I want container based Architecture so that controlled can be better for setting layout, theme, Content Management

A screenshot of a computer

AI-generated content may be incorrect.

Love this direction. Here’s a clean **container-based architecture** for your Next.js + Carbon app that matches your diagram and keeps layout, theme, and content under control.

1. Folder structure\
2. /app
3. layout.tsx // mounts AppContainer once
4. page.tsx // uses PageContainer + Body/Hero/Section
5. globals.scss
6. /components
7. /containers
8. AppContainer.tsx
9. PageContainer.tsx
10. HeaderContainer.tsx
11. FooterContainer.tsx
12. BodyContainer.tsx
13. HeroContainer.tsx
14. SectionContainer.tsx
15. /context
16. ThemeContext.tsx
17. LayoutContext.tsx
18. ContentContext.tsx
19. /types
20. layout.ts
21. content.ts

2) Types (config drives your containers)

// /types/layout.ts

export type ThemeName = "g10" | "g100";

export interface LayoutConfig {

header: { sticky: boolean; height: number };

footer: { sticky: boolean };

containerMaxWidth?: number; // px

theme: ThemeName;

}

// /types/content.ts

export type SectionKind = "hero" | "text" | "grid" | "custom";

export interface SectionBlock {

id: string;

kind: SectionKind;

props: Record<string, any>;

}

3) Contexts (single source of truth)

// /context/ThemeContext.tsx

"use client";

import { createContext, useContext, useState, ReactNode } from "react";

import { Theme } from "@carbon/react";

import type { ThemeName } from "@/types/layout";

type ThemeCtx = { theme: ThemeName; setTheme: (t: ThemeName) => void };

const Ctx = createContext<ThemeCtx | null>(null);

export function ThemeProvider({ initial = "g10", children }: { initial?: ThemeName; children: ReactNode }) {

const [theme, setTheme] = useState<ThemeName>(initial);

return (

<Ctx.Provider value={{ theme, setTheme }}>

<Theme theme={theme}>{children}</Theme>

</Ctx.Provider>

);

}

export const useThemeCtx = () => useContext(Ctx)!;

// /context/LayoutContext.tsx

"use client";

import { createContext, useContext, ReactNode } from "react";

import type { LayoutConfig } from "@/types/layout";

const LayoutCtx = createContext<LayoutConfig | null>(null);

export function LayoutProvider({ config, children }: { config: LayoutConfig; children: ReactNode }) {

return <LayoutCtx.Provider value={config}>{children}</LayoutCtx.Provider>;

}

export const useLayout = () => useContext(LayoutCtx)!;

// /context/ContentContext.tsx

"use client";

import { createContext, useContext, ReactNode } from "react";

import type { SectionBlock } from "@/types/content";

type ContentState = { sections: SectionBlock[] };

const ContentCtx = createContext<ContentState | null>(null);

export function ContentProvider({ sections, children }: { sections: SectionBlock[]; children: ReactNode }) {

return <ContentCtx.Provider value={{ sections }}>{children}</ContentCtx.Provider>;

}

export const useContent = () => useContext(ContentCtx)!;

4) Containers

// /components/containers/AppContainer.tsx

"use client";

import "@carbon/styles/css/styles.css";

import "@/app/globals.scss";

import { ThemeProvider } from "@/context/ThemeContext";

import { LayoutProvider } from "@/context/LayoutContext";

import type { LayoutConfig } from "@/types/layout";

import { ReactNode } from "react";

const DEFAULT\_LAYOUT: LayoutConfig = {

header: { sticky: true, height: 56 },

footer: { sticky: true },

containerMaxWidth: 1200,

theme: "g10",

};

export default function AppContainer({

layout = DEFAULT\_LAYOUT,

children,

}: { layout?: LayoutConfig; children: ReactNode }) {

// sync CSS custom properties once here

if (typeof document !== "undefined") {

document.documentElement.style.setProperty("--app-header-height", `${layout.header.height}px`);

}

return (

<ThemeProvider initial={layout.theme}>

<LayoutProvider config={layout}>{children}</LayoutProvider>

</ThemeProvider>

);

}

// /components/containers/PageContainer.tsx

"use client";

import { useLayout } from "@/context/LayoutContext";

import HeaderContainer from "./HeaderContainer";

import FooterContainer from "./FooterContainer";

import { ReactNode } from "react";

export default function PageContainer({ children }: { children: ReactNode }) {

const layout = useLayout();

return (

<div className="page">

<HeaderContainer sticky={layout.header.sticky} />

<main className="main-wrap" style={{ maxWidth: layout.containerMaxWidth, margin: "0 auto" }}>

{children}

</main>

<FooterContainer sticky={layout.footer.sticky} />

</div>

);

}

// /components/containers/HeaderContainer.tsx

"use client";

import {

Header as CarbonHeader,

HeaderContainer as Shell,

HeaderName,

HeaderNavigation,

HeaderMenuItem,

HeaderGlobalBar,

HeaderGlobalAction,

SkipToContent,

} from "@carbon/react";

import { Search } from "@carbon/icons-react";

export default function HeaderContainer({ sticky = true }: { sticky?: boolean }) {

return (

<Shell

render={() => (

<CarbonHeader aria-label="Site Header" className={`custom-header ${sticky ? "is-sticky" : ""}`}>

<SkipToContent />

<HeaderName href="/" prefix="" className="logo-tight">

<img src="/images/logo.png" alt="La-sani" style={{ height: 28, marginRight: 8 }} />

<span className="brand-text">La-sani</span>

</HeaderName>

<HeaderNavigation aria-label="Primary">

<HeaderMenuItem href="#">Home</HeaderMenuItem>

<HeaderMenuItem href="#">Products</HeaderMenuItem>

<HeaderMenuItem href="#">Services</HeaderMenuItem>

<HeaderMenuItem href="#">Insights</HeaderMenuItem>

<HeaderMenuItem href="#">About</HeaderMenuItem>

<HeaderMenuItem href="#">Contact</HeaderMenuItem>

</HeaderNavigation>

<HeaderGlobalBar>

<HeaderGlobalAction aria-label="Search"><Search size={20} /></HeaderGlobalAction>

</HeaderGlobalBar>

</CarbonHeader>

)}

/>

);

}

// /components/containers/FooterContainer.tsx

"use client";

import { useEffect, useRef } from "react";

export default function FooterContainer({ sticky = true }: { sticky?: boolean }) {

const ref = useRef<HTMLDivElement | null>(null);

useEffect(() => {

const el = ref.current;

if (!el || typeof ResizeObserver === "undefined") return;

const update = () =>

document.documentElement.style.setProperty("--app-footer-height", `${el.offsetHeight}px`);

update();

const ro = new ResizeObserver(update);

ro.observe(el);

return () => ro.disconnect();

}, []);

return (

<footer className={`custom-footer ${sticky ? "is-sticky" : ""}`}>

<div ref={ref} className="footer-inner">

<div className="footer-top">

<div className="footer-col">

<h4 className="footer-title">La-sani Fluid Power LLP</h4>

<p className="footer-text">Industrial hydraulic solutions — engineering & lifecycle support.</p>

</div>

<div className="footer-col">

<h5 className="footer-subtitle">Contact</h5>

<ul className="footer-list">

<li>Doha, Qatar</li>

<li>+974 5555 5555</li>

<li>info@la-sani.com</li>

<li>Sun–Thu, 9:00–18:00</li>

</ul>

</div>

<div className="footer-col">

<h5 className="footer-subtitle">Website</h5>

<ul className="footer-links">

<li><a className="footer-link" href="#">Home</a></li>

<li><a className="footer-link" href="#">Products</a></li>

<li><a className="footer-link" href="#">Services</a></li>

<li><a className="footer-link" href="#">Insights</a></li>

<li><a className="footer-link" href="#">About</a></li>

<li><a className="footer-link" href="#">Contact</a></li>

</ul>

</div>

</div>

<div className="footer-bottom">

<hr className="footer-divider" />

<div className="footer-copy">© {new Date().getFullYear()} La-sani Fluid Power LLP. All rights reserved.</div>

