </div>
    <button class="btn">Login</button>
  </form>
```

```
</body>
</html>
```

- เพิ่ม .btn class แบบตัวอย่างแรกไว้ใน main.css ด้วยก็จะ reuse ปุ่มได้ทันที
- **ผลการรัน**

Form แบบสะอาดตา มี input ที่จัด style ไว้ด้วย .input-field สามารถ reuse ได้หลายฟอร์ม

ต่อไปนี้คือตัวอย่างโปรแกรม **แนวประยุกต์** และ **บูรณาการ** ที่ใช้ @apply ร่วมกับ Framework ยอดนิยม เช่น Vue, React และ Laravel Blade

---

ตัวอย่างที่ 1: Vue 3 + Tailwind + @apply (ปุ่ม reusable)

โครงสร้างไฟล์ (เฉพาะไฟล์สำคัญ)

```
vue-tailwind-apply/
├── src/
│   ├── App.vue
│   ├── components/
│   │   └── BaseButton.vue
│   └── assets/
│       └── main.css
├── tailwind.config.js
├── postcss.config.js
└── package.json
```

assets/main.css

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

.btn {
  @apply bg-indigo-600 text-white font-semibold px-4 py-2 rounded hover:bg-indigo-700;
}
```

components/BaseButton.vue

```
<template>
  <button class="btn" @click="$emit('click')">
    <slot />
```

```
  </button>
</template>

<script>
export default {
  name: 'BaseButton',
};
</script>
```

App.vue

```
<template>
  <div class="p-6">
    <BaseButton @click="clicked">Click me</BaseButton>
  </div>
</template>

<script>
import BaseButton from './components/BaseButton.vue';

export default {
  components: { BaseButton },
  methods: {
    clicked() {
      alert('Button clicked!');
    }
  }
};
</script>

<style src="./assets/main.css"></style>
```

ตัวอย่างที่ 2: React + Tailwind + @apply (Reusable Card Component)

โครงสร้างไฟล์ (สำคัญ)

```
react-tailwind-apply/
├── src/
```

```
│   ├── App.js
│   ├── components/
│   │   └── Card.js
│   └── styles/
│       └── main.css
├── tailwind.config.js
├── postcss.config.js
└── package.json
```

styles/main.css

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

.card {
  @apply p-6 rounded-lg shadow-lg bg-white max-w-sm mx-auto;
}

.card-title {
  @apply text-2xl font-bold mb-2 text-gray-800;
}

.card-body {
  @apply text-gray-600;
}
```

components/Card.js

```
import React from 'react';
import '../styles/main.css';

export default function Card({ title, children }) {
  return (
    <div className="card">
      <h2 className="card-title">{title}</h2>
      <div className="card-body">{children}</div>
    </div>
```

```
  );
}
```

App.js

```
import React from 'react';
import Card from './components/Card';

export default function App() {
  return (
    <div className="min-h-screen bg-gray-100 flex items-center justify-center p-6">
      <Card title="Tailwind @apply Card">
        This card component uses reusable styles via @apply in Tailwind CSS.
      </Card>
    </div>
  );
}
```

ตัวอย่างที่ 3: Laravel Blade + Tailwind + @apply (Form Component)

โครงสร้างไฟล์

```
laravel-tailwind-apply/
├── resources/
│   ├── views/
│   │   ├── components/
│   │   │   └── button.blade.php
│   │   └── welcome.blade.php
│   └── css/
│       └── app.css
├── tailwind.config.js
└── package.json
```

resources/css/app.css

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

.btn {
```

```
  @apply bg-green-600 text-white font-semibold py-2 px-4 rounded hover:bg-green-700;
}
```

resources/views/components/button.blade.php

```
<button {{ $attributes->merge(['class' => 'btn']) }}>
  {{ $slot }}
</button>
```

resources/views/welcome.blade.php

```
<x-layout>
  <form class="max-w-md mx-auto mt-10 p-6 bg-white rounded shadow">
    <label for="email" class="block mb-2 font-medium">Email</label>
    <input type="email" id="email" name="email" class="border border-gray-300 rounded px-3 py-2 w-full mb-4 focus:outline-none focus:ring-2 focus:ring-green-400" />

    <x-button>Submit</x-button>
  </form>
</x-layout>
```

สรุป

Framework	ตัวอย่างการใช้	ประโยชน์หลักจาก @apply
Vue 3	ปุ่ม reusable	ลดซ้ำซ้อน สร้าง component ที่อ่านง่าย
React	Card component	สร้าง UI reusable ง่ายด้วย CSS class กลาง
Laravel Blade	ปุ่ม Blade Component	ใช้ @apply ควบคุม style ปุ่มได้แบบ DRY

ต่อไปนี้คือคำอธิบายและตัวอย่างเชิงลึกเกี่ยวกับ **การสร้าง class แบบ custom utility** ใน Tailwind CSS รวมทั้งตัวอย่างโปรแกรมเต็มไฟล์ + โครงสร้าง + คำอธิบายโค้ด + ผลการรัน

การสร้าง Custom Utility Class ใน Tailwind CSS

---

1. คืออะไร?

