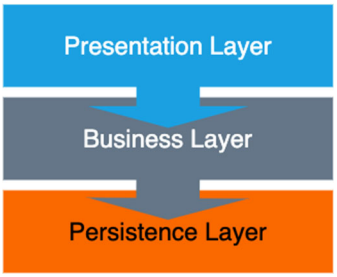
ToiletPaper #148

About hexagons and onions

Author: Marcel Jacob / Software Engineer / Office Leipzig

✕ Problem



A classic layered architecture usually has three layers depending on each other (see image on the right). These are considered top-down and each layer only knows the layer directly below it. However, the dependency of the business layer and the persistence layer must be viewed critically:

Why is the business logic dependent on something like a database? Shouldn't it be possible to execute it independently of a framework or database?

✓ Solution

One possible solution is **hexagonal architecture** or **ports and adapters architecture**. The central terms port and adapter are explained in the figure below.
In our example, the architecture consists of four layers: **presentation, infrastructure, application, and domain**, with the presentation layer omitted for a pure back-end system. However, these are not considered top-down, but from the outside in, with the domain being the core. Outer layers may access code from inner layers, but not vice versa. This principle corresponds to the basic idea of the **onion architecture**, which can be extended with interfaces and their implementations (ports and adapters) to enable access to other layers. The presentation layer (UI) is usually a separate project and will not be explained further in this paper. An overview of the other three layers follows:

Infrastructure Layer

- Can contain framework-specific code
- Contains configurations such as Spring or database configuration
- Connects the output ports of the domain layer via adapters e.g. for the database or messaging

Port - defines an interface

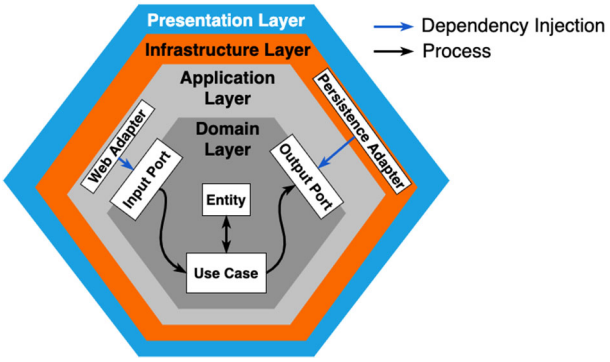
- One **Input Port** from outside to the domain layer, e.g. a specific use case
- One **Output Port** from the domain layer to another system

Adapter - defines the port´s implementation

- is a component of the application and infrastructure layer that connects to the domain layer

Application Layer

- Provides interfaces or other applications for the user, e.g. REST, GraphQL, CLI etc.
- Connects to the domain layer via adapter of an input port



Domain Layer

- Contains the business logic
- Is independent of a framework such as Spring
- Allows communication between layers only via defined input and output ports
- Code in this layer is unlikely to change

This structure makes individual classes easier to test and also to maintain and the business logic is independent and can be moved as a whole if necessary.

If you decide to use hexagonal architecture or a hybrid form with the onion architecture (as in the figure above), you must be aware of the higher complexity during implementation, which is why this architecture style is only worthwhile from a certain project size.

+ Further Aspects

- Hexagonal architecture: https://medium.com/ssense-tech/hexagonal-architecture-there-are-always-two-sides-to-every-story-bc0780ed7d9c
- Hexagonal architecture with Java and Spring: https://reflectoring.io/spring-hexagonal/