</div>

</div>

</footer>

);

}

// /components/containers/BodyContainer.tsx

"use client";

import { ReactNode } from "react";

export default function BodyContainer({ children }: { children: ReactNode }) {

return <div className="body-container">{children}</div>;

}

// /components/containers/HeroContainer.tsx

"use client";

export default function HeroContainer({ title, subtitle }: { title: string; subtitle?: string }) {

return (

<section className="hero">

<h1>{title}</h1>

{subtitle && <p className="hero-sub">{subtitle}</p>}

</section>

);

}

// /components/containers/SectionContainer.tsx

"use client";

export default function SectionContainer({ heading, children }: { heading?: string; children: React.ReactNode }) {

return (

<section className="section">

{heading && <h2 className="section-title">{heading}</h2>}

{children}

</section>

);

}

5) app/layout.tsx (mount AppContainer once)

// app/layout.tsx

import AppContainer from "@/components/containers/AppContainer";

export default function RootLayout({ children }: { children: React.ReactNode }) {

return (

<html lang="en">

<body>

<AppContainer>{children}</AppContainer>

</body>

</html>

);

}

6) app/page.tsx (compose your containers)

// app/page.tsx

"use client";

import PageContainer from "@/components/containers/PageContainer";

import BodyContainer from "@/components/containers/BodyContainer";

import HeroContainer from "@/components/containers/HeroContainer";

import SectionContainer from "@/components/containers/SectionContainer";

export default function Home() {

return (

<PageContainer>

<BodyContainer>

<HeroContainer title="Welcome to Apple Books" subtitle="Testing areas under headers" />

<SectionContainer heading="Latest">

<p>Lasani Fluid Power LLP website preparation</p>

</SectionContainer>

</BodyContainer>

</PageContainer>

);

}

7) SCSS (already tuned for fixed header & bottom footer)

Use the **clean consolidated SCSS** I gave you last turn (the final version with --app-header-height + dynamic footer). It already:

* pins header to top,
* keeps footer at bottom with auto-height (ResizeObserver),
* ensures main/body get the remaining space.

Add a few visual styles for hero/section:

/\* extras \*/

.hero { padding: 24px 0; }

.hero h1 { margin: 0 0 8px; font-size: 2.25rem; }

.hero-sub { margin: 0; color: #6e6e73; }

.section { padding: 24px 0; }

.section-title { margin: 0 0 12px; font-size: 1.25rem; }

**Why this architecture works**

* **Website/Application Container** → AppContainer (mounts global providers & CSS once)
* **Page Level Container** → PageContainer (header/footer orchestration)
* **Header/Footer Containers** → consistent across pages, configurable
* **Body/Hero/Section Containers** → composable content blocks (future CMS can map JSON → blocks)
* **Contexts** → switch theme (g10/g100), change layout heights, or feed dynamic sections without touching pages

When you plug in CMS later, you’ll map CMS “blocks” to your SectionBlock type and render them with a simple registry inside BodyContainer if you want.

If you want, I can add a **block registry** (map kind: "hero" | "text" ..." to a component) so pages can render from pure JSON.

have one question: What about if we want to implement non-carbon themes or custom themse. How typogrpahy is controlled . How Color palletes can be controlled ? --- on AppContainer level.

Great question. You can absolutely run **non-Carbon themes** (or a hybrid) from the **AppContainer** and keep full control of **typography** and **color palettes**. The trick is: treat your theme as a **token object** → publish it as **CSS variables** on the root (or a data-theme="..." scope) → optionally map a subset into Carbon’s CSS vars so Carbon widgets pick it up too.

Below is a practical pattern you can drop in.

1) Define a theme spec (tokens)

// /types/theme.ts

export type ThemeName = "g10" | "g100" | "lasaniLight" | "lasaniDark";

export interface ThemeTokens {

name: ThemeName;

/\* palette \*/

colors: {

bg: string;

surface: string;

text: string;

textMuted: string;

primary: string;

primaryContrast: string;

border: string;

};

/\* typography \*/

font: {

family: string;

weightRegular: number;

weightMedium: number;

weightBold: number;

/\* fluid scale using clamp for h1…h6 + body \*/

sizeBody: string; // e.g., "clamp(14px, 1.1vw, 16px)"

sizeH1: string;

sizeH2: string;

sizeH3: string;

lineHeight: number; // e.g., 1.5

};

/\* radii / spacing (optional) \*/

radius: string;

spacing: number; // base spacing unit in px

}

Example themes

// /themes/index.ts

import { ThemeTokens } from "@/types/theme";

export const lasaniLight: ThemeTokens = {

name: "lasaniLight",

colors: {

bg: "#ffffff",

surface: "#f5f5f7",

text: "#1d1d1f",

textMuted: "#6e6e73",

primary: "#0f62fe",

primaryContrast: "#ffffff",

border: "#e5e5ea",

},

font: {

family:

'-apple-system,BlinkMacSystemFont,"Segoe UI",Roboto,Helvetica,Arial,sans-serif',

weightRegular: 400,

weightMedium: 500,

weightBold: 600,

sizeBody: "clamp(14px, 1.05vw, 16px)",

sizeH1: "clamp(28px, 3.2vw, 40px)",

sizeH2: "clamp(22px, 2.2vw, 28px)",

sizeH3: "clamp(18px, 1.6vw, 22px)",

lineHeight: 1.45,

},

radius: "12px",

spacing: 8,

};

export const lasaniDark: ThemeTokens = {

...lasaniLight,

name: "lasaniDark",

colors: {

bg: "#161616",

surface: "#262626",

text: "#f4f4f4",

textMuted: "#b0b0b0",

primary: "#78a9ff",

primaryContrast: "#161616",

border: "#393939",

},

};

2) AppContainer publishes tokens as CSS variables

 You can still use Carbon’s <Theme> for g10/g100 **or skip it** when using pure custom themes.

 We’ll attach variables on <html data-theme="...">.

// /components/containers/AppContainer.tsx

"use client";

import "@carbon/styles/css/styles.css";

import "@/app/globals.scss";

import { Theme } from "@carbon/react";

import { ReactNode, useEffect } from "react";

import type { ThemeTokens, ThemeName } from "@/types/theme";

import { lasaniLight, lasaniDark } from "@/themes";

const THEMES: Record<ThemeName, ThemeTokens> = {

g10: lasaniLight, // you can alias or create proper Carbon-mapped versions

g100: lasaniDark,

lasaniLight,

lasaniDark,

};

function applyCSSVariables(tokens: ThemeTokens) {

const r = document.documentElement;

r.setAttribute("data-theme", tokens.name);

// core variables you will use in SCSS

r.style.setProperty("--app-bg", tokens.colors.bg);

r.style.setProperty("--app-surface", tokens.colors.surface);

r.style.setProperty("--app-text", tokens.colors.text);

r.style.setProperty("--app-text-muted", tokens.colors.textMuted);

r.style.setProperty("--app-primary", tokens.colors.primary);

r.style.setProperty("--app-primary-contrast", tokens.colors.primaryContrast);

r.style.setProperty("--app-border", tokens.colors.border);

r.style.setProperty("--app-font-family", tokens.font.family);

r.style.setProperty("--app-fs-body", tokens.font.sizeBody);

r.style.setProperty("--app-fs-h1", tokens.font.sizeH1);

r.style.setProperty("--app-fs-h2", tokens.font.sizeH2);

r.style.setProperty("--app-fs-h3", tokens.font.sizeH3);

r.style.setProperty("--app-line", String(tokens.font.lineHeight));

r.style.setProperty("--app-fw-regular", String(tokens.font.weightRegular));

r.style.setProperty("--app-fw-medium", String(tokens.font.weightMedium));

r.style.setProperty("--app-fw-bold", String(tokens.font.weightBold));

r.style.setProperty("--app-radius", tokens.radius);

r.style.setProperty("--app-space", `${tokens.spacing}px`);

// OPTIONAL: map to Carbon CSS vars so Carbon components inherit your theme

// Background / text / link example mappings:

r.style.setProperty("--cds-background", tokens.colors.bg);

r.style.setProperty("--cds-layer", tokens.colors.surface);

r.style.setProperty("--cds-text-primary", tokens.colors.text);

r.style.setProperty("--cds-text-secondary", tokens.colors.textMuted);

r.style.setProperty("--cds-link-primary", tokens.colors.primary);

}

export default function AppContainer({

children,

theme = "lasaniLight",

useCarbonThemeWrapper = false, // set true to wrap with <Theme theme="g10|g100">

}: {

children: ReactNode;

theme?: ThemeName;

useCarbonThemeWrapper?: boolean;

}) {

useEffect(() => {

applyCSSVariables(THEMES[theme]);

}, [theme]);

const tree = <>{children}</>;

// Optional: wrap Carbon’s Theme for built-in g10/g100 mode

return useCarbonThemeWrapper ? (

<Theme theme={theme === "g100" || theme === "lasaniDark" ? "g100" : "g10"}>

{tree}

</Theme>

) : (

tree

);

}

**3) Globals that consume those variables**

This lets you change typography + palette app-wide without touching components.

