ChatGPT

Tailwind CSS UI Libraries & Templates Guide (2025)

Component Libraries (Tailwind CSS Ecosystem)

Tailwind's ecosystem in 2025 offers a rich variety of UI component libraries, each with distinct approaches. Key trade-offs include **headless vs. pre-styled components**, installation method (copy-paste snippets, CLI tools, or NPM packages), and features like theming and dark mode. Accessibility and developer experience remain paramount across all popular libraries [1]. Below is a comparison of major Tailwind-centric libraries:

Library	License	Approach	Integration	Theming & Dark Mode	Ideal Use Cases
Tailwind UI (Tailwind Labs)	Closed-source (paid) [2]	Pre-designed components styled with Tailwind; uses Headless UI for interactivity	Copy-paste code (available in React, Vue & HTML variants)	Light and dark themes included (no additional themes) [3]	High-end **marketing sites** and polished **web apps** where design quality is critical and budget allows a premium kit.
daisyUI (open-source)	MIT [4] [5]	Pre-styled component classes via a Tailwind plugin (e.g. `btn`, `card`)	NPM package (Tailwind plugin); then use semantic classes in HTML	**35+ built-in themes** using CSS variables (light, dark, and more) [6]; easily switchable at runtime	Rapid prototyping or **apps needing quick theming**. Great for **any framework** (no JS needed) when you want a Bootstrap-like developer experience in Tailwind [7].

Library	License	Approach	Integration	Theming & Dark Mode	Ideal Use Cases
Flowbite (open-source core)	MIT [2] (Pro version available)	Pre-styled components with Tailwind classes; includes JS for interactive elements	NPM package or CDN – includes CSS + JS; also offers a React wrapper package	Light and dark mode support out-of-the-box [3] (no multi-theme beyond dark/light)	**General-purpose UI** for websites and web apps. Good for **forms, modals, navbars,** etc., especially if you want ready-made components working in plain HTML or with React. Comes with a Figma design kit for designers [8] [9].
shadcn/ui (open-source)	MIT [10]	Unstyled **headless components** (built on Radix UI) with Tailwind-based JSX templates provided. You copy these into your project for full control [7] [11].	CLI tool (`npx shadcn-ui`) to **add components** (React-only) which pulls in Radix libraries and Tailwind classes [12]. Code lives in your repo.	Default light & dark styling baked in for components (since templates include both), but no elaborate theme system (you can customize Tailwind config or class names for more themes).	**React** projects that need **maximum customization**. Ideal for design systems – you get accessible, well-structured components (Radix UI under the hood) but can restyle and extend them at will [13] [14]. Note: pulling many components can add numerous dependencies (each Radix module, icons, etc.) [15].

Library	License	Approach	Integration	Theming & Dark Mode	Ideal Use Cases
Radix UI (Primitives)	MIT	**Headless UI primitives** – accessible low-level components (e.g. modal, dropdown, tabs) with no default styling [1].	NPM packages (each primitive is a React package) – you compose these with your own Tailwind classes or CSS.	No built-in styling or themes (you implement the design). Pairs well with Tailwind for styling.	When building a design system or custom UI **from scratch in React**. Radix ensures accessibility and robust logic, and you layer Tailwind styles on top. Often used together with shadcn's templates or custom components for full control.
Headless UI (Tailwind Labs)	MIT	**Unstyled, accessible components** (menu, listbox, modal, etc.) for React and Vue [16]. Similar to Radix but smaller scope.	NPM package (`@headlessui/react` or Vue) – use the component logic and structure, add Tailwind classes for look and feel.	No themes or styles included by default. Supports dark mode by design (you add classes conditionally).	**Tailwind projects needing basic interactive UI** (dropdowns, dialogs, popovers) without adopting a full styled library. Pairs nicely with Tailwind UI (which uses Headless UI under the hood) [17].

Library	License	Approach	Integration	Theming & Dark Mode	Ideal Use Cases
Tremor (open-source)	MIT	Pre-styled **data visualization components** (charts, KPI cards, layouts) built with Tailwind. Focused on dashboards.	NPM package (`tremor` for React) – provides ready-made chart components and dashboard widgets.	Light & dark mode supported (Tailwind classes and default themes for charts).	**Data-heavy apps and admin dashboards**. Tremor provides out-of-the-box charts (bar, line, etc.) and data widgets, speeding up **analytics dashboard** development [18] [19].

Notes on Trade-offs: Headless libraries like Radix and Headless UI offer maximum flexibility and ensure you meet accessibility standards, but they require more work – you must write the Tailwind classes or custom CSS for styling. Styled libraries (like Flowbite, DaisyUI) let you move faster with pre-made designs, but you may be constrained by their look (though DaisyUI's extensive themes and utility classes help mitigate this by allowing quick theme swaps [20] [14]). Some libraries ship JavaScript for interactive components – for example, Flowbite's bundle is ~132 KB minified [21], whereas DaisyUI adds **no JS** (pure CSS components). If bundle size and dependency count are concerns, note that shadcn's approach, while flexible, can introduce a lot of packages (on install it can bring ~150+ dependencies including all of Radix, icons, etc.) [15]. Always consider your project's needs: a marketing site might favor fully styled components for speed, while a complex web app might benefit from headless components with a custom design system.

shadcn/ui vs. traditional libraries

A notable newcomer is **shadcn/ui**, which is more of a template repository than a library. It provides copy-paste **source code** for components (built with Radix + Tailwind) instead of a compiled package, meaning *you own the code*. The advantage is unlimited customization and no runtime dependency — the components are part of your codebase [7]. This differs from say, Flowbite or Tailwind UI, where you use what's provided and override if needed. The trade-off is maintenance: with shadcn.ui you're responsible for updating components when underlying libraries change. It's best for React teams that want an internal design system without writing everything from scratch. The design is modern and consistent (uses Radix for logic and accessibility, and a clean aesthetic), suitable for product UIs and dashboards. However, it's tied to React and requires Tailwind knowledge to customize deeply [12].

Tailwind UI (with Headless UI)

Tailwind UI remains a gold standard for production-ready Tailwind components. It's a paid collection by Tailwind Labs, offering polished UI snippets for everything from navbars and modals to complex marketing sections. Tailwind UI emphasizes **design quality and consistency** – each component is meticulously designed and responsive. Many components that require interactivity (like drop-down menus or modals) leverage the open-source **Headless UI** library behind the scenes [17], meaning you still need to include

Headless UI (free) for those parts to work. Tailwind UI components are provided in multiple formats (HTML with Alpine.js for minimal JS, React, and Vue). This library doesn't use a theme system beyond light/dark mode, so customization means editing the classes (which is expected in Tailwind). It's well-suited for teams that want to move fast with beautiful UIs *and* are comfortable tweaking Tailwind classes to match their brand. The downside is cost and the fact that it's not open source; you cannot share Tailwind UI code in an open repo. Also, because it's copy-paste, updates to Tailwind UI (new components or improvements) require manually pulling in changes. Despite this, its **breadth of components and page examples** (including marketing pages and application layouts) is unparalleled for Tailwind designs [22] [23].

