Hybrid Monorepo Structure

Multiple frameworks in one repo. Shared types and contracts across stacks.

#monorepo #multi-framework #polyglot #typescript #api

PNG PDF Copy Prompt

Project Directory

```
platform/
  apps/ Mixed framework…
    customer-web/ Next.js custome…
      src/
      package.json
      next.config.js
    admin-web/ Vue admin dashb…
      src/
      package.json
      vite.config.ts
    mobile/ React Native app
      src/
      package.json
      app.json
  services/ Backend services
    api-gateway/ Main API (Node)
      src/
      package.json
      Dockerfile
    auth-service/ Auth microservi…
      src/
      package.json
      Dockerfile
    worker/ Background jobs…
      main.go
      go.mod
      Dockerfile
  packages/ Shared packages
    types/ Shared TypeScri…
      src/
        api.ts API contracts
      package.json
    validators/ Shared validati…
      src/
        user.ts
      package.json
    config/ Shared tooling …
      eslint/
      tsconfig/
  infrastructure/ IaC and deploym…
    terraform/
    kubernetes/
    docker-compose.yml Local dev
  turbo.json Or nx.json
  package.json
  pnpm-workspace.yaml
  pnpm-lock.yaml
  .gitignore
```

Why This Structure?

Real-world platforms often mix frameworks: React for customers, Vue for internal tools, Go for performance-critical workers. Shared `packages/types/` ensures API contracts are consistent. Each app/service deploys independently while sharing validation and types.

Key Directories

`apps/` - Frontend apps (any framework)

`services/` - Backend services (any language)

`packages/types/` - Shared TypeScript contracts

`infrastructure/` - Deployment and IaC configs

Sharing Strategy

`Types` - Share via `@repo/types` package

`Validation` - Zod schemas in `@repo/validators`

`API contracts` - OpenAPI spec or tRPC router

`Config` - ESLint/TSConfig presets

Cross-Stack Sharing

```
# Import shared types in any app
import type { User } from '@repo/types'
import { userSchema } from '@repo/validators'
```

When To Use This

- Platform with multiple user-facing apps
- Microservices needing shared contracts
- Teams with different framework preferences
- Gradual migration between frameworks

Trade-offs

`CI complexity` - Selective builds per service

`Mixed tooling` - Go + Node need different CI steps

`Onboarding` - More technologies to learn