/\* app/globals.scss \*/

/\* Base \*/

html, body {

margin: 0;

height: 100%;

background: var(--app-bg);

color: var(--app-text);

font-family: var(--app-font-family);

line-height: var(--app-line);

font-weight: var(--app-fw-regular);

font-size: var(--app-fs-body);

}

/\* Headings (fluid) \*/

h1 { font-size: var(--app-fs-h1); font-weight: var(--app-fw-medium); margin: 0 0 calc(var(--app-space) \* 1.5); }

h2 { font-size: var(--app-fs-h2); font-weight: var(--app-fw-medium); margin: 0 0 calc(var(--app-space) \* 1.25); }

h3 { font-size: var(--app-fs-h3); font-weight: var(--app-fw-medium); margin: 0 0 var(--app-space); }

p, li { color: var(--app-text); }

.muted { color: var(--app-text-muted); }

a { color: var(--app-primary); text-decoration: none; }

a:hover { text-decoration: underline; }

.surface {

background: var(--app-surface);

border: 1px solid var(--app-border);

border-radius: var(--app-radius);

}

/\* Buttons (example) \*/

.btn {

display: inline-flex; align-items: center; gap: 8px;

padding: 10px 14px;

border-radius: var(--app-radius);

border: 1px solid var(--app-border);

background: var(--app-surface);

color: var(--app-text);

}

.btn-primary {

background: var(--app-primary);

color: var(--app-primary-contrast);

border-color: transparent;

}

/\* Make Carbon header/footer work with your layout if you use the earlier layout CSS \*/

**4) Switching themes**

Because tokens are applied at the AppContainer, you can expose a simple switcher:

// Example somewhere in your UI

import { useState } from "react";

import AppContainer from "@/components/containers/AppContainer";

export default function RootLayout({ children }: { children: React.ReactNode }) {

const [theme, setTheme] = useState<"lasaniLight" | "lasaniDark">("lasaniLight");

return (

<html lang="en">

<body>

<AppContainer theme={theme}>

{/\* … app … \*/}

{/\* Toggle \*/}

<button className="btn" onClick={() => setTheme(t => t === "lasaniLight" ? "lasaniDark" : "lasaniLight")}>

Toggle theme

</button>

{children}

</AppContainer>

</body>

</html>

);

}

**What this gives you**

* **Non-Carbon or hybrid themes**: total control via your token object.
* **Typography**: fluid scale + weights + family, controlled by variables.
* **Palette**: app variables + optional **mapping into Carbon’s variables** so Carbon components auto-adopt your theme.
* **One source of truth** at **AppContainer**. Swapping themes = swapping one object.

If you want, I can add a **registry + JSON theme loader** so you can store themes in a CMS and apply them at runtime without code changes.

Awesome — here’s a **drop-in “Theme Registry + JSON Loader”** so you can keep themes in a CMS (or a static JSON file), load them at runtime, and apply them globally from **AppContainer** without code changes.

**1) Theme types (tokens)**

/types/theme.ts

export type ThemeName = "g10" | "g100" | "lasaniLight" | "lasaniDark" | string;

export interface ThemeTokens {

name: ThemeName;

colors: {

bg: string;

surface: string;

text: string;

textMuted: string;

primary: string;

primaryContrast: string;

border: string;

};

font: {

family: string;

weightRegular: number;

weightMedium: number;

weightBold: number;

sizeBody: string; // "clamp(14px,1.05vw,16px)"

sizeH1: string;

sizeH2: string;

sizeH3: string;

lineHeight: number;

};

radius: string; // "12px"

spacing: number; // base spacing in px

}

**2) Minimal runtime validator (no extra deps)**

/lib/theme/validateTheme.ts

import type { ThemeTokens } from "@/types/theme";

export function isThemeTokens(x: any): x is ThemeTokens {

try {

return (

typeof x?.name === "string" &&

typeof x?.colors?.bg === "string" &&

typeof x?.colors?.surface === "string" &&

typeof x?.colors?.text === "string" &&

typeof x?.colors?.primary === "string" &&

typeof x?.font?.family === "string" &&

typeof x?.font?.sizeBody === "string" &&

typeof x?.font?.sizeH1 === "string" &&

typeof x?.font?.sizeH2 === "string" &&

typeof x?.font?.sizeH3 === "string" &&

typeof x?.font?.lineHeight === "number" &&

typeof x?.radius === "string" &&

typeof x?.spacing === "number"

);

} catch {

return false;

}

}

**3) Built-in defaults + Registry (with safe register/resolve)**

/lib/theme/registry.ts

import type { ThemeTokens, ThemeName } from "@/types/theme";

export const defaults: Record<ThemeName, ThemeTokens> = {

lasaniLight: {

name: "lasaniLight",

colors: {

bg: "#ffffff",

surface: "#f5f5f7",

text: "#1d1d1f",

textMuted: "#6e6e73",

primary: "#0f62fe",

primaryContrast: "#ffffff",

border: "#e5e5ea",

},

font: {

family:

'-apple-system,BlinkMacSystemFont,"Segoe UI",Roboto,Helvetica,Arial,sans-serif',

weightRegular: 400,

weightMedium: 500,

weightBold: 600,

sizeBody: "clamp(14px,1.05vw,16px)",

sizeH1: "clamp(28px,3.2vw,40px)",

sizeH2: "clamp(22px,2.2vw,28px)",

sizeH3: "clamp(18px,1.6vw,22px)",

lineHeight: 1.45,

},

radius: "12px",

spacing: 8,

},

lasaniDark: {

name: "lasaniDark",

colors: {

bg: "#161616",

surface: "#262626",

text: "#f4f4f4",

textMuted: "#b0b0b0",

primary: "#78a9ff",

primaryContrast: "#161616",

border: "#393939",

},

font: {

family:

'-apple-system,BlinkMacSystemFont,"Segoe UI",Roboto,Helvetica,Arial,sans-serif',

weightRegular: 400,

weightMedium: 500,

weightBold: 600,

sizeBody: "clamp(14px,1.05vw,16px)",

sizeH1: "clamp(28px,3.2vw,40px)",

sizeH2: "clamp(22px,2.2vw,28px)",

sizeH3: "clamp(18px,1.6vw,22px)",

lineHeight: 1.45,

},

radius: "12px",

spacing: 8,

},

};

const REGISTRY = new Map<ThemeName, ThemeTokens>(

Object.entries(defaults) as [ThemeName, ThemeTokens][]

);

export function registerTheme(tokens: ThemeTokens) {

REGISTRY.set(tokens.name, tokens);

}

export function getTheme(name: ThemeName): ThemeTokens | undefined {

return REGISTRY.get(name);

}

export function getAllThemes(): ThemeTokens[] {

return [...REGISTRY.values()];

}

**4) Loader from JSON (CMS or static URL)**

/lib/theme/loader.ts

import type { ThemeTokens, ThemeName } from "@/types/theme";

import { isThemeTokens } from "./validateTheme";

import { registerTheme } from "./registry";