- Custom utility คือการเพิ่ม class ใหม่ ๆ ที่ไม่อยู่ใน Tailwind เอง โดยใช้วิธีเพิ่มในไฟล์ CSS หรือใน tailwind.config.js
- ช่วยให้เราเพิ่ม style เฉพาะที่ต้องการใช้งานบ่อย ๆ ลงใน Tailwind อย่างมีระบบ

---

2. วิธีสร้าง Custom Utility

มี 2 วิธีหลัก:

(1) สร้างในไฟล์ CSS ด้วย @layer utilities + @apply (แนะนำ)

```
@layer utilities {
  .text-shadow {
    text-shadow: 2px 2px 4px rgba(0,0,0,0.3);
  }
}
```

(2) สร้างด้วย Tailwind Plugin ใน tailwind.config.js

```
module.exports = {
  // ...
  plugins: [
    function({ addUtilities }) {
      const newUtilities = {
        '.text-shadow': {
          textShadow: '2px 2px 4px rgba(0,0,0,0.3)',
        },
      }
      addUtilities(newUtilities, ['responsive', 'hover'])
    }
  ]
}
```

ตัวอย่างโปรแกรมเต็มไฟล์

ตัวอย่างที่ 1: Custom Utility text-shadow ด้วย @layer utilities ใน CSS

โครงสร้างไฟล์

```
custom-utility-text-shadow/
├── index.html
├── styles/
│   └── main.css
├── tailwind.config.js
└── postcss.config.js
```

ไฟล์ styles/main.css

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

/* สร้าง custom utility */
@layer utilities {
  .text-shadow {
    text-shadow: 2px 2px 4px rgba(0,0,0,0.3);
  }
}
```

ไฟล์ index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Custom Utility Example</title>
  <link href="./styles/main.css" rel="stylesheet" />
</head>
<body class="bg-gray-100 flex items-center justify-center min-h-screen">
  <h1 class="text-4xl font-bold text-blue-600 text-shadow">
    Text with Shadow Effect
  </h1>
</body>
</html>
```

ผลการรัน

- ตัวอักษรใหญ่สีน้ำเงินมีเงาเบา ๆ (text-shadow) ซึ่งได้จาก class .text-shadow ที่สร้างขึ้นเอง

ตัวอย่างที่ 2: Custom Utility สำหรับ Animation ด้วย Plugin ใน tailwind.config.js

โครงสร้างไฟล์

```
custom-utility-animation/
├── index.html
├── styles/
│   └── main.css
├── tailwind.config.js
```

```
└── postcss.config.js
```

ไฟล์ tailwind.config.js

```
module.exports = {
  content: ["./index.html"],
  theme: {
    extend: {},
  },
  plugins: [
    function({ addUtilities }) {
      const newUtilities = {
        '.animate-spin-slow': {
          animation: 'spin 3s linear infinite',
        },
      }
      addUtilities(newUtilities, ['responsive', 'hover']);
    }
  ]
}
```

ไฟล์ styles/main.css

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

ไฟล์ index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Custom Animation Utility</title>
  <link href="./styles/main.css" rel="stylesheet" />
</head>
<body class="bg-gray-50 flex items-center justify-center min-h-screen">
  <div class="w-16 h-16 border-4 border-blue-600 border-t-transparent rounded-full animate-
spin-slow"></div>
</body>
```

```
</html>
```

ผลการรัน

- วงกลมหมุนช้า ๆ (3 วินาทีต่อรอบ) ใช้ class .animate-spin-slow ที่เราสร้างเอง

ตัวอย่างที่ 3: Custom Utility สำหรับการจัดขอบเงา (box-shadow)

โครงสร้างไฟล์

```
custom-utility-box-shadow/
├── index.html
├── styles/
│   └── main.css
├── tailwind.config.js
└── postcss.config.js
```

ไฟล์ styles/main.css

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

@layer utilities {
  .shadow-3xl {
    box-shadow: 0 20px 25px -5px rgba(0,0,0,0.1), 0 10px 10px -5px rgba(0,0,0,0.04);
  }
}
```

ไฟล์ index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Custom Box Shadow</title>
  <link href="./styles/main.css" rel="stylesheet" />
</head>
<body class="bg-gray-100 flex items-center justify-center min-h-screen">
  <div class="bg-white p-8 rounded shadow-3xl max-w-md text-center">
    <h2 class="text-2xl font-bold mb-2">Custom Box Shadow</h2>
    <p>This box uses a custom shadow utility.</p>
```

