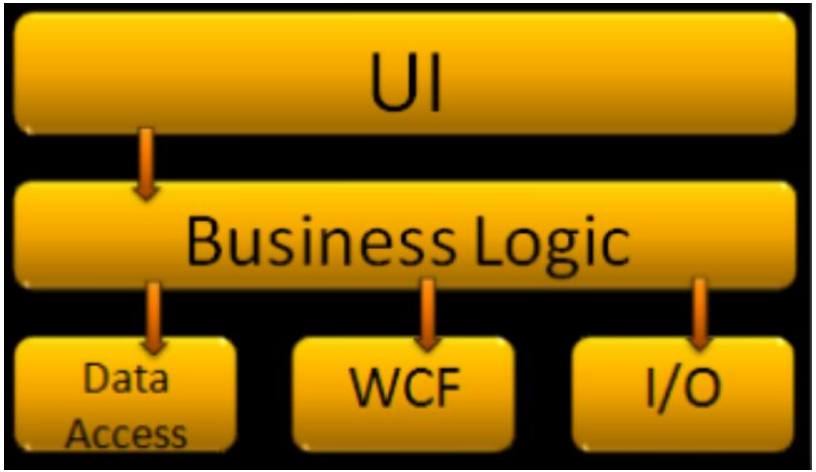
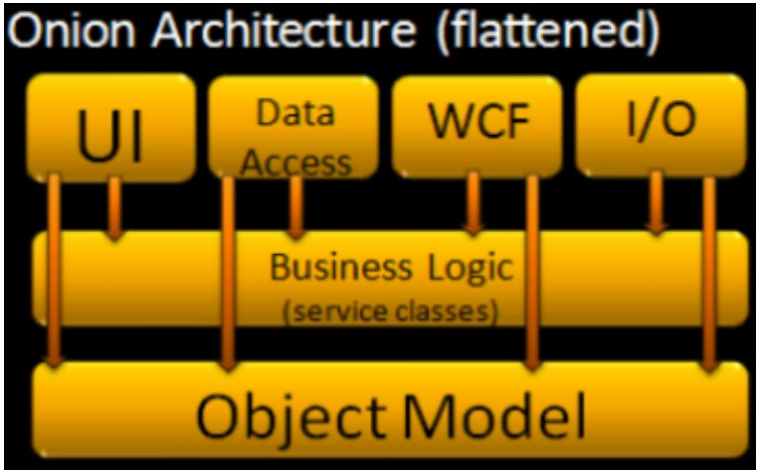
Onion Architecture

by example

Jeffrey Palermo published in 2008 three posts



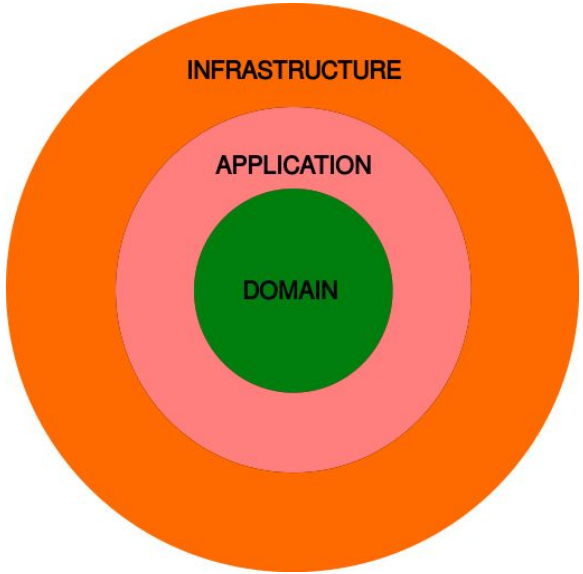


Key tenets of Onion Architecture:

- The application is built around an independent object model
- Inner layers define interfaces. Outer layers implement interfaces
- Direction of coupling is toward the center
- All application core code can be compiled and run separate from infrastructure

My key tenets of Onion Architecture:

- The application is build around domain: independent object model
- Infrastructure implements interfaces. Application calls infrastructure through interfaces.
- Direction of coupling is toward the center
- All application code can be test separate from infrastructure