/\*\* Accepts:

\* - URL returning an array of ThemeTokens

\* - URL returning an object { themes: ThemeTokens[] }

\*/

export async function loadThemesFromUrl(url: string): Promise<ThemeTokens[]> {

const res = await fetch(url, { cache: "no-store" });

if (!res.ok) throw new Error(`Theme fetch failed: ${res.status}`);

const data = await res.json();

const themes: ThemeTokens[] = Array.isArray(data) ? data : data?.themes ?? [];

const valid = themes.filter(isThemeTokens);

valid.forEach(registerTheme);

return valid;

}

/\*\* Resolve a theme by name, trying registry first, then optional fallback URL. \*/

export async function resolveTheme(

name: ThemeName,

options?: { fallbackUrl?: string }

): Promise<ThemeTokens | undefined> {

const { getTheme } = await import("./registry");

const local = getTheme(name);

if (local) return local;

if (options?.fallbackUrl) {

const loaded = await loadThemesFromUrl(options.fallbackUrl);

return loaded.find((t) => t.name === name);

}

return undefined;

}

**5) AppContainer that can take themeName or themeUrl**

/components/containers/AppContainer.tsx

"use client";

import "@carbon/styles/css/styles.css";

import "@/app/globals.scss";

import { ReactNode, useEffect, useState } from "react";

import type { ThemeTokens, ThemeName } from "@/types/theme";

import { defaults, getTheme, registerTheme } from "@/lib/theme/registry";

import { resolveTheme, loadThemesFromUrl } from "@/lib/theme/loader";

function applyCSSVariables(tokens: ThemeTokens) {

const r = document.documentElement;

r.setAttribute("data-theme", tokens.name);

// core variables

r.style.setProperty("--app-bg", tokens.colors.bg);

r.style.setProperty("--app-surface", tokens.colors.surface);

r.style.setProperty("--app-text", tokens.colors.text);

r.style.setProperty("--app-text-muted", tokens.colors.textMuted);

r.style.setProperty("--app-primary", tokens.colors.primary);

r.style.setProperty("--app-primary-contrast", tokens.colors.primaryContrast);

r.style.setProperty("--app-border", tokens.colors.border);

r.style.setProperty("--app-font-family", tokens.font.family);

r.style.setProperty("--app-fs-body", tokens.font.sizeBody);

r.style.setProperty("--app-fs-h1", tokens.font.sizeH1);

r.style.setProperty("--app-fs-h2", tokens.font.sizeH2);

r.style.setProperty("--app-fs-h3", tokens.font.sizeH3);

r.style.setProperty("--app-line", String(tokens.font.lineHeight));

r.style.setProperty("--app-fw-regular", String(tokens.font.weightRegular));

r.style.setProperty("--app-fw-medium", String(tokens.font.weightMedium));

r.style.setProperty("--app-fw-bold", String(tokens.font.weightBold));

r.style.setProperty("--app-radius", tokens.radius);

r.style.setProperty("--app-space", `${tokens.spacing}px`);

// map a few Carbon vars so Carbon widgets adopt palette

r.style.setProperty("--cds-background", tokens.colors.bg);

r.style.setProperty("--cds-layer", tokens.colors.surface);

r.style.setProperty("--cds-text-primary", tokens.colors.text);

r.style.setProperty("--cds-text-secondary", tokens.colors.textMuted);

r.style.setProperty("--cds-link-primary", tokens.colors.primary);

}

export default function AppContainer({

children,

themeName = "lasaniLight",

themeUrl, // optional URL to JSON from CMS

preloadAll = false, // if true, load all themes from URL on mount

}: {

children: ReactNode;

themeName?: ThemeName;

themeUrl?: string;

preloadAll?: boolean;

}) {

const [active, setActive] = useState<ThemeTokens>(defaults.lasaniLight);

useEffect(() => {

let cancelled = false;

async function init() {

try {

if (themeUrl && preloadAll) await loadThemesFromUrl(themeUrl);

let t = getTheme(themeName);

if (!t && themeUrl) t = await resolveTheme(themeName, { fallbackUrl: themeUrl });

// final fallback -> default

t ??= defaults.lasaniLight;

if (!cancelled) {

setActive(t);

applyCSSVariables(t);

// remember user choice (optional)

localStorage.setItem("theme:name", t.name);

}

} catch (e) {

console.error("Theme init failed", e);

if (!cancelled) {

setActive(defaults.lasaniLight);

applyCSSVariables(defaults.lasaniLight);

}

}

}

// allow persisted user choice

const saved = typeof window !== "undefined" ? localStorage.getItem("theme:name") : null;

if (saved && getTheme(saved)) registerTheme(getTheme(saved)!);

init();

return () => { cancelled = true; };

}, [themeName, themeUrl, preloadAll]);

return <>{children}</>;

}

**6) Example themes JSON (host in CMS or public/themes.json)**

/public/themes.json

{

"themes": [

{

"name": "lasaniLight",

"colors": {

"bg": "#ffffff",

"surface": "#f5f5f7",

"text": "#1d1d1f",

"textMuted": "#6e6e73",

"primary": "#0f62fe",

"primaryContrast": "#ffffff",

"border": "#e5e5ea"

},

"font": {

"family": "-apple-system,BlinkMacSystemFont,'Segoe UI',Roboto,Helvetica,Arial,sans-serif",

"weightRegular": 400,

"weightMedium": 500,

"weightBold": 600,

"sizeBody": "clamp(14px,1.05vw,16px)",

"sizeH1": "clamp(28px,3.2vw,40px)",

"sizeH2": "clamp(22px,2.2vw,28px)",

"sizeH3": "clamp(18px,1.6vw,22px)",

"lineHeight": 1.45

},

"radius": "12px",

"spacing": 8

},

{

"name": "lasaniDark",

"colors": {

"bg": "#161616",

"surface": "#262626",

"text": "#f4f4f4",

"textMuted": "#b0b0b0",

"primary": "#78a9ff",

"primaryContrast": "#161616",

"border": "#393939"

},

"font": {

"family": "-apple-system,BlinkMacSystemFont,'Segoe UI',Roboto,Helvetica,Arial,sans-serif",

"weightRegular": 400,

"weightMedium": 500,

"weightBold": 600,

"sizeBody": "clamp(14px,1.05vw,16px)",

"sizeH1": "clamp(28px,3.2vw,40px)",

"sizeH2": "clamp(22px,2.2vw,28px)",

"sizeH3": "clamp(18px,1.6vw,22px)",

"lineHeight": 1.45

},

"radius": "12px",

"spacing": 8

}

]

}

If your CMS returns an array directly, that works too.

7) Use it in your **app/layout.tsx**

// app/layout.tsx

import AppContainer from "@/components/containers/AppContainer";

export default function RootLayout({ children }: { children: React.ReactNode }) {

return (

<html lang="en">

<body>

{/\* themeUrl can be /themes.json (static) or a CMS endpoint \*/}

<AppContainer themeUrl="/themes.json" preloadAll themeName="lasaniLight">

{children}

</AppContainer>

</body>

</html>

);

}

8) Optional: a simple **theme switcher** anywhere

"use client";

import { useState, useEffect } from "react";

import { loadThemesFromUrl } from "@/lib/theme/loader";

import { getAllThemes } from "@/lib/theme/registry";

export default function ThemeSwitcher() {

const [names, setNames] = useState<string[]>([]);

useEffect(() => {

// ensure loaded if coming from CMS

loadThemesFromUrl("/themes.json").finally(() => {

setNames(getAllThemes().map((t) => t.name));

});

}, []);

return (

<select

defaultValue={localStorage.getItem("theme:name") || "lasaniLight"}

onChange={(e) => {

localStorage.setItem("theme:name", e.target.value);

location.reload(); // simplest global re-apply

}}

>

{names.map((n) => (

<option key={n} value={n}>{n}</option>

))}

</select>

);

}

**How this answers your goals**

* **Non-Carbon or custom themes**: stored as JSON tokens, not code.
* **Typography control**: all sizes/weights/family via tokens → CSS variables → global styles.
* **Color palettes control**: tokens → CSS variables → optional mapping to Carbon CSS variables so Carbon components inherit your colors automatically.
* **AppContainer-level control**: one place to resolve, load, validate, register, and apply themes.

