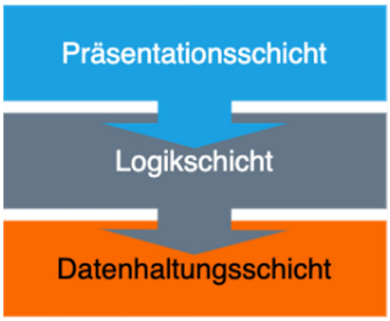
ToiletPaper #148

Von Sechsecken und Zwiebeln

Autor: Marcel Jacob / Software Engineer / Standort Leipzig

✕ Problem



In einer klassischen Schichtenarchitektur hat man in der Regel drei Layer, die von voneinander abhängen (siehe Bild rechts). Diese werden meist Top-Down betrachtet und jede Schicht kennt nur die direkt darunterliegende. Die Abhängigkeit der Logik-Schicht vom Daten-Layer ist aber kritisch zu betrachten:

Warum ist die Fachlogik abhängig von etwas wie einer Datenbank? Sollte diese nicht unabhängig von einem Framework oder einer Datenbank ausgeführt werden können?

✓ Lösung

Eine mögliche Lösung stellt die Hexagonale Architektur oder auch Ports-and-Adapters-Architektur dar. Die im Mittelpunkt stehenden Begriffe Port und Adapter sind im Bild unten erklärt.
In unserem Beispiel besteht die Architektur aus vier Schichten: Presentation, Application, Domain und Infrastructure, wobei die Präsentationsschicht bei einem reinen Backend-System entfällt. Diese werden aber nicht Top-Down betrachtet, sondern von außen nach innen, wobei die Domain den Kern darstellt. Äußere Schichten dürfen auf Code innerer Schichten zugreifen, aber nicht umgekehrt. Dieses Prinzip entspricht der Grundidee der Onion-Architektur, die mit Schnittstellen und deren Implementierungen (Ports und Adapters) erweitert werden können, um den Zugriff auf anderen Layer zu ermöglichen. Der Presentation-Layer (UI) ist meist ein separates Projekt und wird hier nicht weiter betrachtet. Es folgt ein Überblick über die anderen drei Layer:

Der **Infrastructure**-Layer

- Kann frameworkspezifischen Code enthalten
- Enthält Konfigurationen wie z.B. die Spring- oder die Datenbankkonfiguration
- Verbindet die Output Ports des Domain Layers via Adapter z. B. für die Datenbank oder Messaging

Der **Application**-Layer

- Stellt Interfaces für die User oder andere Applikationen bereit, z. B. REST, GraphQL, CLI etc.
- Verbindet sich via Adapter eines Input Ports zum Domain-Layer

Der **Domain**-Layer

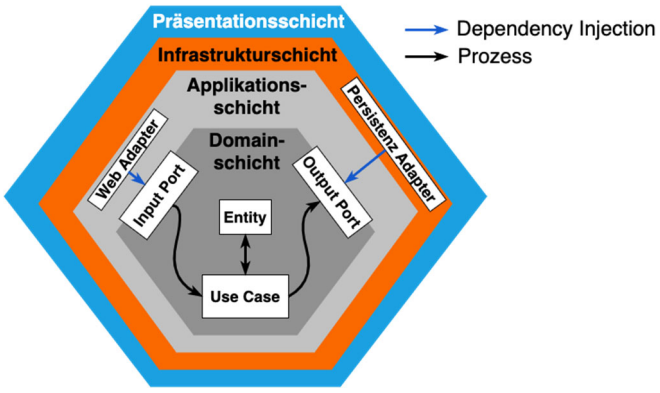
- Enthält die Businesslogik
- Ist unabhängig von einem Framework wie z. B. Spring
- Erlaubt die Kommunikation zwischen den Layern nur über definierte Input und Output Ports
- Code in diesem Layer ändert sich in der Regel selten

Port - definiert ein Interface

- ein **Input Port** von außerhalb zum Domain-Layer, z.B. einen bestimmten Use Case
- ein **Output Port** vom Domain-Layer zu einem anderen System

Adapter - definiert die Implementierung des Ports

- ist eine Komponente des Application- oder Infrastructure-Layers, die sich zum Domain-Layer verbindet



Durch diesen Aufbau sind die einzelnen Klassen besser test- und auch wartbar, die Fachlogik ist unabhängig und kann bei Bedarf als Ganzes verschoben werden.

Entscheidet man sich für die Hexagonale Architektur oder eine Mischform mit der Onion-Architektur (wie in der Abbildung oben), muss man sich der höheren Komplexität bei der Implementierung bewusst sein, weshalb sich dieser Architekturstil erst ab einer gewissen Projektgröße lohnt.

+ Weiterführende Aspekte

- Coffee Talk: Einführung in Hexagonale Architektur - Theorie und Praxis im Projekt
- Hexagonal: https://www.embedded-software-engineering.de/ports-and-adapters-eine-software-architektur-fuer-moderne-applikationen-a-663985/
- Onion-Architektur und DDD: https://www.innoq.com/de/blog/ddd-mit-onion-architecture-umsetzen/