# NestJS CQRS Project Structure

Command Query Responsibility Segregation with @nestjs/cqrs. Separate command and query handlers with domain events.

#nestjs #typescript #cqrs #event-sourcing #ddd #domain-driven

PNG PDF Copy Prompt

## Project Directory

```
myproject/
  src/ Application sou…
    main.ts
    app.module.ts Imports CqrsMod…
    config/
      configuration.ts
    common/
      interfaces/
        aggregate-root.interface.ts
  modules/ Feature modules
    orders/
      orders.module.ts Registers handl…
      orders.controller.ts
      commands/ Write operations
        impl/
          create-order.command.ts
          cancel-order.command.ts
        handlers/
          create-order.handler.ts
          cancel-order.handler.ts
      queries/ Read operations
        impl/
          get-order.query.ts
          list-orders.query.ts
        handlers/
          get-order.handler.ts
          list-orders.handler.ts
      events/ Domain events
        impl/
          order-created.event.ts
          order-cancelled.event.ts
        handlers/
          order-created.handler.ts Side effects
      domain/ Aggregate root
        order.aggregate.ts
      dto/
        create-order.dto.ts
      repositories/
        order.repository.ts
  database/
    database.module.ts
    migrations/
      .gitkeep
  test/
    app.e2e-spec.ts
    jest-e2e.json
  nest-cli.json
  tsconfig.json
  tsconfig.build.json
  package.json
  .env.example
  .eslintrc.js
  .prettierrc
  .gitignore
  README.md
```

## Why This Structure?

CQRS separates reads from writes—queries return data, commands change state. The `@nestjs/cqrs` package provides `CommandBus`, `QueryBus`, and `EventBus`. Domain events enable loose coupling between modules.

## Key Directories

**`commands/`** - Write operations—CreateOrder, UpdateOrder, etc.

**`queries/`** - Read operations—GetOrder, ListOrders, etc.

**`events/`** - Domain events emitted after state changes

**`domain/`** - Aggregate roots encapsulating business logic

**`repositories/`** - Persistence abstraction for aggregates

## Getting Started

1. `npm install`
2. `npm install @nestjs/cqrs`
3. `cp .env.example .env`
4. `npm run migration:run`
5. `npm run start:dev`

## Request Flow

Controller → CommandBus → CommandHandler → Aggregate → EventBus → EventHandler

## When To Use This

- Complex domains with rich business logic
- Read and write loads differ significantly
- Need audit trail of all state changes
- Multiple projections of the same data
- Event-driven architecture requirements

## Trade-offs

**`Boilerplate`** - Each action needs command, handler, and event classes

**`Eventual consistency`** - Read models may lag behind writes

**`Learning curve`** - DDD and event sourcing concepts take time