```
  </div>
</body>
</html>
```

ผลการรัน

- กล่องสีขาวมีเงาที่ออกแบบเองซึ่งไม่อยู่ใน Tailwind ปกติ แต่สร้างจาก .shadow-3xl

---

สรุป

วิธีสร้าง **Custom Utility**	การใช้งาน	ข้อดี
@layer utilities ใน CSS	ง่าย เร็ว ไม่ต้องแก้ config	เหมาะกับโปรเจกต์เล็ก/กลาง
Plugin ใน tailwind.config.js	รองรับ responsive, pseudo-class	เหมาะกับโปรเจกต์ใหญ่, ปรับแต่งซับซ้อน

ชุดที่ 1: ตัวอย่างแบบเต็มไฟล์ (พื้นฐาน) — 3 โปรแกรม

ตัวอย่าง 1: Text Shadow Utility

โครงสร้างไฟล์

```
custom-utility-basic-1/
├── index.html
├── styles/
│   └── main.css
├── tailwind.config.js
└── postcss.config.js
```

styles/main.css

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

@layer utilities {
  .text-shadow {
    text-shadow: 2px 2px 4px rgba(0,0,0,0.3);
  }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Text Shadow Utility</title>
  <link href="./styles/main.css" rel="stylesheet" />
</head>
<body class="bg-gray-100 flex items-center justify-center min-h-screen">
  <h1 class="text-5xl font-bold text-blue-600 text-shadow">Text with Shadow</h1>
</body>
</html>
```

ผลการรัน:

ข้อความ “Text with Shadow” สีฟ้าขนาดใหญ่ มีเงาเบา ๆ จาก .text-shadow

---

ตัวอย่าง 2: Custom Rounded Corners

styles/main.css

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

@layer utilities {
  .rounded-xl-custom {
    border-radius: 1.5rem;
  }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Custom Rounded Corners</title>
  <link href="./styles/main.css" rel="stylesheet" />
</head>
<body class="bg-gray-200 flex items-center justify-center min-h-screen">
```

```
  <div class="bg-indigo-600 text-white p-8 rounded-xl-custom shadow-lg max-w-sm text-center">
    Custom Rounded Corners
  </div>
</body>
</html>
```

ผลการรัน:

กล่องสีม่วงมีขอบมนใหญ่กว่าปกติ (1.5rem) ด้วย .rounded-xl-custom

---

ตัวอย่าง 3: Custom Box Shadow

styles/main.css

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

@layer utilities {
  .shadow-3xl {
    box-shadow: 0 20px 25px -5px rgba(0,0,0,0.1),
                0 10px 10px -5px rgba(0,0,0,0.04);
  }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Custom Shadow</title>
  <link href="./styles/main.css" rel="stylesheet" />
</head>
<body class="bg-gray-100 flex items-center justify-center min-h-screen">
  <div class="bg-white p-8 rounded shadow-3xl max-w-md text-center">
    Custom Box Shadow Utility
  </div>
</body>
</html>
```

ผลการรัน:

กล่องสีขาวมีเงาแบบซับซ้อนมากกว่าปกติ ด้วย .shadow-3xl

ชุดที่ 2: ตัวอย่างแนวประยุกต์ (Vue, React, Laravel) — 3 โปรแกรม

ตัวอย่าง 1: Vue 3 - Custom Utility text-shadow

โครงสร้าง

```
vue-custom-utility/
├── src/
│   ├── App.vue
│   └── assets/
│       └── main.css
├── tailwind.config.js
└── postcss.config.js
```

assets/main.css

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

@layer utilities {
  .text-shadow {
    text-shadow: 2px 2px 6px rgba(0,0,0,0.4);
  }
}
```

src/App.vue

```
<template>
  <div class="p-6">
    <h1 class="text-4xl font-bold text-red-600 text-shadow">Vue Custom Utility</h1>
  </div>
</template>

<script>
export default {
  name: 'App',
```

```
};
</script>

<style src="./assets/main.css"></style>
```

ตัวอย่าง 2: React - Custom Utility shadow-3xl

โครงสร้าง

```
react-custom-utility/
├── src/
│   ├── App.js
│   └── styles/
│       └── main.css
├── tailwind.config.js
└── postcss.config.js
```

styles/main.css

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

@layer utilities {
  .shadow-3xl {
    box-shadow: 0 25px 30px -10px rgba(0,0,0,0.15),
                0 15px 20px -10px rgba(0,0,0,0.05);
  }
}
```

src/App.js

```
import React from "react";
import "./styles/main.css";

export default function App() {
  return (
    <div className="bg-gray-100 min-h-screen flex items-center justify-center p-6">
      <div className="bg-white p-8 rounded shadow-3xl max-w-md text-center">
        React Custom Shadow Utility
```