

Hexagonal Architecture Project Structure

Ports and Adapters architecture isolating domain logic from infrastructure. Python implementation with FastAPI.

#hexagonal #python #architecture #ddd #ports-adapters #clean-code



Project Directory

myproject/

pyproject.toml Project configu...

README.md

.gitignore

.env.example Copy to .env

> src/ Source layout

> myproject/

__init__.py

> domain/ Core business l...

__init__.py

> models/ Domain entities

__init__.py

user.py User entity

order.py Order entity

> services/ Domain services

__init__.py

user_service.py

order_service.py

> ports/ Abstract interf...

__init__.py

repositories.py Repository prot...

services.py External servic...

exceptions.py Domain exceptio...

> adapters/ Infrastructure ...

__init__.py

> inbound/ Driving adapters

__init__.py

> api/ REST API

__init__.py

main.py FastAPI app

> routes/

__init__.py

users.py

orders.py

schemas.py Pydantic DTOs

dependencies.py DI wiring

> cli/ CLI adapter

__init__.py

commands.py

> outbound/ Driven adapters

__init__.py

> persistence/ Database adapte...

__init__.py

database.py DB connection

user_repository.py

order_repository.py

orm_models.py SQLAlchemy mode...

> external/ Third-party ser...

__init__.py

email_service.py

payment_gateway.py

> config/ Configuration

__init__.py

settings.py Pydantic settin...

> tests/

__init__.py

conftest.py Shared fixtures

> unit/ Domain logic te...

__init__.py

test_user_service.py

test_order_service.py

> integration/ Adapter tests

__init__.py

test_user_repository.py

test_api.py

Why This Structure?

Hexagonal Architecture (Ports and Adapters) isolates your core domain logic from external concerns. The domain defines abstract `ports` (interfaces), and `adapters` implement them for specific technologies. This makes your business logic framework-agnostic and highly testable.

Key Directories

domain/ - Pure business logic with zero external dependencies

domain/ports/ - Abstract protocols that define how domain interacts with outside

adapters/inbound/ - Driving adapters: API, CLI, events that trigger domain

adapters/outbound/ - Driven adapters: databases, APIs, services that domain uses

Port and Adapter Example

```
# domain/ports/repositories.py
from typing import Protocol
from domain.models.user import User

class UserRepository(Protocol):
    def get_by_id(self, id: str) -> User | None: ...
    def save(self, user: User) -> None: ...

# adapters/outbound/persistence/user_repository.py
class SqlUserRepository:
    def get_by_id(self, id: str) -> User | None:
        # SQLAlchemy implementation
        ...
```

When To Use This

- Business logic needs to outlive frameworks
- Multiple entry points (API, CLI, events, scheduled jobs)
- Swapping infrastructure without touching domain
- Team has distinct domain and infrastructure expertise
- Long-term projects where technology choices may change

Dependency Rule

Dependencies point inward: adapters depend on domain, never the reverse. The domain module should have zero imports from adapters. Use dependency injection to provide concrete implementations at runtime.

Trade-offs

More boilerplate - Interfaces, DTOs, mappers add initial overhead

Overkill for simple apps - CRUD apps don't need this isolation

Learning curve - Team needs to understand ports vs adapters distinction

Testing Strategy

Unit tests - Test domain services with mock ports (in-memory implementations)

Integration tests - Test adapters against real infrastructure (DB, APIs)

E2E tests - Full stack through inbound adapters (API routes)

Best Practices

- Domain models should be plain Python classes, not ORM models
- Create mappers to convert between domain and adapter representations
- Use `typing.Protocol` for ports to avoid abstract base classes
- Keep inbound adapters thin—delegate to domain services immediately
- One port can have multiple adapters (Postgres in prod, SQLite in tests)

Naming Conventions

Ports - Named by what domain needs: `UserRepository`, `EmailSender`

Adapters - Named by technology: `SqlUserRepository`, `SmtplibEmailSender`

Services - Domain services: `UserService`, `OrderService`