DaisyUI vs. Flowbite

DaisyUI and Flowbite are often compared as they both provide a large set of pre-styled components for Tailwind. **DaisyUI** is a plugin that magically adds component classes to your Tailwind CSS – for example, using `<button class="btn btn-primary">` gives you a nicely styled button without writing any custom CSS. Under the hood, DaisyUI defines those styles using Tailwind utilities. It includes dozens of components (cards, alerts, navbars, etc.) and a whopping collection of built-in themes [6] that you can toggle via data attributes or class (`data-theme="forest"`, etc.). It's entirely CSS-based (no JS bundled), so components that need interactive behavior (like a modal open/close) rely on simple HTML tricks or require you to add JS (for example, using HTML `<details>` for accordion, or your own small script). DaisyUI's strength is **rapid development**: you get the ease of something like Bootstrap combined with Tailwind's flexibility. It's great for **forms-heavy apps, internal tools, or quick MVPs** where design consistency and speed matter more than completely bespoke style. Because it's framework-agnostic, you can use it anywhere Tailwind is set up, even in a simple static HTML page.

Flowbite, on the other hand, comes with a bit more structure: it has an optional JS bundle for interactive components (dropdowns, modals, carousels, etc.), and also offers a React component library wrapping those (Flowbite React). Flowbite's design language is clean and slightly inspired by Bootstrap and Material design elements. It includes more unique components than Tailwind UI by count (55+ unique components) [24] [25], including things like datepickers, which Tailwind UI doesn't provide. Flowbite can be added via CDN or npm; if you include its CSS, you get utility classes plus component styles. Unlike DaisyUI, Flowbite doesn't give you semantic class names like `btn` – you still use regular Tailwind classes in HTML, or you use their React components. This means a bit more manual work when using plain HTML (you might copy snippets from their docs). Flowbite is **framework-agnostic** in core (works with any frontend, even just HTML) [26]. It also has theming support but not as extensive as DaisyUI – typically just dark mode toggle. Use Flowbite when you want a ready-to-go UI kit with minimal design effort: **dashboards, marketing pages, or even quick prototypes** can benefit. Since it's open-source MIT, it's also safe for commercial projects without license fees [2]. One thing to watch is that including everything can bloat your CSS; you should tree-shake with PurgeCSS (which Tailwind does automatically) and include only the JS you need.

Radix UI vs. Headless UI

Both **Radix UI** and **Headless UI** aim to solve the "logic and accessibility" part of building common components. **Radix UI** (by WorkOS/Modulz) offers a larger set of primitives and is very robust – covering everything from popovers and dropdowns to sliders and toast notifications. It's low-level: you get a structured set of nested components and prop APIs to control behavior, and you bring your own styles. Radix is popular in the React + Tailwind world (shadcn/ui is built on it) because it lets developers avoid reinventing complex a11y logic (e.g., focus trapping in modals, keyboard navigation in menus). **Headless UI**

is a smaller library (by the makers of Tailwind) focusing on a handful of components – mostly menu, listbox (select), combobox (autocomplete), dialog, popover, tabs, accordion. It's a bit higher-level than Radix in some cases (easier to drop in), but far fewer components. If your needs are covered by Headless UI, it's a simpler choice and pairs naturally with Tailwind (Tailwind Labs provides example code for styling each component). For anything outside its list, Radix is the go-to. Radix has more components and tends to receive frequent updates, whereas Headless UI is stable but slower to add new features. Both are **accessible and screen-reader tested**, so you can't go wrong on that front. They also both support React (Headless UI has Vue as well) and are MIT licensed.

Other Notable Libraries

- **HyperUI / Sailboat UI / Lofi UI / etc** – These are collections of **copy-paste Tailwind components** (often just HTML snippets with Tailwind classes) rather than full libraries. For example, **HyperUI** provides dozens of snippets for cards, navbars, etc., with no JS needed [27]. **Sailboat UI** similarly offers 150+ components (uses Alpine.js for any interactivity) [28]. These are fantastic for quickly grabbing a design piece (e.g., a pricing table) and dropping it into your project. They require you to already have Tailwind set up. No runtime overhead aside from what Tailwind itself generates. Use these when you need *just a few specific components* and want to avoid adding dependencies.
- **NextUI (HeroUI)** – NextUI (now rebranded to **HeroUI**) is a React component library that in its latest version is built on Tailwind under the hood [29]. It uses an approach of providing pre-built React components (like buttons, modals, etc.) that are styled with Tailwind utilities (no CSS-in-JS, so zero runtime styling overhead) [29]. It also leverages **React Aria** for accessibility [30]. HeroUI/NextUI includes a theme plugin for Tailwind to easily customize colors and design tokens [31] [32]. This library sits somewhat between a headless and styled library – you use their components and can adjust them via theming, but you don't see Tailwind classes unless you eject. It's a promising solution for React developers who want a **Material-UI-like experience** but with Tailwind's performance and styling paradigm.
- **Material Tailwind & Others** – There are a few Tailwind implementations of other design systems. **Material Tailwind** by Creative Tim is a set of Tailwind components following Google's Material Design (with React and HTML versions). **Mantine** and **Chakra UI** are not Tailwind-based (they have their own styling systems), but some teams use them alongside Tailwind for specific components. However, as of 2025, the trend is toward sticking to pure Tailwind solutions to avoid style conflicts.
- **Accessibility & Dark Mode:** Most modern Tailwind libraries prioritize accessibility. For example, all Radix and Headless components come accessible by default (ARIA attributes, focus management, etc.), and libraries like DaisyUI explicitly mention accessibility best practices. Dark mode support is ubiquitous now – any serious library will have a dark theme or at least guidance. DaisyUI provides many themes including dark [6], Flowbite has dark mode built-in [3], Tailwind UI components include dark variants. When picking a library, ensure it either supports dark mode or you can theme it easily if your app requires it.

Templates & UI Kits (Free & Premium)

Beyond individual components, Tailwind's ecosystem offers **full-page templates and UI kits** to jump-start your design. These range from **free open-source templates** for landing pages and dashboards to premium multi-page kits. When choosing a template, consider the tech stack (plain HTML vs. React/Next.js vs. Astro, etc.), the license/cost, and how well it fits your project's needs (blog, SaaS app, e-commerce, portfolio, etc.).