Business requirements

1. Student can enroll or leave course

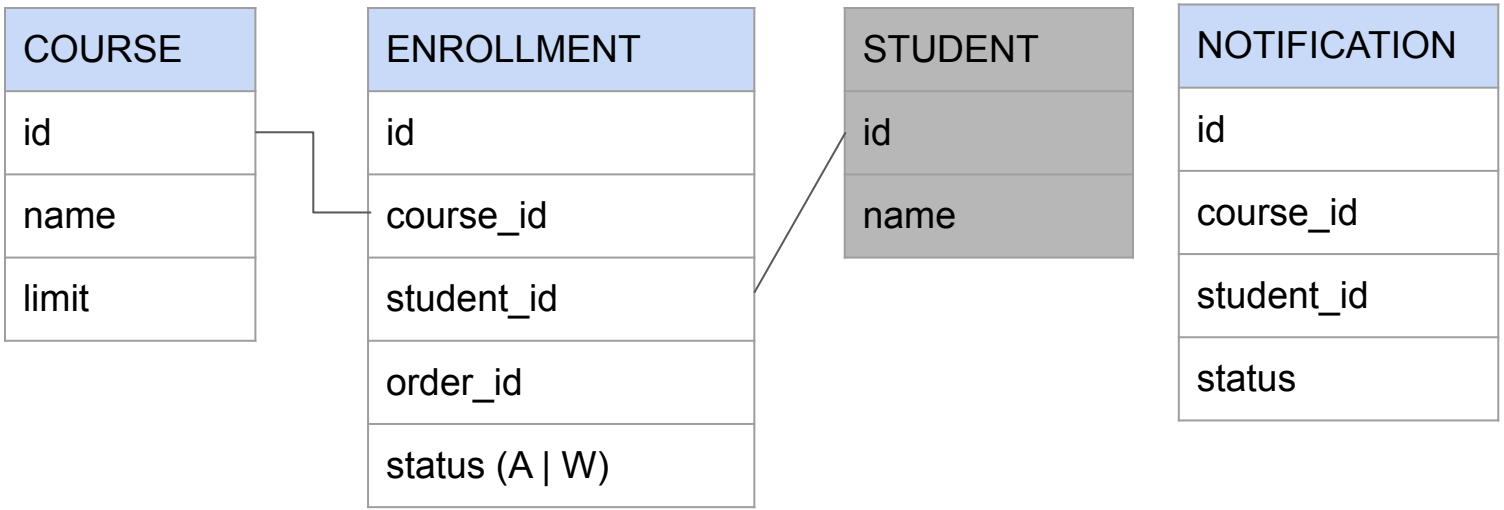
2. Course has a limit. Below limit enrollment is “active”, above is “waiting”. Enroll result must be known to student.

3. If student with active enrollment leaves course, first waiting enrollment became active. Activated student must be notified.

4. Enroll and leave must be idempotent

5. It is possible add seats to courses. They must be fairly distributed over courses. Activated students must be notified.

Database structure



Example is starting…

Concurrency is ignored

Notified mean insert to table “notification”

Please be relaxed and friendly

Black box test drawback

1. Slow, because docker compose startup
2. Slow, because application startup
3. Unstable
4. Inexplicit/blurred, because road to precondition is indirect
5. Hard for synchronous, almost impossible for asynchronous
6. Average prices over last minute - hugh?
7. But simple “Hello world” black box test could be valuable
8. But sometimes best option

Spring Test Framework and Onion Architecture

Slice (@DataJpaTest, @WebMvcTest)	infrastructure
@SpringJUnitConfig	application
@Test	domain
@SpringBootTest	blackbox
@ApplicationModuleTest (modulith)	anti-onion

When Onion is not profitable?

1. Anemic domain, business logic in application layer -> just hexagon
2. Gateway -> not hexagon
3. Criteria API (database browser) -> not hexagon
4. Websocket can be a challenge

Domain: 13 KLOC, 386 tests, 4 s

Application, 3 KLOC, 414 tests, 4 Spring contexts, 13 s

Infrastructure, 3 KLOC, 80 tests, 25 s

Coverage 87% - 74% - 40%

Modules/packages

1,2,3,4 - OK

More modules, more complexity, better segregation

If domain is separated module, good is is postpone and add when really needed

Where live DTO and mappers?

Short version:

application layer is responsible for providing data for infrastructure layer.

Long version:

DTO is Siamese twin of EJB 2.0 Persistent entity. Infrastructure (except repositories) does not operate on entities, but on request and response model . Think about models, not DTOs - DTOs died with EJB 2.0

We are what we pretend to be, so we must be careful what we pretend to be.

Kurt Vonnegut, Mother night

Onion benefits

1. Domain focused on business rules, not DB query, HTTP get, cache etc.
2. Application shows execution flow
3. Easy navigation from infrastructure
4. No mapping 1-1
5. Logic is tested fast and precisely

Key takeaways

Onion architecture is extension of hexagon

Most important is domain object model without external references (fact model).

All application code can be tested separated from infrastructure

Links

Jeffrey Pallermo Onion

Victor Rentea Onion Overengineering

Oskar Dudycz Odchudź swoje agregaty

Allegro Tech Blog Onion

Sara Pellegrini Kill aggregate

github michaldo onion