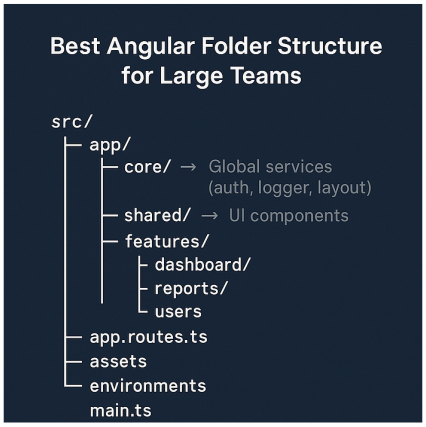
< Go to the original



Best Angular Folder Structure for Large Teams (2025 Guide)

Organizing a large Angular project? Discover the folder structure strategy that helped us scale an Angular app across multiple teams

Angular Adventurer
Follow

a11y-light · May 20, 2025 (Updated: May 20, 2025) · Free: No

A battle-tested guide to scalable Angular architecture, feature-based organization, and team-friendly project structures.

> Non-Member? Click here to read for free

Why Folder Structure Matters in Angular Projects

When you're building a small Angular app, folder structure barely matters. But once you're working with multiple teams, dozens of features, and long-term maintainability in mind, poor structure becomes your biggest bottleneck.

If your Angular app feels like this:

Copy

```
src/
  app/
    user/
    shared/
    utils/
    user2/
    random-services/
```

You're not alone — and this post is for you.

We've worked on enterprise Angular projects across multiple teams. After several failed attempts and painful merge conflicts, we landed on a **folder strategy that scales beautifully**.

This guide shares that exact structure — and why it works.

Angular Folder Structure for Large Teams (2025-Ready)

Here's what our structure looks like:

Copy

```
src/
├── app/
│   ├── core/              → Global services (auth, logger, layout)
│   ├── shared/            → UI-only, reusable components
│   ├── features/          → Business domains (users, reports, dashboard)
│   │   ├── dashboard/
│   │   ├── reports/
│   │   └── users/
│   └── app.routes.ts      → Global routing setup
├── assets/
├── environments/
└── main.ts
```

Core Module in Angular: App-Wide Services & Singletons

Use `core/` for one-time setup code that's needed globally across the app.

Examples:

- Auth service
- App-wide HTTP interceptors
- Logging service
- Layout service
- Shell or base route guards

These services are usually singletons and should be injected via `providedIn: 'root'`.

Avoid putting UI or feature-specific logic here.

Shared Module in Angular: Where to Put UI Components, Pipes & Directives

The `shared/` folder is **strictly for presentational elements**.

Things like:

- UI components (`Button`, `Modal`, `Dropdown`)
- Reusable pipes (`[illegible]`, `[illegible]`)

- Reusable pipes (`formatDate`, `truncate`)
- Directives (`autofocus`, `debounceClick`)

Rule of thumb: If it uses business logic, it doesn't belong here.

Feature Modules: Domain-Based Foldering for Angular Enterprise Projects

The `features/` directory is the heart of your app. Each domain (e.g., `users`, `dashboard`) is organized into its own folder.

Example — `features/reports/`:

Copy

```
reports/
├── components/
├── services/
├── store/         → Component Store / Signals / NGRX
├── reports.routes.ts
├── reports.component.ts
```

Each feature contains:

- Its own routing config
- State management (optional)
- Smart and dumb components
- Scoped services

Every feature is isolated and independently deployable.

Angular Routing Strategy for Scalable Apps

Each feature lazily loads its component or module via routing. Use **standalone components** if possible.

Example:

Copy

```
{
  path: 'reports',
  loadComponent: () =>
    import('./features/reports/reports.component').then(m => m.ReportsComponent),
}
```

This keeps the initial bundle small and allows each team to deploy without stepping on others.

Do's & Don'ts of Angular Folder Organization

Do:

- Separate `core/`, `shared/`, and `features/`
- Use route-based lazy loading
- Keep features isolated
- Match folder names to route paths
- Use feature-level routing files

Don't:

- Dump everything into `app/`
- Put business logic inside `shared/`
- Create catch-all folders like `utils/`
- Cross-import between features
- Over-nest folders (no 6-level-deep trees)

Naming Conventions That Keep Teams Sane

Stick to predictable names inside each feature:

Copy

```
dashboard.component.ts
dashboard.service.ts
dashboard.routes.ts
```

Pair components with their HTML and SCSS in the same folder:

Copy

```
dashboard/
  dashboard.component.ts
  dashboard.component.html
  dashboard.component.scss
```

Bonus Tips for Enterprise Angular Teams

- **Use Barrel Files** only inside a feature folder — avoid `shared/index.ts` exporting everything.
- **Enforce rules with ESLint**: no cross-feature imports, no UI logic in services, etc.
- **Assign folder ownership**: each team owns specific `features/` to avoid merge hell.
- **Document the architecture**: new devs should know where things go instantly.

Final Thoughts

A well-organized Angular project doesn't just look clean — it:

- Reduces onboarding time
- Speeds up PR reviews
- Prevents regressions
- Helps teams scale independently

This folder structure gave our team the clarity we needed to grow fast without drowning in tech debt. Try it out — and adapt it to your organization's size, processes, and team autonomy.

#angular #web-development #scalable-code #frontend-architecture #angular-best-practices