

Modular Monolith Project Structure

Single deployable with independent modules, clear boundaries, and internal APIs. TypeScript with NestJS.

Updated 2025-12-22

#modular-monolith #typescript #architecture #nestjs #modules #ddd

PNG

PDF

Copy

Prompt

Project Directory

myproject/

- package.json
- tsconfig.json
- nest-cli.json
- .gitignore
- .env.example
- README.md
- src/
 - main.ts App bootstrap
 - app.module.ts Root module, im...
 - modules/ Feature modules
 - users/ User management...
 - users.module.ts Module definiti...
 - api/ Public API
 - users.controller.ts
 - users.facade.ts Internal API fo...
 - dto/
 - create-user.dto.ts
 - user-response.dto.ts
 - domain/ Module domain
 - entities/
 - user.entity.ts
 - services/
 - user.service.ts
 - events/
 - user-created.event.ts
 - infrastructure/
 - repositories/
 - user.repository.ts
 - orders/ Order managemen...
 - orders.module.ts
 - api/
 - orders.controller.ts
 - orders.facade.ts
 - dto/
 - create-order.dto.ts
 - domain/
 - entities/
 - order.entity.ts
 - order-item.entity.ts
 - services/
 - order.service.ts
 - infrastructure/
 - repositories/
 - order.repository.ts
 - billing/ Billing module
 - billing.module.ts
 - api/
 - billing.controller.ts
 - billing.facade.ts
 - domain/
 - entities/
 - services/
 - invoice.service.ts
 - infrastructure/
 - shared/ Shared kernel
 - shared.module.ts
 - domain/
 - base.entity.ts
 - domain-event.ts
 - value-objects/
 - money.ts
 - email.ts
 - infrastructure/
 - database/
 - database.module.ts
 - prisma.service.ts
 - event-bus/
 - event-bus.module.ts
 - event-bus.service.ts
 - utils/
 - guards/
 - decorators/
 - filters/
 - http-exception.filter.ts
 - config/
 - app.config.ts
 - database.config.ts
 - prisma/
 - schema.prisma
 - migrations/
 - tests/
 - jest.config.ts
 - modules/
 - users/
 - orders/

Why This Structure?

A modular monolith provides microservices-like boundaries without distributed complexity. Each module owns its domain, exposes a facade for inter-module communication, and can be extracted to a service later. Single deployment, simple operations, strong boundaries.

Key Directories

modules/ - Independent feature modules with clear ownership

api/facade.ts - Internal API other modules use—never import directly from domain

shared/ - Shared kernel: base classes, value objects, infrastructure

domain/ - Each module's private domain logic

Module Communication via Facade

```
// modules/orders/api/orders.facade.ts
@Injectable()
export class OrdersFacade {
  constructor(private orderService: OrderService) {}

  async getOrdersForUser(userId: string): Promise<OrderDTO> {
    return this.orderService.findByUser(userId);
  }
}

// modules/billing/domain/services/invoice.service.ts
constructor(private ordersFacade: OrdersFacade) {}
// Use facade, never import OrderService directly
```

When To Use This

- Starting a project that may grow to microservices
- Want strong boundaries without distributed overhead
- Small team that can't operate microservices
- Domains are clear but network latency is unacceptable
- Need to ship fast with clean architecture

Module Boundaries

No cross-imports - Modules import only facades, never internal code

Own database tables - Each module owns its tables, no shared tables

Events for reactions - Use domain events for cross-module side effects

Trade-offs

Discipline required - Easy to break boundaries if team isn't careful

Shared database - Still one DB, harder to scale independently

Single deploy - Can't deploy modules independently

Testing Strategy

Unit tests - Test domain services in isolation

Module tests - Test through facade with mocked dependencies

Integration - Full module with real database

Best Practices

- Enforce boundaries with linting rules (eslint-plugin-boundaries)
- Each module has its own barrel file (index.ts) exporting only public API
- Use domain events for cross-module communication
- Shared kernel should be minimal—only truly shared concepts
- Consider module extraction criteria upfront

Naming Conventions

Modules - Plural domain names: `users`, `orders`, `billing`

Facades - `{module}.facade.ts` for internal API

Events - `{entity}-{action}.event.ts` : `user-created.event.ts`