Landing Page & Marketing Templates

Tailwind CSS is hugely popular for landing pages in 2025, and many high-quality templates exist: - **Cruip Templates:** Cruip offers both free and premium Tailwind templates. **“Simple Light”** and **“Open”** are two free Next.js/Tailwind landing page templates on GitHub [33] [34] . They feature modern layouts (hero sections with images, feature grids, CTAs) and are fully responsive. Cruip’s premium templates (available on their site or as part of “Cruip Unlimited”) include beautifully designed startup and SaaS landing pages with advanced components (pricing tables, sign-up forms). Cruip templates often come in pure React/Next.js form, making them easy to integrate into a Next.js project. They are released under permissive licenses (MIT for the free ones [35]). Use these as a starting point for **startup launch pages, product sites, or marketing websites** – they follow latest design trends and are optimized for fast deployment on Vercel. - **Tailwind Labs Templates:** Tailwind UI (premium) comes not just with components but also **page examples**. These include landing pages, about pages, contact forms, etc., designed to be strung together for a coherent site. If you have Tailwind UI access, you can assemble a marketing site quickly by mixing these examples (e.g., a hero from one example, a pricing section from another). - **Medevel’s Top Picks:** According to a curated list by Medevel in July 2025, the Tailwind ecosystem has many open templates [36] . Examples include **Finwise (Next.js)** – a mobile app landing page with smooth Tailwind + Framer Motion animations [37] , and **Inazuma** – a clean, company landing page template rebuilt in Tailwind for flexibility [38] . These are free and open-source, often on GitHub, and come with features like built-in SEO tags, accessibility, and performance optimizations (Finwise boasts a near-perfect Lighthouse score) [39] . - **Astro & Svelte Templates:** Tailwind is framework-agnostic, so you’ll find templates for Astro, SvelteKit, and others. **AstroWind** is a famous free template (MIT licensed) that combines Astro and Tailwind for a blog/landing site. It includes out-of-the-box SEO optimization, dark mode, RTL support, and great performance [40] . If you prefer a static-site approach with Tailwind, AstroWind is a solid choice for **blogs, documentation sites, or landing pages** that need Astro’s island architecture and Tailwind’s styling. - **Other Sources:** Sites like UIdeck and Tailwind Made aggregate free templates. **UIdeck** provides a range of free Tailwind CSS templates for startups, SaaS, and business websites [41] . **Tailwind Awesome** and **GitHub** searches can surface community templates – always check the date and Tailwind version (some older templates might need class name updates if they were built for Tailwind 2.x or 3.x).

AI & Niche Templates: With the surge in AI products, some templates specifically target AI and tech startups. For instance, Vercel released a **Next.js AI Chatbot** example (using OpenAI APIs) which doubles as a template for building chat interfaces, and it’s free [42] [43] . There’s also **Hume AI’s voice interface template** (Next.js) for voice-based chatbots [44] . These templates give you a head start in building conversational UIs or AI service landing pages – useful for hackathons or MVPs of AI-powered apps. They typically integrate Tailwind for styling the chat layout, message bubbles, etc., and come ready to deploy on Vercel.

Dashboard & Admin Templates

Tailwind is equally popular for admin dashboards due to its ability to create custom, lightweight UIs. Here are some widely used templates/kits: - **TailAdmin (React):** TailAdmin is a free React + Tailwind admin dashboard template [45] . It provides the building blocks for a typical dashboard (side navigation, top bar, cards, tables, etc.) and comes in a free version with one default dashboard and dozens of components [46] . Uniquely, TailAdmin offers a Figma design source even for the free version, and a pro version with multiple pre-built dashboard variations (Analytics, E-commerce, CRM, etc.) [47] . It’s MIT-licensed (open-source on GitHub) and a great starting point for an internal tool or SaaS admin panel. If using the free version, you get

TypeScript React code, which you can plug into your project. The **design is clean and minimal**, easy to brand by adjusting Tailwind's color palette. - **Mosaic Lite (Cruip): Mosaic Lite** is a free React/Tailwind dashboard template by Cruip [48]. It includes a set of pages (dashboard homepage, settings, profile, etc.) and uses Chart.js for charts out of the box [49]. With nearly 2k GitHub stars, it's a popular choice and demonstrates how to set up a Tailwind layout with a fixed sidebar and content area. Cruip also offers Mosaic in a pro version with more pages and features. The free template is ideal for **SaaS dashboards or admin panels** – you get a polished starting point with charts, and you can expand it as needed. - **Notus React (Creative Tim):** Notus is an older but still relevant Tailwind React kit that includes both **landing page sections and dashboard UI** components [50]. It follows a design inspired by Material Kit, but built with Tailwind classes. Notus React is free (MIT) and provides a hybrid of marketing + admin pages. If your project needs both a public website and an admin area, Notus can be a convenient all-in-one. Keep in mind it was built with Tailwind 2.x and React 17, so you may need to upgrade it to Tailwind 3 or 4 (which mostly involves updating config and classes). - **Dashwind (DaisyUI Admin): Dashwind** is a free **admin template built with DaisyUI + React** [51]. It showcases how DaisyUI's pre-themed components can be used to create a slick dashboard. Because it uses DaisyUI, theming is easy (you can switch the entire dashboard's look by changing the theme name). Dashwind also comes with state management set up (Redux Toolkit) and some boilerplate for forms, etc. [51]. If you are already a fan of DaisyUI's class style, this template gives a huge jump-start for building a functional app (for example, a CRM or inventory management UI). - **Admin One (JustBoil):** Admin One is a Tailwind CSS 3 + React 18 dashboard that includes features like **dark mode, styled scrollbars, and reusable component classes** [52] [53]. It's completely free and is known for its "batteries included" setup – routing, auth screens, etc., are laid out. License is MIT. This template is great for **quickly launching an internal admin** with modern design touches. The dark mode toggle and overall styling make it feel like a premium template for $0. - **Horizon UI (Tailwind Edition):** Horizon UI was a popular Chakra UI dashboard, and now there's a Tailwind React version. It's a beautiful, trendily designed dashboard with lots of pre-built pages (including uncommon ones like an **NFT dashboard page**). Horizon UI Tailwind version is open-source (with a commercial variant available) [54]. It claims to be one of the **fastest and trendiest** Tailwind dashboards, with example pages for profile, authentication, and more [55]. Use Horizon if you want a more out-of-the-box **polished UI/UX** – e.g., for a product where the dashboard is customer-facing and needs to impress. - **Material Tailwind Dashboard:** This is a free admin template that combines Tailwind with a Material Design style (by Creative Tim) [56]. It includes ~40 components (buttons, cards, navbars, etc.) styled to look like Google's Material design, but using Tailwind classes under the hood [57]. Good for those who want a familiar Material aesthetic but prefer Tailwind's utility approach for customization. It's free (MIT) and comes with light/dark modes. However, since it's Material-esque, it might not blend well if your frontend site isn't Material – consider using it if you specifically like that style.

