Project Example: Full Portfolio + Blog + Dashboard with App Router

You've been assigned to build a **personal website** that includes:

- A **public portfolio** (Home, Projects, About, Contact)
- A **blog** (SEO, markdown, static pages)
- An **admin dashboard** (auth-protected, editable content)
- Uses **Next.js App Router**, **Tailwind**, **MDX**, **middleware-based auth**, and **API routes**

🔷 Step-by-Step Architecture Plan

🔹 Step 1: Set Up the App Router Project

```
npx create-next-app@latest my-portfolio --app --typescript
cd my-portfolio
npm install tailwindcss @tailwindcss/typography @tailwindcss/forms @auth/core
```

🔹 Step 2: Setup Folder Structure (Best Practice)

```
src/
├── app/
│   ├── layout.tsx          # Root layout (includes <html> & <body>)
│   ├── page.tsx            # Homepage
│   ├── about/
│   │   └── page.tsx        # About page
│   ├── projects/
│   │   └── page.tsx
│   ├── blog/
│   │   ├── layout.tsx      # Blog section layout
│   │   ├── page.tsx        # Blog home
│   │   └── [slug]/
│   │       └── page.tsx    # Dynamic blog post
│   ├── dashboard/          # Admin dashboard (auth-protected)
│   │   ├── layout.tsx
│   │   ├── page.tsx
│   │   └── edit/
│   │       └── page.tsx
│   ├── api/
│   │   └── contact/route.ts    # API endpoint for contact form
│   ├── not-found.tsx
│   ├── error.tsx
│   └── loading.tsx
├── components/
│   ├── ui/               # Reusable elements like Button, Card
│   ├── layout/             # Navbar, Footer, DashboardLayout
│   └── blog/              # BlogCard, MDXRenderer
├── lib/                 # utils (e.g., fetchers, slug generator)
├── middleware.ts            # Route protection logic
├── styles/              # Tailwind config, global.css
├── hooks/               # useDarkMode, useScroll, etc.
├── types/               # Post, Project, User
└── content/
    ├── blog/            # Markdown blog posts
    └── projects/          # Optional: project data
```

🔹 Step 3: Plan Routing (App Router Logic)

Route Path	File	Behavior
`/`	`app/page.tsx`	Home with animated hero & featured work
`/about`	`app/about/page.tsx`	Static about section
`/projects`	`app/projects/page.tsx`	Projects grid
`/blog`	`app/blog/page.tsx`	Blog listing (uses MDX/Markdown)
`/blog/[slug]`	`app/blog/[slug]/page.tsx`	Dynamic blog content
`/dashboard`	`app/dashboard/page.tsx`	Admin dashboard (protected via middleware)

`/dashboard/edit`	`app/dashboard/edit/page.tsx`	Content editor

🔹 Step 4: Layout Strategy

Global layout (app/layout.tsx):

```
import '../styles/globals.css'
import { Navbar, Footer } from '@/components/layout'

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body className="min-h-screen bg-white text-black">
        <Navbar />
        <main>{children}</main>
        <Footer />
      </body>
    </html>
  )
}
```

Nested layout example (app/blog/layout.tsx):

```
export default function BlogLayout({ children }) {
  return (
    <div className="blog-layout">
      <h1 className="text-2xl font-bold">My Blog</h1>
      {children}
    </div>
  )
}
```

🔹 Step 5: Middleware for Auth

```
// middleware.ts
import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'

export function middleware(request: NextRequest) {
  const token = request.cookies.get('auth_token')
  const isDashboard = request.nextUrl.pathname.startsWith('/dashboard')

  if (isDashboard && !token) {
    return NextResponse.redirect(new URL('/login', request.url))
  }

  return NextResponse.next()
}

export const config = {
  matcher: ['/dashboard/:path*']
}
```

🔹 Step 6: Blog via MDX / Markdown

Use `contentlayer` or manual approach with `fs` + `gray-matter`.

```
// app/blog/[slug]/page.tsx
import { getPostBySlug } from '@/lib/blog'

export default async function BlogPost({ params }) {
  const post = await getPostBySlug(params.slug)
  return <article dangerouslySetInnerHTML={{ __html: post.content }} />
}
```

🔹 Step 7: API Routes (e.g., Contact Form)

```
// app/api/contact/route.ts
export async function POST(req: Request) {
  const body = await req.json()
  const { name, email, message } = body
  // Save to DB or send email
  return new Response(JSON.stringify({ success: true }), { status: 200 })
}
```