If you tell me the CMS you plan to use (Contentful, Sanity, Strapi, headless WP), I’ll give you the exact shape for its API response and a minimal /app/api/themes/route.ts proxy that caches the JSON.

pls note following 10 themes .. Pls let what you will prepare

|  |  |
| --- | --- |
| # | Name |
| 1. Material Design (Google) | A very influential system introduced by Google providing visual, interaction and motion guidelines across platforms (web, mobile). It includes component libraries, style guides, and is widely adopted. ([DesignRush](https://www.designrush.com/best-designs/websites/trends/design-system-examples?utm_source=chatgpt.com" \o "10 Best Design System Examples for 2025 | DesignRush)) |
| 2. Carbon Design System (IBM) | Created by IBM, this open-source system emphasises enterprise-scale UI, theming with design tokens, and strong scalability. ([Wikipedia](https://en.wikipedia.org/wiki/Carbon_Design_System?utm_source=chatgpt.com)) |
| 3. Fluent Design System (Microsoft) | From Microsoft, this system emphasises depth, motion, translucency and scale across devices — good for modern, dynamic UIs. ([Wikipedia](https://en.wikipedia.org/wiki/Fluent_Design_System?utm_source=chatgpt.com)) |
| 4. Apple Human Interface Guidelines (Apple) | Developed by Apple Inc., this is more than just a “theme” — it’s a full set of guidelines for building clean, intuitive experiences on Apple platforms. ([DesignRush](https://www.designrush.com/best-designs/websites/trends/design-system-examples?utm_source=chatgpt.com" \o "10 Best Design System Examples for 2025 | DesignRush)) |
| 5. Atlassian Design System | Built by Atlassian Corporation Plc for their suite of tools; strong at maintaining consistency across many products and handling complex workflows. ([Built In](https://builtin.com/articles/design-system?utm_source=chatgpt.com)) |
| 6. Adobe Spectrum (Adobe) | From Adobe Inc.—aimed at providing cohesive UI across multiple Adobe applications; a good reference for rich interfaces. ([Built In](https://builtin.com/articles/design-system?utm_source=chatgpt.com)) |
| 7. Shopify Polaris | Created by Shopify Inc. for e-commerce contexts; strong for building merchant dashboards or storefront UIs. ([Built In](https://builtin.com/articles/design-system?utm_source=chatgpt.com)) |
| 8. Mailchimp Pattern Library | From Mailchimp—more lightweight than some others, but very good for web/marketing interfaces and pattern libraries. ([Built In](https://builtin.com/articles/design-system?utm_source=chatgpt.com)) |
| 9. Uber Base Design System | From Uber Technologies Inc., designed for scaling across web and mobile with flexible components and strong customization. ([Built In](https://builtin.com/articles/design-system?utm_source=chatgpt.com)) |
| 10. Audi Design System | From Audi AG—good for unified brand experience across digital + physical touchpoints (which may be relevant if your restaurant brand extends into kiosks/cloud). ([Built In](https://builtin.com/articles/design-system?utm_source=chatgpt.com)) |

Got it. I’ll set you up so those 10 “themes” (really, design systems) can be plugged into your app as **first-class themes**—without locking you to any one library.

Below is what I’ll prepare for you and how it snaps into the AppContainer you already have.

**What I’ll deliver**

**1) A unified theme spec (tokens) that can represent all 10**

Extended from what we built:

// /types/theme.ts

export type ThemeName =

| "material" | "carbon" | "fluent" | "apple"

| "atlassian" | "spectrum" | "polaris"

| "mailchimp" | "uberBase" | "audi"

| string;

export interface ThemeTokens {

name: ThemeName;

// Color primitives

colors: {

bg: string; // page background

surface: string; // cards / layers

text: string;

textMuted: string;

primary: string; // brand / action

primaryContrast: string;

border: string;

// Optional accents & states

success?: string; warning?: string; danger?: string; info?: string;

link?: string; linkHover?: string;

// Elevation overlays (for Fluent/Material/Apple)

overlay?: string;

};

// Typography

font: {

family: string;

weightRegular: number; weightMedium: number; weightBold: number;

sizeBody: string; sizeH1: string; sizeH2: string; sizeH3: string;

lineHeight: number;

letterSpacing?: number;

};

// Shape & motion

radius: string; // e.g., Material 12px, Apple 12–16px

spacing: number; // base unit (px)

shadowSm?: string; shadowMd?: string; shadowLg?: string; // Fluent/Material depth

motion?: { durationFast?: string; durationBase?: string; easing?: string };

}

**2) A theme registry with 10 JSON theme files**

Each design system gets a JSON token file in /public/themes/<name>.json and a combined /public/themes.json. Examples:

* /public/themes/material.json
* /public/themes/carbon.json
* …(all ten)
* /public/themes.json → { "themes": [ … ] }

Each file uses **approximated, license-safe tokens** (not copy/paste from proprietary libs) aligned with the brand philosophies (colors, rounding, motion).

I’ll include realistic defaults for:

* **Material**: bold primary, elevation shadows, 12px radius, Google font stack.
* **Fluent**: soft shadows, acrylic overlay, 8px radius, Segoe UI stack.
* **Apple HIG**: high contrast text, large tracking, 12–16px radius feel, SF stack.
* **Carbon (IBM)**: g10-like palette & spacing rhythm.
* **Atlassian, Spectrum, Polaris, Mailchimp, Uber Base, Audi**: tasteful approximations consistent with their public look/feel.

You can change any value later without touching code.

// /lib/theme/adapter.ts

import type { ThemeTokens } from "@/types/theme";

export function applyCSSVariables(tokens: ThemeTokens) {

const r = document.documentElement;

r.setAttribute("data-theme", tokens.name);

// App variables (used by globals.scss)

r.style.setProperty("--app-bg", tokens.colors.bg);

r.style.setProperty("--app-surface", tokens.colors.surface);

r.style.setProperty("--app-text", tokens.colors.text);

r.style.setProperty("--app-text-muted", tokens.colors.textMuted);

r.style.setProperty("--app-primary", tokens.colors.primary);

r.style.setProperty("--app-primary-contrast", tokens.colors.primaryContrast);

r.style.setProperty("--app-border", tokens.colors.border);

r.style.setProperty("--app-font-family", tokens.font.family);

r.style.setProperty("--app-fs-body", tokens.font.sizeBody);

r.style.setProperty("--app-fs-h1", tokens.font.sizeH1);

r.style.setProperty("--app-fs-h2", tokens.font.sizeH2);

r.style.setProperty("--app-fs-h3", tokens.font.sizeH3);

r.style.setProperty("--app-line", String(tokens.font.lineHeight));

r.style.setProperty("--app-fw-regular", String(tokens.font.weightRegular));

r.style.setProperty("--app-fw-medium", String(tokens.font.weightMedium));

r.style.setProperty("--app-fw-bold", String(tokens.font.weightBold));

r.style.setProperty("--app-radius", tokens.radius);

r.style.setProperty("--app-space", `${tokens.spacing}px`);

if (tokens.shadowSm) r.style.setProperty("--app-shadow-sm", tokens.shadowSm);

if (tokens.shadowMd) r.style.setProperty("--app-shadow-md", tokens.shadowMd);

if (tokens.shadowLg) r.style.setProperty("--app-shadow-lg", tokens.shadowLg);

// Map to Carbon CSS variables (so Carbon components match)

r.style.setProperty("--cds-background", tokens.colors.bg);

r.style.setProperty("--cds-layer", tokens.colors.surface);

r.style.setProperty("--cds-text-primary", tokens.colors.text);

r.style.setProperty("--cds-text-secondary", tokens.colors.textMuted);

r.style.setProperty("--cds-link-primary", tokens.colors.link ?? tokens.colors.primary);

}

If you ever choose to render **Material UI** on some pages, I’ll include a materialAdapter(tokens) that builds an MUI theme object from the same tokens—no duplication.