Many of these templates are open source and can be found on GitHub – always check the README for integration steps. Usually, you can either clone the repository or copy the relevant folders (components, layouts, pages) into your project. Pay attention to the **Tailwind config** in these templates; some use specific Tailwind plugins (for instance, forms, typography, or line-clamp plugins) or custom `extend` configurations for colors and fonts to match the template design.

UI Kits & Multi-Page Packs

There are also comprehensive UI kits that provide a collection of pages and components: - **Shuffle.dev and Tailkits:** These offer **hundreds of components and multiple templates** that you can mix and match with a drag-and-drop builder (Shuffle) or as downloads. For example, Shuffle's Tailwind library has 7,000+ components categorized by theme [58] [59]. These are commercial services, but useful if you want a large

library of snippet designs to construct unique pages without designing from scratch. - **Tailwind UI kit by themes.dev / Wicked Templates:** These marketplaces sell bundles of Tailwind templates (often with multiple color schemes, dark mode, etc.). **Wicked Templates** and **Tailwind Awesome** have kits focusing on niches like **portfolio sites, coming soon pages,** or **e-commerce** product pages. - **AI-Specific Kits:** As seen on Tailkits, some premium kits target AI startups – for example, a bundle of AI-focused landing blocks, or a **React chat template built with shadcn/ui** for a polished chat interface [60] [61]. If you're building an AI SaaS, these can save time by providing styled chat bubbles, conversation layouts, and sections to highlight AI features. - **E-commerce Templates:** Tailwind can be used for e-commerce fronts; a notable one is **Next.js Commerce** (open-source on Vercel's GitHub). It's a full **headless commerce storefront** using Tailwind, integrating with providers like Shopify. While complex, it's a treasure trove of patterns for product grids, carts, etc., all in Tailwind. For simpler needs, Tailwind UI's e-commerce components (if you have access) cover product listings, shopping cart pop-overs, etc.

Optimized for React/Next.js vs. Plain HTML

Many modern Tailwind templates are built with React/Next.js to take advantage of components and deployment platforms like Vercel: - **Next.js Templates:** Using a Next.js-based template (such as those by Cruip or the SaaS starters below) means you get a project structure ready to deploy. Routing, head tags, and often some state management or form handling is preconfigured. If your project will be in React, starting with these is wise since you can directly integrate auth, state libraries, etc. - **Plain HTML Templates:** Some free templates (like older ones on Tailwind Starter Kit or certain UIdeck templates) are just static HTML with Tailwind classes. These can be integrated into any environment (including Razor, Blade, Django, etc.), but if you later migrate to React, you'll convert those to JSX. They are fine for quick landing pages where you just need to edit text/images. For a knowledge-base or AI copilot, HTML templates are "framework neutral" but might require more manual work to break into reusable pieces.

Lovable Integration: *Lovable* (an AI-powered app builder) can generate a Tailwind project structure. To integrate a template or kit into a Lovable-generated project, follow these tips: - **Match the Framework:** If Lovable generated a Next.js project, prefer templates that are Next.js/React for easier merging. You can drop in pages or components from the template into the Next app. For example, to use the "Open" landing page, you might copy its `pages/index.js` (or the new `app/page.tsx` if using App Router) and any custom components/styles into your Lovable project. - **Tailwind Config:** Check if the template's tailwind.config.js has custom theme extensions (colors, fonts). Merge those into your Lovable project's Tailwind config so the styles look correct. For instance, many templates define brand colors or enable dark mode via class strategy. Ensure your project's config is adjusted (e.g., include `daisyUI` plugin if using a DaisyUI template, or add Flowbite plugin if required). - **Install Dependencies:** For component libraries or kits, install any needed packages. Example: using Flowbite components, run `npm install flowbite` (and `flowbite-react` if you use React components). For DaisyUI, `npm install daisyui` and add it to your Tailwind plugins. For shadcn/ui components, you'd run the CLI to add them (or copy the code) – make sure to also install Radix UI packages and any icons as specified. - **Copy-paste vs. CLI:** Some libraries offer CLIs for integration. **shadcn/ui's CLI** will prompt for which component to add and handle installation [62]. DaisyUI and Flowbite require adding their plugin or CSS include. Tailwind UI is copy-paste from the documentation (ensure you grab both the HTML and any JS logic or ARIA labels they note). When copying HTML, don't forget to include the same classes and parent elements as in the example for it to function and style correctly. - **Theming & Customization:** If the template or kit has theming (like DaisyUI's themes or HeroUI's plugin), decide if you will use it or replace it. For DaisyUI, you can set `data-theme="..."` on `<html>` or body to choose a theme, or define a custom theme in tailwind.config. For Flowbite, you might

override Tailwind's color palette to match your brand (Flowbite will use Tailwind's default colors unless you customize). For kits like TailAdmin or others, you can simply swap utility classes for colors or spacing as needed – since it's Tailwind, you rarely need to write new CSS. - **Responsive Design:** All reputable templates are mobile-first responsive. Ensure your layout includes the appropriate meta viewport tag (Lovable likely adds this). Test the integrated pages on different screen sizes. Tailwind's small (`sm:`) and larger breakpoint classes should already be in use from the template – if you find any element not adapting, you might need to adjust classes. For example, if you integrate a template's navbar, double-check that its mobile menu logic (maybe a hamburger button) is implemented – you might need to copy a small piece of JS or use Headless UI's `Disclosure` component if it was not a full React template. - **Testing integration:** After merging, use Tailwind's **JIT mode** output to see that all classes from the template are being generated (Tailwind will purge unused styles; if the HTML is in your project files, they'll be included). If some styles seem missing, it could be that class names weren't picked up – e.g., if they are assembled dynamically in React, ensure those class strings are not mistakenly excluded by PurgeCSS. This is more of a concern for advanced use, but worth noting. - **Don't Mix Too Much:** It's tempting to use multiple UI libraries together (e.g., DaisyUI for some parts and Flowbite for others). Technically this can work, but be cautious: you might end up with style inconsistencies (one uses rounded borders everywhere, the other uses sharp corners). If you do combine, try to unify the theme (for example, override DaisyUI's theme to closer match Flowbite's style or vice versa). Also, watch for **class name overlaps** – DaisyUI's classes like `btn` could conflict with some other library's usage. However, since most others stick to using long Tailwind utility class lists, conflicts are rare. It's generally best to choose one primary component style approach for a cohesive look.

Integration Guidance for an AI Co-pilot

How to integrate these components and templates into your project (Lovable-generated or otherwise) is as important as choosing them. Here are actionable guidelines:

- **Setting up Tailwind:** Ensure your project has Tailwind configured (Lovable likely does this for you). That means a `tailwind.config.js` and the Tailwind directives (`@tailwind base; @tailwind components; @tailwind utilities;`) in your CSS. Most third-party components assume Tailwind is already working.

