

## Program szkolenia:

# Architektura aplikacji i systemów - Wzorce architektoniczne dla projektantów

## Informacje:

<b>Nazwa:</b>	<b>Architektura aplikacji i systemów - Wzorce architektoniczne dla projektantów</b>
<b>Kod:</b>	<b>Arch-patterns</b>
<b>Kategoria:</b>	Wzorce architektoniczne
<b>Odbiorcy:</b>	architekci
<b>Czas trwania:</b>	3 dni
<b>Forma:</b>	50% wykłady / 50% warsztaty

Szkolenie prezentuje wybrane Wzorce Projektowe i Architektoniczne w praktycznym i niepodręcznikowym ujęciu osadzonym w kontekście projektowania aplikacji Webowych, platform, systemów i frameworków. Podczas szkolenia prezentowane są przykłady praktycznego zastosowania zaczerpnięte z rzeczywistych systemów klas: ERP, narzędzia wizualne, systemy rozproszone, serwery.

Podczas szkolenia uczestnicy nabędą zintegrowaną wiedzę na temat zdobyci nowoczesnej inżynierii oprogramowania pozwalającą im na tworzenie zaawansowanych systemów. Omawiane zagadnienia leżą u podstaw nowoczesnych frameworków i technologii – co zwiększa poziom ich zrozumienia i pozwala na świadome korzystanie. Przedstawiamy techniki łączenie wzorców w struktury wyższego rzędu.

Szkolenie przeznaczone dla projektantów i architektów pragnących poszerzyć swe kompetencje w zakresie standardowych stylów architektonicznych.

**Program jest ogólną ramą merytoryczną. Jego realizacja wykracza poza 3 dni. W trakcie analizy przedszkoleniowej wybieramy wzorce, które będą przydatne dla zespołu i skupiamy się tylko na nich w trakcie szkolenia. W ciągu 3 dni zwykle jesteśmy w stanie zaadresować ok 70% wymienionych wzorców.**

## Zalety szkolenia:

- Nowoczesne architektury (CqRS - wspierająca DDD)
- Możliwość oparcia przykładów o Kafka
- Dobór klasy rozwiązania do klasy problemu