**4) AppContainer updated to load from registry/URL (already started)**

You’ll keep:

* themeName (e.g., "material", "fluent", "apple", …)
* themeUrl="/themes.json" (or a CMS endpoint)
* A small runtime validator to ensure the JSON conforms.

AppContainer applies the variables with the adapter above—so your **typography + palette** update globally, and Carbon components inherit automatically.

**5) globals.scss that consumes variables**

Your existing SCSS already does most of this. I’ll add a small typography block:

html, body {

background: var(--app-bg);

color: var(--app-text);

font-family: var(--app-font-family);

font-size: var(--app-fs-body);

line-height: var(--app-line);

}

h1 { font-size: var(--app-fs-h1); font-weight: var(--app-fw-medium); }

h2 { font-size: var(--app-fs-h2); font-weight: var(--app-fw-medium); }

h3 { font-size: var(--app-fs-h3); font-weight: var(--app-fw-medium); }

.surface {

background: var(--app-surface);

border: 1px solid var(--app-border);

border-radius: var(--app-radius);

box-shadow: var(--app-shadow-sm, none);

}

.btn-primary {

background: var(--app-primary);

color: var(--app-primary-contrast);

border-radius: var(--app-radius);

}

**6) Two fully-filled examples (Material + Fluent) to start**

**Material (approx.) — /public/themes/material.json**

{

"name": "material",

"colors": {

"bg": "#FFFFFF",

"surface": "#F7F7F7",

"text": "#1F1F1F",

"textMuted": "#666A6D",

"primary": "#2962FF",

"primaryContrast": "#FFFFFF",

"border": "#E5E7EB",

"link": "#2962FF",

"overlay": "rgba(0,0,0,0.04)"

},

"font": {

"family": "Roboto, -apple-system, BlinkMacSystemFont, 'Segoe UI', Helvetica, Arial, sans-serif",

"weightRegular": 400,

"weightMedium": 500,

"weightBold": 700,

"sizeBody": "clamp(14px, 1.05vw, 16px)",

"sizeH1": "clamp(32px, 3.2vw, 44px)",

"sizeH2": "clamp(24px, 2.4vw, 32px)",

"sizeH3": "clamp(18px, 1.8vw, 24px)",

"lineHeight": 1.5

},

"radius": "12px",

"spacing": 8,

"shadowSm": "0 1px 2px rgba(0,0,0,.08)",

"shadowMd": "0 3px 8px rgba(0,0,0,.12)",

"shadowLg": "0 10px 24px rgba(0,0,0,.18)"

}

**Fluent (approx.) — /public/themes/fluent.json**

{

"name": "fluent",

"colors": {

"bg": "#FFFFFF",

"surface": "#FAFAFA",

"text": "#1B1B1B",

"textMuted": "#70757A",

"primary": "#115EA3",

"primaryContrast": "#FFFFFF",

"border": "#E0E0E0",

"link": "#115EA3",

"overlay": "rgba(255,255,255,0.6)"

},

"font": {

"family": "'Segoe UI', -apple-system, BlinkMacSystemFont, Roboto, Helvetica, Arial, sans-serif",

"weightRegular": 400,

"weightMedium": 600,

"weightBold": 700,

"sizeBody": "clamp(14px, 1.05vw, 16px)",

"sizeH1": "clamp(30px, 3vw, 40px)",

"sizeH2": "clamp(22px, 2.2vw, 28px)",

"sizeH3": "clamp(18px, 1.6vw, 22px)",

"lineHeight": 1.45

},

"radius": "8px",

"spacing": 8,

"shadowSm": "0 1px 2px rgba(0,0,0,.06)",

"shadowMd": "0 4px 12px rgba(0,0,0,.10)",

"shadowLg": "0 8px 24px rgba(0,0,0,.14)"

}

**7) How you switch themes (no code changes)**

**In app/layout.tsx:**

<AppContainer themeUrl="/themes.json" themeName="material" preloadAll>

{children}

</AppContainer>

**Or add a simple selector that sets localStorage.setItem("theme:name", "<name>") and reloads.**

**Summary**

* **Single token spec covers color, type, shape, motion.**
* **10 JSON themes (one file each + combined file) ready to load from /themes.json or a CMS.**
* **Adapter publishes CSS variables and maps to Carbon variables.**
* **Globals consume variables for typography/palette → whole app updates instantly.**
* **Optional per-library adapters (e.g., Material UI) if you ever mix stacks.**

**If you want, tell me which 3 themes you’ll actually ship first (e.g., Apple, Carbon, Fluent). I’ll tailor those JSONs to look great with your current header/footer immediately.**

1. Apple Human Interface Guidelines (Apple) 2. Fluent Design System (Microsoft) 3. Materials

**Absolutely—here’s a clean, complete checklist of all files to create or update to implement the new container-based architecture + theme system (Apple / Fluent / Material), including the layout + header/footer behavior we set up.**

**✅ Core app files (create / update)**

1. **app/layout.tsx *(update)***

* **Mounts AppContainer once (global providers + theme loader).**
* **Ensures CSS import order: Carbon first, then globals.scss.**

1. **app/page.tsx *(create or update)***

* **Uses PageContainer composition (Header + main + Footer).**
* **Example shows HeroContainer + SectionContainer usage.**

1. **app/globals.scss *(create/update with final consolidated CSS)***

* **Fixed header + fixed (auto-height) footer layout.**
* **Main area fills remaining space (min-height: calc(100vh - header - footer)).**
* **Spacing/typography/variables consumption (--app-\*).**
* **Footer grid + header spacing tweaks.**

**🧱 Container components (create)**

1. **components/containers/AppContainer.tsx**

* **Loads Carbon CSS + globals.scss.**
* **Loads theme tokens (from registry/URL), applies CSS variables.**
* **Accepts themeName, themeUrl, preloadAll.**

1. **components/containers/PageContainer.tsx**

* **Orchestrates page frame: HeaderContainer → <main class="main-wrap"> → FooterContainer.**
* **Reads layout config (if you later add LayoutContext overrides).**

1. **components/containers/HeaderContainer.tsx**

* **Carbon header (brand, nav, actions).**
* **Uses classes that our SCSS targets (custom-header, logo-tight).**

1. **components/containers/FooterContainer.tsx**

* **Two-row footer (3 columns + copyright).**
* **Uses ResizeObserver to set --app-footer-height dynamically.**

1. **components/containers/BodyContainer.tsx**

* **Simple wrapper for body content (future CMS can mount sections inside).**

1. **components/containers/HeroContainer.tsx**

* **Hero block (title + optional subtitle).**

1. **components/containers/SectionContainer.tsx**

* **Generic content section with optional heading.**

**🧠 Contexts (create)**

1. **context/ThemeContext.tsx**

* **(Optional if you need reactive theme toggling beyond CSS variables)**
* **For now, theme application is handled in AppContainer; keep this ready if you want UI-based toggles without reloads later.**

1. **context/LayoutContext.tsx**

* **Provides LayoutConfig (header sticky, footer sticky, container max width, default theme name, etc.).**

1. **context/ContentContext.tsx**

* **(Optional) For CMS-driven sections in future (array of blocks).**

**🧾 Types (create)**

1. **types/theme.ts**

* **ThemeName, ThemeTokens interface (colors, typography, spacing, radius, shadows, motion).**

1. **types/layout.ts**

* **LayoutConfig & ThemeName import for container settings.**

1. **types/content.ts**

* **SectionBlock shape (e.g., kind: "hero" | "text" | "grid" | "custom", props).**

**🧩 Theme system (create)**

1. **lib/theme/validateTheme.ts**

* **Minimal runtime validator for theme JSON shape.**

1. **lib/theme/registry.ts**

* **In-memory store of themes: defaults + register/get/getAll helpers.**

1. **lib/theme/loader.ts**

* **Load themes from a URL (/themes.json or your CMS).**
* **Resolve by name with fallback behavior.**