- **Installing Component Libraries:** For any library that isn't pure-CSS:

- **DaisyUI:** Install via `npm i daisyui` and add `"daisyui"` to the `plugins` array in tailwind.config. DaisyUI will generate all its component styles automatically. You can then use Daisy's classes (like `btn`, `card`, etc.) directly in your JSX/HTML. Configure themes by adding `daisyui: { themes: [...] }` in tailwind.config if you want only specific ones or custom.
- **Flowbite:** Install `npm i flowbite`. In your main JS (e.g., `app.jsx` or `index.js`), import the JS if you use components like modals (`import 'flowbite';`). Also, include Flowbite's CSS: either via CDN link in HTML or by adding `@import "flowbite/plugin";` in Tailwind config plugins (Flowbite provides a plugin to generate its components' styles). Flowbite also has `flowbite-react` – if you prefer JSX components, `npm i flowbite-react` and use those (ensure to wrap your app in their `Flowbite` context provider if needed, for theming).
- **shadcn/ui components:** Follow the docs on ui.shadcn.com – typically run `npx shadcn-ui@latest init` once to set up, then `npx shadcn-ui add component-name`. This will scaffold the component files (JSX + Tailwind classes) in your project. The CLI will also install

dependencies like `@radix-ui/react-*` and class variance authority (CVA) if the components use it. After adding, you can import and use the component as if you wrote it yourself. *Tip:* If using many shadcn components, periodically update them by checking the shadcn/ui site for changes – since you won't get package updates (the code is yours), you'll manually sync improvements.

- **Headless UI / Radix:** Just install via npm (`@headlessui/react`, `@radix-ui/react-dialog`, etc.) and follow usage examples. You'll write the JSX and Tailwind classes as shown in their docs. For Headless UI, for example, to integrate a `<Listbox>` dropdown: copy the example from their docs, adjust options, and style it with Tailwind classes to match your app. It's straightforward but remember to include any required context providers (Radix components often need none beyond the component tag, Headless UI needs none either).
- **Tailwind UI:** Since it's not a library, just copy the needed code. If you use a React variant from Tailwind UI, also ensure you have Headless UI installed for things like `<Menu>` or `<Dialog>`. Tailwind UI's site usually notes if a component requires Headless UI. Include any auxiliary code (for example, a bit of JS to toggle mobile menu state or to switch tabs – Tailwind UI often provides that in the documentation snippet).

- **Templates (HTML/Next):** If it's a Next.js template, you can copy the page files and components. For example, integrating **Next.js SaaS Starter**: you might use it as a base instead of Lovable's output if it's drastically different. But you can also copy specific parts (like the pricing section or team bios) into your existing pages. Keep the folder structure organized – place assets (images, icons) either in public or the proper folder as referenced in the code.

- **Theming Tips:** Tailwind's strength is customization. Leverage the config to customize any library:

- If a library's default styles are close but not exact, override them via Tailwind classes or config. For instance, Flowbite might use the `blue-500` as primary color by default. If your brand color is green, you can set `theme.colors.primary` to your green and find if Flowbite components allow using `text-primary` classes, or just override by adding your own classes.
- DaisyUI allows full theme creation: you can duplicate one of their themes in the config and adjust values (colors, fonts) easily. This way, all Daisy components will use your colors automatically.

- For shadcn or Tailwind UI components, you'll manually adjust classes – which is fine. Consider defining some CSS variables or Tailwind theme extensions for consistency (e.g., set `primary` color in Tailwind theme and use `text-primary` in classes). This makes it easier to change later or to apply dark mode variants.

- **Ensuring Responsiveness:** Always verify that your integrated components behave on mobile. Most Tailwind UI and DaisyUI components are mobile-first already (Tailwind CSS itself is mobile-first). However, when you compose a page, ensure you use grid and flex utilities appropriately:

- Use Tailwind's responsive prefixes (`sm:`, `md:`) to adjust layout. For example, for a 3-column feature section on desktop that should stack on mobile, you might have `grid grid-cols-1 md:grid-cols-3 gap-6`.
- Check any fixed widths or margins that might break smaller screens. A common issue is copying a template that has a lot of padding on desktop and forgetting to adjust it on mobile – Tailwind lets you easily reduce `px-16` to `px-4` on small screens by `px-4 sm:px-16`.

- If using a template's existing classes, likely they handled this, but if you tweak things, keep the mobile-first approach.

- **Test dark mode:** If your project uses dark mode (via `class="dark"` on `<html>` or media), test that components visible in both modes switch appropriately. Many libraries provide dark classes (e.g., `dark:bg-gray-800`). If not, you might need to add them. DaisyUI themes, for example, automatically adjust if you set `data-theme="dark"`.

- **CLI vs. Copy-Paste vs. NPM:** Which integration method is best often depends on how much you plan to customize:

- **CLI (shadcn/ui):** best when you want a *starting point* and will heavily customize the components. It injects code into your project which you then treat as your own. This is great for long-term maintainability in a product where design will evolve.
- **Copy-paste (Tailwind UI, HyperUI, etc.):** fastest when you just need a specific design element or page. The snippet is isolated and you might only change text or small styles. The risk is inconsistency if you pull from multiple sources with different styles, but Tailwind can unify them if you tweak classes.
- **NPM Package (Flowbite, Headless UI, HeroUI):** convenient if you want to regularly pull updates or you prefer using pre-built React components. This can save time (e.g., `<Flowbite.Button color="blue">` rather than writing classes). But you are constrained to their API unless you eject. Also, updates could change styles slightly, so lock versions when you have a stable design.

- **Hosted/CDN (older approach):** You can include Tailwind CSS and component library CSS via CDN for simple sites (no build step). E.g., link Flowbite's CSS and JS in a `<script>`. This is fine for prototypes but for a serious app, use a build step for performance (and to tree-shake unused CSS).

- **Developer Experience:** All these integrations aim to improve DX. Tailwind's utility classes mean that once integrated, customizing is just editing class strings, which AI co-pilots excel at. Encourage using consistent class naming patterns and perhaps extracting repeating patterns with `@apply` in your CSS if needed. For example, if your design calls for many similar card styles, you could `@apply shadow-lg rounded-xl p-6` in a custom CSS class and use that class across components (though Tailwind purists often just use the utilities directly in each element).

By following these guidelines, an AI co-pilot or any developer can confidently incorporate Tailwind UI components or templates into a project, ensuring the result is cohesive, responsive, and maintainable.

Design Trends in 2025 (Tailwind UI Patterns)

Staying up-to-date with design trends is key to building UIs that feel modern. In 2025, several UI design trends are influencing Tailwind-based projects:

- **Neumorphism (Soft UI) Comeback:** Neumorphism – the "soft shadow" design style that blends elements into backgrounds with subtle inner/outer shadows – has made a quiet comeback [63]. When used sparingly (e.g. for cards or toggle switches), it gives a **tactile, embossed look** that can make interfaces feel more tangible. Tailwind can implement neumorphism using utility classes for shadows and colors (e.g., a light gray background with `shadow-inner` for inset shadows, or

custom shadow sizes). The key is low contrast and **very soft shadows** [64] [65]. Neumorphic elements work well in **health and wellness apps or minimalist dashboards** where a calm aesthetic is desired [66] [67]. Caution: ensure enough contrast for accessibility; usually this means you won't apply neumorphism to text or critical icons – it's more for decorative chrome.