🔹 Step 8: Final Steps for Production Readiness

- Add SEO via `metadata` export:

```
export const metadata = {
  title: 'Home | My Portfolio',
  description: 'Showcase of my work and blog'
}
```

- Optimize images via `next/image`
- Configure `next.config.js`:

```
images: { domains: ['mycms.com'] }
```

- Set up CI/CD with GitHub + Vercel
- Add `.env` secrets for tokens, keys, endpoints

✅ Summary: Your Flow in Interview Terms

> **"How would you architect a portfolio + blog + dashboard app?"**

Answer:
I'd use App Router for clean layouts, nested routing, and React Server Components. I'd split public and private routes (`dashboard`) using middleware, and structure code into domain-driven folders: `app`, `components`, `lib`, `hooks`, `types`. Blog content is pulled via MDX, and API endpoints like `/api/contact` handle form logic server-side. Layout.tsx manages global design and SSR ensures SEO readiness.

Perfect! Let's go deep into the **core concept of `layout.tsx` in App Router** — one of the most important architectural building blocks in modern Next.js.

🔷 Layout in Next.js App Router – Explained in 5 Clear Steps

✅ 1. Definition (What is `layout.tsx`?)

In **Next.js App Router**, a `layout.tsx` is a **shared UI wrapper** that persists across route changes. It wraps pages with consistent elements like navigation bars, footers, sidebars, or global providers.

- Every folder in the `/app` directory **can have its own `layout.tsx`**
- These layouts are **nested** and **shared**
- Unlike `pages/_app.tsx`, App Router lets you scope layouts per section

🧠 2. Real-World Analogy

> Think of `layout.tsx` as the **frame of a house**.
> Each room (route) changes, but the walls, ceiling, floor (layout) stay the same.

Deeper Analogy:

- `Root layout.tsx`: The whole house frame – has Navbar, Footer, Theme
- `Dashboard layout.tsx`: A room theme – like an office room layout inside the house
- `page.tsx`: Just the furniture/content in that room

🧪 3. Simple Example

// src/app/layout.tsx

import './globals.css'

import { Navbar } from '@/components/layout/Navbar'

```
import { Footer } from '@/components/layout/Footer'

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>
        <Navbar />
        <main>{children}</main>
        <Footer />
      </body>
    </html>
  )
}
```

- This wraps every page (`page.tsx`) under `/app` with Navbar and Footer.
- You don’t need to re-import Navbar in every route!

🧱 4. Layout Hierarchy Flow (Parent > Child > Page)

```
/app/
├── layout.tsx       → Global layout (Navbar, Footer)
├── blog/
│   ├── layout.tsx   → Blog-specific wrapper (Sidebar, Header)
│   └── [slug]/
│       └── page.tsx  → Actual blog post content
```

Flow:
`Root layout → Blog layout → Blog post page`

✅ You can also use `loading.tsx`, `error.tsx`, `not-found.tsx` alongside layouts!

🎨 5. How to Categorize & Use Styles in Layout

```
styles/
├── globals.css      # Tailwind base, utilities, custom CSS
├── layout.css       # Shared layout-specific styles (body, wrapper)
├── navbar.css       # For Navbar styling
├── dashboard.css    # For dashboard section
```

Import in layout:

```
import '@/styles/globals.css'
import '@/styles/layout.css'
```

✅ Keep **global layout styling** in one file, and split per component/section for maintainability.

💬 Interview Question + Ideal Answer

> ❓**Q: What is `layout.tsx` in App Router and how does it differ from `_app.tsx` in Pages Router?**
> ✅ **A:**
> In App Router, `layout.tsx` replaces `_app.tsx` and allows for **nested, persistent layouts**. Every route folder can define its own layout. It wraps the page with shared UI like navbars or context providers. Unlike `_app.tsx` which is global-only, `layout.tsx` supports **section-scoped structure** — perfect for public vs admin areas.

🧭 Summary Steps to Architect Layout

Step	What You Do	File
1	Define global layout (Navbar/Footer)	`app/layout.tsx`
2	Set up Tailwind/global styles	`styles/globals.css`
3	Add section layout if needed (e.g. blog)	`app/blog/layout.tsx`
4	Use layout structure to avoid repetition	Wrap children via `main`
5	Keep styles modular per layout section	`styles/layout.css` etc.

Would you like me to now move into **SEO, performance, and auth** (Series 4 topics) using this layout foundation?