1. **lib/theme/adapter.ts**

* **Applies CSS variables from tokens (--app-\*) and maps key values to Carbon CSS vars (--cds-\*) so Carbon components adopt your theme.**

**These four files power the JSON-driven, AppContainer-level theme control.**

**🎨 Theme JSON assets (create in public/)**

1. **public/themes/apple.json**
2. **public/themes/fluent.json**
3. **public/themes/material.json**

* **Token files exactly as provided (colors, fonts, spacing, shadows, motion).**

1. **public/themes.json *(combined; optional but recommended)***

* **Aggregates all three themes into { "themes": [ … ] } for one-call loading.**

**🖼️ Public assets (create/confirm)**

1. **public/images/logo.png**

* **Header logo (24–32px tall recommended).**

**🧪 Optional utilities (nice-to-have)**

1. **components/ThemeSwitcher.tsx *(optional)***

* **Tiny dropdown to switch themes at runtime by setting localStorage("theme:name") then reload.**
* **Great for verifying Apple / Fluent / Material quickly.**

1. **components/layout/Header.tsx & components/layout/Footer.tsx**

* **If you’re migrating from earlier versions, keep them only if needed; otherwise, use the new containers/\* versions.**

**📦 Implementation order (recommended)**

1. **Styles: app/globals.scss (final consolidated layout + header/footer rules).**
2. **Containers: AppContainer, PageContainer, HeaderContainer, FooterContainer, then HeroContainer, SectionContainer, BodyContainer.**
3. **Types + Theme system: types/\*, then lib/theme/\*.**
4. **Theme JSON: public/themes/\*.json + public/themes.json.**
5. **Core app files: update app/layout.tsx and app/page.tsx to wire everything.**
6. **Assets: public/images/logo.png.**
7. **(Optional) ThemeSwitcher for quick testing.**

**🔧 Critical notes**

* **Import order in app/layout.tsx:**

import "@carbon/styles/css/styles.css"; // Carbon first

import "@/app/globals.scss"; // your overrides last

* **Header/Footer behavior: Header fixed; Footer fixed with auto-height; ResizeObserver sets --app-footer-height so body padding and main min-height are always correct.**
* **Theme control: Switch using themeName="apple" | "fluent" | "material" and/or load from themeUrl="/themes.json". Tokens → CSS variables → app + Carbon components update instantly.**
* **Typography & palette: Fully driven by tokens; no hardcoded colors or font sizes in components.**

**If you want, I can start by generating the first file exactly as code (e.g., app/globals.scss), then we proceed file-by-file in this order.**

**Excellent — this is the core of a truly professional theme system, so let’s unpack it carefully.  
Yes — the JSON theme model we designed is comprehensive enough to cover all typography, color, spacing, and motion needs for your design systems (Apple / Fluent / Material), but it’s intentionally modular so you can extend it later (e.g., to add display sizes, accent palettes, or motion curves).**

**Below is the deep explanation of how typography & color scales are handled inside our theme JSON system — and how the AppContainer + adapter.ts turns that into live CSS variables that scale correctly.**

**🎨 1️⃣ Color Architecture**

**Each theme’s colors block covers the core semantic roles rather than raw hues:**

| **Token** | **Purpose** | **Example** |
| --- | --- | --- |
| **bg** | **Page background** | **#FFFFFF (Apple / Material Light)** |
| **surface** | **Cards, panels, menus** | **#F5F5F7** |
| **text** | **Primary text color** | **#1D1D1F** |
| **textMuted** | **Secondary / placeholder text** | **#6E6E73** |
| **primary** | **Brand or accent color** | **#0071E3 (Apple Blue)** |
| **primaryContrast** | **Text on primary** | **#FFFFFF** |
| **border** | **Divider / outline** | **#E5E5EA** |
| **link** | **Hyperlink color** | **defaults to primary** |
| **overlay** | **Semi-transparent overlays / glass** | **rgba(0,0,0,0.04)** |

**All stateful or contextual colors (success / warning / danger) can be added later if you start implementing status banners or toasts.**

**Inside the theme adapter, each color token is exported as a CSS variable:**

--app-bg: #fff;

--app-surface: #f5f5f7;

--app-text: #1d1d1f;

--app-text-muted: #6e6e73;

--app-primary: #0071e3;

--app-primary-contrast: #fff;

--app-border: #e5e5ea;

--app-link: #0071e3;

**…and mapped to Carbon CSS variables:**

--cds-background: var(--app-bg);

--cds-layer: var(--app-surface);

--cds-text-primary: var(--app-text);

--cds-text-secondary: var(--app-text-muted);

--cds-link-primary: var(--app-link);

**Thus, Carbon components automatically adopt your palette, while your own SCSS (and Tailwind-like classes) read from --app-\*.**

**🔠 2️⃣ Typography Architecture**

**The font object covers family, weights, and scale.  
It’s structured as a typographic *system*, not single values:**

"font": {

"family": "-apple-system, BlinkMacSystemFont, 'SF Pro Text', 'Helvetica Neue', Helvetica, Arial, sans-serif",

"weightRegular": 400,

"weightMedium": 500,

"weightBold": 600,

"sizeBody": "clamp(14px, 1.05vw, 16px)",

"sizeH1": "clamp(30px, 3.0vw, 42px)",

"sizeH2": "clamp(24px, 2.3vw, 30px)",

"sizeH3": "clamp(18px, 1.7vw, 22px)",

"lineHeight": 1.42,

"letterSpacing": 0

}

**✅ Coverage:**

|  |  |  |  |
| --- | --- | --- | --- |
| Token | Role | Example Value | Description |
| family | Font family stack | Apple: SF Pro / Fluent: Segoe UI / Material: Roboto | Inherits across the whole app. |
| weightRegular, weightMedium, weightBold | Core weights | 400 / 500 / 700 | Works with global font-weight mapping. |
| sizeBody | Base body text | clamp(14px, 1.05vw, 16px) | Uses CSS clamp() for fluid scale. |
| sizeH1, sizeH2, sizeH3 | Heading scales |  | Fluid responsive heading sizes. |
| lineHeight | Default leading | 1.45 | Multiplied automatically in SCSS. |
| letterSpacing | Optical spacing | optional | Useful for uppercase or Apple HIG body text. |

**The fluid typography (clamp()) means text scales smoothly from small screens to desktops *without* media queries.**

**⚙️ 3️⃣ Applying Typography (how scale + weight are used)**

The adapter writes variables:

--app-font-family: -apple-system, ...;

--app-fw-regular: 400;

--app-fw-medium: 500;

--app-fw-bold: 700;

--app-fs-body: clamp(14px, 1.05vw, 16px);

--app-fs-h1: clamp(30px, 3vw, 42px);

--app-fs-h2: clamp(24px, 2vw, 30px);

--app-fs-h3: clamp(18px, 1.6vw, 22px);

--app-line: 1.45;

**Your globals.scss then uses these variables consistently:**

body {

font-family: var(--app-font-family);

font-weight: var(--app-fw-regular);

font-size: var(--app-fs-body);

line-height: var(--app-line);

}

h1, .h1 {

font-size: var(--app-fs-h1);

font-weight: var(--app-fw-bold);

line-height: calc(var(--app-line) \* 1.1);

}

h2, .h2 {

font-size: var(--app-fs-h2);

font-weight: var(--app-fw-medium);

}

h3, .h3 {

font-size: var(--app-fs-h3);

font-weight: var(--app-fw-medium);

}

**Now, changing *only the theme JSON* updates:**

* **Fonts (SF Pro → Segoe UI → Roboto)**
* **Text sizes and rhythm**
* **Heading hierarchy**
* **Color and contrast for text**

**…without touching any component.**

**📏 4️⃣ Scales & Responsive Behavior**

**We’ve chosen clamp(min, vw-based, max) for every text size.  
For example:**

"sizeH1": "clamp(30px, 3vw, 42px)"