- **Glassmorphism:** The frosted-glass effect remains popular for giving UIs a **sleek, futuristic feel** [68]. Glassmorphism involves translucent backgrounds with backdrop blur (CSS `backdrop-filter`) to simulate frosted glass panels over vibrant backgrounds [69] [70]. Designers use it in **hero sections, modals, or card overlays** to add depth. In Tailwind, this is achieved with classes like `backdrop-blur` (and setting appropriate opacity on background color, e.g. `bg-white/30`). By 2025, browser support for backdrop-filter is solid on all modern browsers, so it's safe to use. Glassmorphism combined with bright gradients can make a UI feel "high-tech" – it's seen in fintech dashboards, crypto apps, and some **AI platform UIs** (to convey a modern, technical vibe) [71] [72]. Tip: ensure text on a blurred background remains readable (often a slight drop shadow on text or an overlay will help). Use this trend **strategically** – e.g., a glassmorphic navbar over a colorful canvas, or cards on a dashboard with blur to separate from a background graphic.

- **Neubrutalism:** On the other end of the spectrum, **neubrutalism** (new brutalism) is trending for startups that want a bold, artsty look [73] [74]. This style features high contrast, flat colors (often almost retro 90s or neon), thick borders, and quirky typography. Tailwind's flexibility makes it capable of brutalist design – you might use stark color classes (like `bg-yellow-400` with `text-pink-800`), add thick borders (`border-4 border-black`), and play with unusual spacing. Neubrutalism rejects the subtlety of neumorphism/glassmorphism and instead **"shouts" visually** [74] [75]. This can make a site stand out (great for creative agencies or dev portfolio sites). If using this style, consistency is key: apply the bold look to all elements (buttons with no shadows, just solid fills, for example) so it's clearly intentional.

- **Dark Mode as a Standard:** By 2025, users expect dark mode. All the major UI kits and templates support it, and your design should too unless there's a specific reason not to. Tailwind makes dark mode easy with its `dark:` variant. The typical approach is toggling a `.dark` class on `<html>` (or using OS preference via `media` queries). **Ensure color choices for dark mode** are well tested – e.g., using Tailwind's slate/gray ranges for neutral backgrounds (slate-900 etc.) and adjusting text to lighter colors. Many templates now come with dark mode ready – e.g., AstroWind, Finwise, and others note dark mode support out of the box [40]. It's a good practice to design light and dark in tandem. Also, **remember to handle images or illustrations** (use CSS `invert` or provide alternative images if needed for dark backgrounds).

- **Micro-interactions & Motion:** Animation and motion design continue to be important. Two preferred ways to add motion in Tailwind projects:

- **CSS Transitions & Animations:** Tailwind has utility classes for transitions (`transition duration-300 ease-in-out`) and keyframe animations (like `animate-spin`, `animate-bounce`). These are great for simple hovers, button presses, loading spinners, etc. Use them for subtle effects like a hover glow or card lift (`hover:shadow-xl hover:-translate-y-1`).
- **Advanced JS libraries:** For orchestrated animations, **Framer Motion** remains a top choice for React. It provides an intuitive API to animate Tailwind-styled components (e.g., fade in, slide, staggered

lists). Framer Motion is powerful for things like page transitions or complex UI element animations. Another emerging tool is **Motion One** – a lightweight animation library that directly uses the Web Animations API (good for when you want to animate in non-React contexts or need very high performance) [76] [77] . It's only ~5kb and works with any framework, so you can trigger Motion One animations on DOM elements that have Tailwind classes without any React dependency. By 2025, Framer Motion and Motion One have actually converged under a unified **"motion.dev"** project (Framer Motion's author announced Motion as the future) – the idea is to provide high performance, orchestrated animations easily [78] . **Use case:** adding a nice spring transition when a modal opens, scroll-reveal animations as sections enter viewport (for which some use AOS, but you can do the same with Intersection Observer + Framer Motion for more control).

- **Other libraries:** GSAP is still a go-to for complex timeline-based animations (e.g., fine-grained control over an intro sequence). Lottie is popular for integrating animated SVG/canvas illustrations (you can even control them on scroll). In Tailwind projects, these can be used as needed – just ensure to set container sizes and maybe use Tailwind's `aspect-w/h` utilities for responsive Lottie containers.

- The rule of thumb: **Enhance UX, don't distract.** Animations should make the interface feel smoother and more responsive, like giving feedback on actions or drawing attention to key changes. For example, a slight shake on an invalid form field (can be done with a CSS keyframe or a quick Motion One sequence) can help user experience. Tailwind's utility classes can be used in combination with animation libraries (e.g., apply `opacity-0` class and then let Framer Motion add `opacity-100`).

- **Preferred Layout Structures:** In 2025, building layouts often favors **CSS Grid for overall page layout** and **flexbox for smaller UI groupings**. This "grid-first" mentality means you might design your page in terms of rows and columns from the start:

- Tailwind's `grid` utilities make it easy to create responsive grids. For example, a marketing layout might be `grid-cols-1 md:grid-cols-2` for a two-column layout that stacks on mobile. Grid is also great for dashboards (e.g., a fluid 12-column grid for analytics cards that reflow).
- **Mobile-first, of course:** Tailwind is inherently mobile-first (styles apply to all sizes unless overridden by a breakpoint). Ensure your design process also thinks mobile-first: design the small screen layout, then progressively enhance for larger screens with `md:`, `lg:` classes. This ensures you don't inadvertently hide content for mobile users or make assumptions that break on small screens.
- **Container queries:** By 2025, container queries have started rolling out in browsers and Tailwind has experimental support via plugins. This allows components to adapt based on the parent container size, not just the viewport. While not widely used yet, if your UI has embeddable widgets or you want truly modular responsive components, container queries could be a game changer. Tailwind's plugin (if enabled) might give classes like `cq-sm:` for container-breakpoints. Keep an eye on this as it matures for fine-tuned layouts especially in complex dashboards where panels resize.
- **Spacing and padding trends:** Modern designs are embracing more **white space** (or "empty space") especially in content-heavy sites – generous `py-16` or `pt-20` in sections is common, with Tailwind it's trivial to adjust. Also, the use of **fluid spacing** (clamp with `min`, `vw`, etc.) sometimes appears, but Tailwind primarily uses fixed spacing scale. If needed, you can always extend Tailwind with fluid values or use `max-w-prose` and such for readable text widths.

