Phoenix Umbrella Project Structure

Multi-app umbrella for large systems. Separate apps with clear boundaries and shared dependencies.

#phoenix #elixir #umbrella #monorepo #microservices

PNG PDF Copy Prompt

Project Directory

```
my_platform/
  mix.exs  Umbrella root
  mix.lock  Shared lockfile
  .formatter.exs
  .gitignore
  config/  Shared configur…
    config.exs  Imports all app…
    dev.exs
    prod.exs
    test.exs
    runtime.exs
  apps/  Child applicati…
    core/  Shared business…
      mix.exs
      lib/
        core/
          accounts/  User context
          billing/  Payment context
          repo.ex
      priv/
        repo/
          migrations/
      test/
    web/  Customer-facing…
      mix.exs
      lib/
        web/
          application.ex
          endpoint.ex
          router.ex
          live/
          controllers/
          components/
      assets/
      test/
    admin/  Admin dashboard
      mix.exs
      lib/
        admin/
          application.ex
          endpoint.ex
          router.ex
          live/
      assets/
      test/
    api/  JSON API
      mix.exs
      lib/
        api/
          application.ex
          endpoint.ex
          router.ex
          controllers/
      test/
```

Why This Structure?

Umbrellas let you split a large Phoenix system into multiple apps that share dependencies and configuration. `core/` holds business logic and database access. `web/`, `admin/`, and `api/` are separate Phoenix apps that depend on `core/`. Each app can be developed, tested, and deployed independently.

Key Directories

`apps/core/` - Shared contexts, schemas, and Repo—no web code

`apps/web/` - Customer-facing Phoenix app

`apps/admin/` - Internal admin interface

`apps/api/` - REST/GraphQL API for external consumers

`config/` - Root config imported by all child apps

Getting Started

1. `mix phx.new my_platform --umbrella`
2. `cd my_platform`
3. `cd apps && mix phx.new admin --no-ecto`
4. `mix deps.get`
5. `mix ecto.create`
6. `mix phx.server`

When To Use This

- Multiple user-facing applications (customer, admin, API)
- Clear domain boundaries requiring separation
- Teams organized around different apps
- Shared business logic across web interfaces
- Preparing for potential service extraction

App Dependencies

```
# apps/core/mix.exs
defp deps do
  [{:ecto_sql, "~> 3.10"}, {:postgrex, ">= 0.0.0"}]
end

# apps/web/mix.exs
defp deps do
  [{:phoenix, "~> 1.7"}, {:core, in_umbrella: true}]
end

# apps/web can now call Core.Accounts.get_user/1
```

Trade-offs

Complexity - More moving parts than a single app

Shared database - All apps typically share one Repo

Deployment - Usually deployed as one release, not separate services

Best Practices

- Keep `core/` free of any Phoenix/web dependencies
- Apps depend on `core/`, never on each other
- Use separate endpoints and ports for each web app
- Share authentication via `core/` contexts
- Run tests with `mix test --umbrella` from root