**This formula ensures:**

* **Minimum size at smallest breakpoints.**
* **Smooth growth through vw unit scaling.**
* **Maximum cap for large screens.**

**This replaces old breakpoints (like @media (min-width: 768px)).**

**🧱 5️⃣ Extensibility for future refinements**

**You can easily extend the JSON for:**

"typescale": {

"display": "clamp(48px,4vw,64px)",

"caption": "clamp(12px,0.9vw,14px)",

"overline": "clamp(10px,0.8vw,12px)"

},

"colors": {

"accent": "#FF4081",

"neutral": "#CCCCCC"

}

**…and the adapter will simply propagate those to CSS variables.**

**We can later add a themeTypeScale.ts helper that generates accessible modular scales (1.25 ratio for Apple, 1.33 for Material, etc.) from a base sizeBody.**

**🧮 6️⃣ How weights and scales interact in live UI**

* **Apple → lighter optical weight (SF Pro Text, 400–600)**
* **Fluent → slightly heavier (Segoe UI 400/600/700)**
* **Material → Roboto’s balanced (400/500/700)**

**Because weights are separate tokens, each system’s intended typographic feel is preserved, even if the body size is the same.**

**🎯 7️⃣ Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Purpose | Where defined | Controlled by |
| Theme JSON | **Declarative color, type, spacing tokens** | **/public/themes/\*.json** | **Designer / CMS** |
| Theme Adapter | **Translates tokens → CSS vars** | **lib/theme/adapter.ts** | **Code** |
| SCSS | **Consumes vars** | **app/globals.scss** | **Style layer** |
| Components | **Render semantically (h1, p, .btn-primary)** | **JSX** | **Auto-inherit** |
| Runtime | **Applies vars per selected theme** | **AppContainer** | **User / UI toggle** |

**If you like, the next step is to add full typography scales (Display / Headline / Title / Body / Caption) for these three themes so your entire system—from hero titles down to labels—can respond perfectly.  
Would you like me to extend the JSON spec and SCSS to include that full five-tier typography scale?**

you define single material theme json .. Will it cover light and dark

**ChatGPT said:**

**Thought for 9s**

**Short answer: yes—one Material JSON can cover both light and dark.  
Do it by putting both modes in the same file and letting AppContainer pick the active mode (light | dark | system) at runtime.**

**Here’s exactly how.**

**1) Single-file, multi-mode Material theme JSON**

{

"name": "material",

"font": {

"family": "Roboto, -apple-system, BlinkMacSystemFont, 'Segoe UI', Helvetica, Arial, sans-serif",

"weightRegular": 400,

"weightMedium": 500,

"weightBold": 700,

"sizeBody": "clamp(14px, 1.05vw, 16px)",

"sizeH1": "clamp(32px, 3.2vw, 44px)",

"sizeH2": "clamp(24px, 2.4vw, 32px)",

"sizeH3": "clamp(18px, 1.8vw, 24px)",

"lineHeight": 1.5

},

"radius": "12px",

"spacing": 8,

"shadowSm": "0 1px 2px rgba(0,0,0,.08)",

"shadowMd": "0 3px 8px rgba(0,0,0,.12)",

"shadowLg": "0 10px 24px rgba(0,0,0,.18)",

"modes": {

"light": {

"colors": {

"bg": "#FFFFFF",

"surface": "#F7F7F7",

"text": "#1F1F1F",

"textMuted": "#666A6D",

"primary": "#2962FF",

"primaryContrast": "#FFFFFF",

"border": "#E5E7EB",

"link": "#2962FF",

"overlay": "rgba(0,0,0,0.04)"

}

},

"dark": {

"colors": {

"bg": "#121212",

"surface": "#1E1E1E",

"text": "#ECECEC",

"textMuted": "#B5B5B5",

"primary": "#82B1FF",

"primaryContrast": "#0D0D0D",

"border": "#2A2A2A",

"link": "#82B1FF",

"overlay": "rgba(255,255,255,0.06)"

},

"shadowSm": "0 1px 2px rgba(0,0,0,.5)",

"shadowMd": "0 6px 12px rgba(0,0,0,.55)",

"shadowLg": "0 12px 28px rgba(0,0,0,.6)"

}

}

}

** Shared tokens (typography, radius, spacing, base shadows) sit at the root.**

** Per-mode overrides (colors, optional shadows, etc.) live under modes.light / modes.dark.**

**2) Minimal type tweak (optional)**

**If you keep TypeScript types:**

// types/theme.ts

export interface ThemeTokens {

name: string;

font: { /\* …as before… \*/ };

radius: string;

spacing: number;

shadowSm?: string; shadowMd?: string; shadowLg?: string;

// single-file multi-mode:

modes: {

light: Partial<Pick<ThemeTokens, "shadowSm" | "shadowMd" | "shadowLg">> & {

colors: ThemeColors

};

dark: Partial<Pick<ThemeTokens, "shadowSm" | "shadowMd" | "shadowLg">> & {

colors: ThemeColors

};

};

}

export interface ThemeColors {

bg: string; surface: string; text: string; textMuted: string;

primary: string; primaryContrast: string; border: string;

link?: string; overlay?: string;

}

**(If you don’t want to change types now, you can keep them looser; the logic below still applies.)**

**3) AppContainer / adapter changes (choose the active mode)**

* **Add a themeMode prop: 'light' | 'dark' | 'system' (default 'light').**
* **For 'system', read window.matchMedia('(prefers-color-scheme: dark)').**

**Merge logic (pseudo-code used in adapter):**

function pickMode(tokens, mode: 'light' | 'dark') {

// Merge root + selected mode (mode can override shadows etc.)

return {

...tokens,

colors: tokens.modes[mode].colors,

shadowSm: tokens.modes[mode].shadowSm ?? tokens.shadowSm,

shadowMd: tokens.modes[mode].shadowMd ?? tokens.shadowMd,

shadowLg: tokens.modes[mode].shadowLg ?? tokens.shadowLg,

};

}

**Then call your existing applyCSSVariables(merged) and also set attributes for debugging/styling hooks:**

document.documentElement.setAttribute("data-theme", tokens.name);

document.documentElement.setAttribute("data-mode", mode);

**System mode support:**

const media = window.matchMedia('(prefers-color-scheme: dark)');

const resolvedMode = themeMode === 'system' ? (media.matches ? 'dark' : 'light') : themeMode;

// add listener to update on OS changes

media.addEventListener?.('change', e => applyMode(e.matches ? 'dark' : 'light'));

**4) Globals / components need no changes**

**Your globals.scss already consumes variables like:**

html, body {

background: var(--app-bg);

color: var(--app-text);

}

a { color: var(--app-primary); }

.surface { background: var(--app-surface); border-color: var(--app-border); }

**When mode switches, the adapter updates those CSS variables → UI flips instantly.**

**5) Carbon component colors in dark mode**

**We already map your tokens to Carbon CSS vars:**

--cds-background: var(--app-bg);

--cds-layer: var(--app-surface);

--cds-text-primary: var(--app-text);

--cds-text-secondary: var(--app-text-muted);

--cds-link-primary: var(--app-primary);

**So Carbon components follow the mode automatically.**

**6) How to use**

// app/layout.tsx

import AppContainer from "@/components/containers/AppContainer";

export default function RootLayout({ children }: { children: React.ReactNode }) {

return (

<html lang="en">

<body>

<AppContainer

themeUrl="/themes.json" // contains apple, fluent, material (multi-mode material)

themeName="material"

themeMode="system" // 'light' | 'dark' | 'system'

preloadAll

>

{children}

</AppContainer>

</body>

</html>

);

}

**Optional: add a UI toggle that sets localStorage('theme:mode') to 'light' | 'dark' | 'system', then re-applies without reload.**

**TL;DR**

* **One Material JSON can indeed handle both light and dark by nesting modes.light and modes.dark.**
* **AppContainer picks the mode (light / dark / system), merges mode overrides with shared tokens, and publishes CSS variables.**
* **Your SCSS and Carbon components auto-adopt the active mode—no component changes needed.**