- **Vertical rhythm:** Many Tailwind templates use the Typography plugin for rich text content, which ensures a good vertical rhythm (consistent spacing for headings, paragraphs, lists). This is important for blog sections or documentation pages that might be part of your app.

- **Visual Flourishes:** A few other design patterns seen:

- **Gradients and Duotone**: Gradients (especially with Tailwind's `bg-gradient-to-r from-indigo-500 to-purple-500` etc.) are still in, often as background fills for hero sections or call-to-action banners. By 2025, we also see duotone images and icons (Tailwind doesn't handle that directly, but you can apply filters or use SVGs).
- **3D and Depth:** Subtle 3D elements like illustrations or even actual 3D models (using three.js or model viewer) might be present in cutting-edge app UIs. While not directly related to Tailwind, you might style around them (e.g., a 3D canvas taking half the screen with Tailwind classes controlling layout).
- **Motion Graphics:** Some sites use animated backgrounds (CSS or canvas-based). Tailwind can be used to position and layer these (absolute positioning, z-index). One trendy example: an animated gradient or noise texture background under content (with Tailwind classes to size it full-screen and give overlay color).

In summary, **modern Tailwind UIs in 2025 balance minimalism with vibrant accents** – many designs remain clean and content-focused, using things like neumorphism or glassmorphism as a sprinkle for depth. Dark mode and accessibility are table stakes. Motion is used to delight and guide users (with libraries like Framer Motion [79] making it smoother). And new CSS capabilities (grid, container queries) are leveraged via Tailwind to build responsive layouts that work across the huge variety of devices people use today. By incorporating these trends thoughtfully, your UI will feel current and engaging.

Starter Kits & Boilerplates (Next.js, Full-Stack & More)

When starting a new project, using a starter kit can save weeks of setup time. In the Tailwind + Next.js world, 2025 offers some excellent boilerplates that emphasize scalability and clean architecture. These starter projects usually include not just Tailwind and components, but also backend integration, authentication, testing, and more. Here are some of the most notable ones:

- **Next.js Enterprise Boilerplate (by Blazity):** An enterprise-grade boilerplate built with Next.js 13 (App Router) and Tailwind CSS v4 [80]. It comes packed with best practices: TypeScript, ESLint/Prettier, Jest and Playwright for testing, Storybook for UI components, and even headless UI components via Radix UI [80] [81]. It's designed for **large-scale applications** – think of it as a blueprint for a tech startup or enterprise app. It emphasizes a clean **folder structure** (organized into domain-oriented components, perhaps), and includes CI/CD workflows. This boilerplate has an MIT license [82] and is quite popular (thousands of stars on GitHub), meaning it's community-vetted. If you're building a serious app that needs things like authentication, testing pipelines, etc., this is a great starting point. It uses **Radix UI + Tailwind** for its UI components, demonstrating how to use headless components at scale [80] [83]. It also integrates CVA (Class Variance Authority) to manage Tailwind class consistency in design system tokens [83]. Essentially, it's an **opinionated, production-ready template** you can fork. Expect a somewhat higher learning curve (because of all the tools integrated), but excellent maintainability.

- **T3 Stack (Create T3 App):** The **T3 Stack** isn't specifically Tailwind UI focused, but it's a popular full-stack Next.js starter that includes Next.js, tRPC (typesafe API), Prisma (database ORM), NextAuth (auth), **Tailwind CSS**, and TypeScript. It's geared towards developers who want an end-to-end typesafe environment. The project scaffold (via `create-t3-app`) sets up a clean monorepo-like structure. It's MIT licensed and widely used in the community. If your AI copilot will help build full-stack apps, T3 is a great context to know – you get a solid backend and frontend foundation with Tailwind ready to style your components. The T3 Stack doesn't come with pre-designed UI components out of the box (beyond Tailwind's default styling), so you might still integrate a UI kit like shadcn or DaisyUI on top of it.

- **Next.js SaaS Starter Kit (e.g. by Blazity or Vercel):** We saw a **Next SaaS Starter** (by Blazity) that is a free MIT-licensed template targeting marketing sites for SaaS with a blog, CMS integration, etc. [84] [35]. There's also the official **Vercel SaaS Subscription Starter** which combines Next.js with Stripe for subscriptions, Supabase for auth and data, and Tailwind for styling [85] [86]. These starter kits are fantastic for building a subscription-based product – they handle the boilerplate of plans, payments, and accounts. They usually include a minimal UI (Tailwind styled) that you can then customize. For example, Vercel's Subscription Starter is designed to be **"the all-in-one subscription starter kit"** for SaaS, with out-of-the-box subscription billing and a simple dashboard [85]. License on the Vercel starters is typically MIT as well (Vercel encourages use of their templates for any purposes). Use these kits if you're building something like a SaaS web app or any product that needs user accounts and payments from day one – it will save a lot of setup on the backend, and Tailwind makes it easy to skin the frontend to your branding.

- **Platformatic/Multitenant Starter (Vercel Platforms Starter):** Vercel's **Platforms Starter Kit** is a template for building multi-tenant applications (e.g., like Notion or Shopify, where each user/team has their own space) using Next.js App Router, Tailwind, and Prisma + Redis [87]. It demonstrates patterns for routing with tenant-specific content and is a great reference for those building B2B apps where each customer has a subdomain or separate data. It's also MIT licensed and can be deployed with one click. The inclusion of Tailwind means all the UI is easily customizable. It uses NextAuth for auth, Prisma for DB, etc. If your project is to build the next multi-tenant SaaS, this kit offers a **scalable starting architecture**. The code structure emphasizes separation of concerns (with clearly defined "modules" for app, dashboard, etc.), which keeps it maintainable.

- **Taxonomy (by shadcn): Taxonomy** is an open-source example app built by shadcn (the author of shadcn/ui) to showcase Next.js 13 features and his UI library usage [88]. It includes a full stack app with authentication, a Prisma database, and a rich text editor, all styled with Tailwind and using **shadcn's components**. Taxonomy is essentially a **template for a content-heavy app** (like a blog platform or knowledge base) – it has features like dark mode, a text editor (using TipTap), and shows how to combine server components with client components elegantly. This is a great boilerplate if you want to see how an expert structures a Next.js project with Tailwind + Radix (via shadcn). It's MIT licensed and meant for learning, but one could fork it as a starting point for a content application. The folder structure and use of hooks, context, etc., are exemplary for a modern Next app.

- **Remix/Blitz/Nuxt, etc.:** While the question focuses on Tailwind and Next.js, note that there are starter kits in other frameworks too: e.g., **Blitz.js** (a Rails-like fullstack React framework, now on Remix) had a Tailwind-powered SaaS template. **Nuxt 3** (for Vue) has a Tailwind starter as well. If your

project isn't locked to React, you might consider those, but the mainstream and most documented ones are Next.js based.