## Szczegółowy program:

### 1. Dokumentowanie architektury w podejściu 4C - Context, Containers, Components, Classes

#### 1.1. Context

1.1.1. Kontekst w jakim będzie działał system

1.1.2. Główni aktorzy

1.1.3. Procesy na poziomie organizacji w które zaangażowany jest system

1.1.4. Rola innych systemów

#### 1.2. Containers

1.2.1. Kontenery techniczne w jakich będzie rozmieszczony system

1.2.2. Wzorce integracji

1.2.3. Protokoły komunikacji

#### 1.3. Components

1.3.1. Komponenty i Serwisy w ujęciu SOA

1.3.2. Dane kanoniczne wymieniane pomiędzy Serwisami

1.3.3. Projektowanie API

1.3.4. Zasada jednego źródła prawdy

#### 1.4. Classes

1.4.1. Wstęp do technik i wzorców projektowania na poziomie klas

### 2. Architektura systemu - przegląd podejść

#### 2.1. Modularyzacja

2.1.1. Poziomy separacji

2.1.2. Techniki integracji modułów

#### 2.2. Architektura Micro Kernel

#### 2.3. Command + Command Handler

2.4. Ports and Adapters

2.5. Restful

2.5.1. 4 poziomy dojrzałości

2.5.2. Rest jako protokół aplikacyjny a nie transportowy

2.6. Event Driven Architecture

2.6.1. Architektury oparte o zdarzenia

2.6.1.1. Events broker

2.6.1.2. Events Bus

2.6.1.3. Eventual Consistency

2.6.2. Zależności czasowe pomiędzy zdarzeniami

2.6.3. Sagi – orkiestracja zdarzeń

### 3. Architektury aplikacji

3.1. Podejście warstwowe

3.1.1. Różnice pomiędzy Layer a Tier

3.1.2. Podział na logikę aplikacji i logikę domenową

3.1.2.1. Logika aplikacji

3.1.2.2. Modelowanie Use Case/User Story

3.1.2.3. Stanowo czy bezstanowo

3.1.2.4. Problemy projektowania API modułu

3.1.2.5. Logika domenowa i Techniki Domain Driven Design - Building Blocks

3.1.2.6. Agregaty

3.1.2.7. Encje

3.1.2.8. Value Objects

3.1.2.9. Serwisy Domenowe

3.1.2.10. Polityki

3.1.2.11. Specyfikacje

3.1.2.12. Fabryki

3.1.2.13. Repozytoria

3.1.2.14. Modelowanie niezmienników

3.1.2.15. Poziomy modelu: Capacity, Operations, Policy – dostrajanie modelu, Decision Support

3.1.3. Strategiczne testowanie warstw – ogólny przegląd podejścia

3.1.3.1. Problem eksplozji kombinatorycznej przypadków testowych

3.1.3.2. Testowanie górnych warstw w podejściu end2end

3.1.3.3. Testowanie dolnych warstw w podejściu jednostkowym

3.2. Praktyczne wykorzystanie technik Inversion of Control (DI, Events, AOP)

3.2.1. Dependency Injection – podstawowa technika IoC

3.2.1.1. Wykorzystanie zamiast wzorców fabrykujących

3.2.1.2. Budowanie konkretnych Strategii, Dekoratorów itd. w zależności od stanu aplikacji (kontekst, konfiguracja)

3.2.1.3. Otwartość na rozbudowę dzięki wzorcom Strategii

3.2.2. Systemy sterowane zdarzeniami – silniejsza technika IoC

3.2.2.1. Użycie do tworzenia rozszerzalnych architektur opartych o pluginy

3.2.2.2. Użycie do tworzenia skalowalnych systemów wysokiej wydajności (wykorzystanie kolejek, np. JMS)

3.2.2.3. Sagi - Modelowanie złożonych procesów zdarzeniowych

3.2.3. Aspect Oriented Programming

3.2.3.1. Wstęp do AOP

3.2.3.2. Techniki Interceptorów

3.2.3.3. Przykłady zastosowania AOP

3.3. Projektowanie systemów otwartych na rozbudowę

3.3.1. pluginy

3.3.2. listenery

3.3.3. wstrzykiwanie rozszerzeń

3.3.4. podejście aspektowe

3.4. Wzorce warstwy dostępu do danych

3.4.1. DAO

3.4.2. Repository

3.4.3. Active Record

3.4.4. Object-relational Mapping - na przykładzie Java Persistence API (Hibernate)

3.4.5. Problemy ORM

3.4.5.1. Niezgodność paradygmatów

3.4.5.2. Mapowanie granic obiektów

3.4.5.3. Kompozycja czy agregacja - mapowanie

3.4.5.4. 3 podejścia do blokowania optymistycznego

3.4.5.5. Wersjonowanie

3.4.5.6. Blokowanie obiektów na podstawie których tworzymy inne obiekty (READ)

3.4.5.7. Blokowanie spójności grafów obiektów poprzez ich korzeń (WRITE)

## 4. Modularyzacja

4.1. Sepracja Bounded Context

4.1.1. Techniki wyznaczania granic modułów

4.1.2. Techniki hermetyzacji modułów

4.1.2.1. API

4.1.2.2. DTO

4.2. Integracja modułów - podejścia

4.2.1. Zdarzenia

4.2.2. Wywołania API

4.2.3. Silniki procesów biznesowych

## 5. Command-query Responsibility Segregation – rozszerzona architektura warstwowa

5.1. Wsparcie dla Domain Driven Design

5.2. Rozwiązanie problemów z niedopasowaniem ORM do przeglądu danych w Gridach

5.3. Zorientowanie na skalowanie i rozszerzalność

5.4. Tworzenie dedykowanych modeli: do odczytu, do operacji biznesowych

5.4.1. Rozwiązanie problemów z wydajnością

5.4.2. Architektura zewnętrznego indeksu z użyciem noSQL

## 6. noSQL

6.1. Zastosowania

6.2. Rodzaje baz - dobór do problemu

## 7. Testability – projektowanie architektur aplikacji zorientowanych na testy

7.1. Strategiczne testowanie

7.1.1. Problem eksplozji kombinatorycznej przypadków testowych

7.1.2. Mapowanie warstw aplikacji na piramidę testów

7.1.3. Strategia

7.1.3.1. Warstwa domenowa - testowanie jednostkowe

7.1.3.2. Warstwa serwisów - testowanie end2end

7.2. Dążenie do uruchamiania logiki poza serwerem – zwiększanie produktywności, redukcja czasu używanego na redeploy

7.3. Zagadnienia podatności architektury na testy: problemy i pułapki

7.4. Techniki testowania jednostkowego: dummy, fake, stub, mock

7.5. Narzędzia testowania jednostkowego i integracyjnego (JUnit, Mockito)

7.6. 3 poziomy testów

7.6.1. Specification by Example - cele biznesowe

7.6.2. Flow - User Story

7.6.3. Automatyzacja interakcji z UI - Agenty będące abstrakcją nad skrypcem testowym