When choosing a starter, consider: - **Community and Maintenance:** A starter with active maintainers (like the T3 stack or Blazity's kits) will keep up with new Next.js and Tailwind versions. For example, when Next.js 13 introduced the App Router, these starters quickly updated to incorporate best practices with it. - **Features vs Simplicity:** Some boilerplates include *everything and the kitchen sink* (which can be great, but you may not need all of it). For instance, the Next.js Enterprise has CI, analytics, etc. configured. If that's overkill for a small project, a simpler starter like the Next.js default + Tailwind (via `npx create-next-app --example with-tailwindcss`) might suffice and you add pieces as you need. - **License:** Most of these are MIT (as cited above [82]), which means you can use them freely even in commercial projects. Just don't remove the license file – keep attribution as required. Some premium starters (like certain Creative Tim products or Themeforest items) might have more restrictive licenses (e.g., one-time purchase per project). Always check to avoid compliance issues. - **Folder Structure & Clean Code:** A hallmark of a good starter is a logical project structure. Look for separation of concerns: e.g., all UI components in a `components` directory (possibly further divided by domain or feature), pages in a `pages` or `app` directory for Next.js, utility functions in `lib` or `utils`, etc. Also, things like having a consistent naming convention, TypeScript interfaces/types in a `types` folder, etc., are signs of a thoughtful boilerplate. These choices greatly affect maintainability – it's easier to onboard new developers (or have an AI copilot understand the project) if things are organized and not too ad-hoc. - **Performance and SEO:** Some starters put emphasis on performance (e.g., Next Enterprise aiming for perfect Lighthouse scores [89] [90]) and SEO (SaaS starter kits often include meta tags, OpenGraph setup, sitemap generation). If those are important for your project, choose a starter that has them set up so you don't miss those details.

Finally, remember that a starter kit is a starting point – you're free to modify or strip out parts that you don't need. The value is in not having to reinvent common features (auth, config, styling setup) and learning from the best practices baked in. As you build with Tailwind and these kits, you'll appreciate how opinionated setups (like using **Radix UI in a starter** for all interactive components, or using **CVA for class management** as in Next Enterprise [83]) can enforce consistency across the app. And consistency is key to long-term maintainability.

Sources: The information above is compiled from the official documentation and comparisons of Tailwind CSS libraries and templates, including DaisyUI's component library comparisons [24] [91], a DEV Community roundup of free Tailwind UI libraries [92] [93], template showcases from Medevel and Dev.to [37] [45], and various project repositories (Next.js boilerplates [82], open-source templates, etc.). These resources provide insights into the features, trade-offs, and use cases of each tool mentioned. Each library or template referenced above is backed by community usage and up-to-date as of 2025, ensuring that this guide reflects current best practices in Tailwind CSS UI development.

---

[1] [16] UI Components: ShadCN, Tailwind UI, Headless, React Aria, Radix UI - Syntax #751
https://syntax.fm/show/751/ui-components-shadcn-tailwind-ui-headless-react-aria-radix-ui

[2] [3] [17] [22] [23] [24] [25] Tailwind UI vs Flowbite - daisyUI is a Tailwind UI alternative — Tailwind CSS Components (version 5 update is here)
https://daisyui.com/compare/tailwindui-vs-flowbite/

4 5 6 7 11 13 14 20 shadcn/ui Alternative in 2025 - Why daisyUI is better? — Tailwind CSS Components (version 5 update is here)
https://daisyui.com/alternative/shadcn/

8 9 45 46 47 48 49 50 51 52 53 54 55 56 57 Top 7+ Free Tailwind React Admin Dashboard Templates for 2025 - DEV Community
https://dev.to/tailwindcss/top-7-free-tailwind-react-admin-dashboard-templates-for-2024-1gc9

10 12 15 21 26 62 91 Flowbite vs shadcn/ui - daisyUI is a Flowbite alternative — Tailwind CSS Components (version 5 update is here)
https://daisyui.com/compare/flowbite-vs-shadcn/?lang=en

18 19 27 28 92 93 10 Best Free UI Libraries Based on Tailwind CSS in 2025 - DEV Community
https://dev.to/joodi/10-best-free-ui-libraries-based-on-tailwind-css-in-2025-3a98

29 30 31 32 HeroUI (Previously NextUI) - Beautiful, fast and modern React UI Library
https://www.heroui.com/

33 34 36 37 38 39 40 79 84 10+ Most Used Tailwind CSS Landing Page Templates in 2025 (Free & Open Source)
https://medevel.com/10-most-used-tailwind-css-landing-page-templates-in-2025-free-open-source/

35 GitHub - Blazity/next-saas-starter: ⚡ Free Next.js responsive landing page template for SaaS products made using JAMStack architecture.
https://github.com/Blazity/next-saas-starter

41 Free Tailwind CSS Templates - UIdeck
https://uideck.com/tailwind-templates

42 43 44 58 59 60 61 Tailwind AI Website Templates | Tailkits
https://tailkits.com/templates/categories/ai/

63 Your UI Is Boring — Bring It to Life with Neumorphism & Glassmorphism in 2025 | by Aryan Garg | Bootcamp | Jun, 2025 | Medium
https://medium.com/design-bootcamp/your-ui-is-boring-bring-it-to-life-with-neumorphism-glassmorphism-in-2025-0b3b44eb75e9

64 65 66 67 69 70 71 72 73 74 75 UI Trends: Neumorphism vs. Glassmorphism vs. Neubrutalism
https://www.cccreative.design/blogs/differences-in-ui-design-trends-neumorphism-glassmorphism-and-neubrutalism

68 Neumorphism vs. Glassmorphism: UI Design Trends for 2025
https://syngrid.com/neumorphism-vs-glassmorphism-which-trend-works-in-2025/

76 Framer Motion is now independent. Introducing Motion. : r/reactjs
https://www.reddit.com/r/reactjs/comments/1gqadrv/framer_motion_is_now_independent_introducing/

77 Matt Perry - (Framer) Motion - The Evolution of Animation Libraries
https://www.devtools.fm/episode/124

78 Framer Motion is now independent, introducing Motion - Motion Blog
https://motion.dev/blog/framer-motion-is-now-independent-introducing-motion

80 81 83 89 90 Next.js Enterprise Boilerplate
https://vercel.com/templates/saas/nextjs-enterprise-boilerplate

82 GitHub - Blazity/next-enterprise: An enterprise-grade Next.js boilerplate for high-performance, maintainable apps. Packed with features like Tailwind CSS, TypeScript, ESLint, Prettier, testing tools, and more to accelerate your development.
https://github.com/Blazity/next-enterprise

85 86 87 Next.js Starter Templates & Themes | Vercel
https://vercel.com/templates/next.js

88 shadcn-ui/taxonomy: An open source application built using the new ...
https://github.com/shadcn-ui/taxonomy