

## Program szkolenia:

# Wzorce projektowe i architektura dla platformy Java EE

## Informacje:

<b>Nazwa:</b>	<b>Wzorce projektowe i architektura dla platformy Java EE</b>
<b>Kod:</b>	<b>java-arch-arch</b>
<b>Kategoria:</b>	Architektura Java
<b>Grupa docelowa:</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 systemów na platformie Java EE. 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. Przedstawiamy techniki łączenie wzorców w struktury wyższego rzędu.

## Zalety szkolenia:

- Nowoczesne architektury (CqRS - wspierająca DDD)
- Szersza perspektywa
- Dobór klasy rozwiązania do klasy problemu

## Szczegółowy program:

### 1. Architektura systemu

#### 1.1. Modularyzacja

##### 1.1.1. Poziomy separacji

##### 1.1.2. Techniki integracji modułów

###### 1.1.2.1. Zdarzenia

###### 1.1.2.2. Wywołania API

###### 1.1.2.3. Silniki procesów biznesowych

#### 1.2. SOA

##### 1.2.1. Duplikacja danych, ale nie duplikacja logiki

##### 1.2.2. Zasada jednego Źródła Prawdy

#### 1.3. Event Driven z wykorzystaniem JMS

##### 1.3.1. Architektury oparte o zdarzenia

###### 1.3.1.1. Events broker

###### 1.3.1.2. Events Bus

###### 1.3.1.3. Eventual Consistency

##### 1.3.2. Zależności czasowe pomiędzy zdarzeniami

##### 1.3.3. Sagi – orkiestracja zdarzeń

##### 1.3.4. Wykorzystanie JMS jako silnika zdarzeń biznesowych

#### 1.4. Alternatywne podejścia do architektury systemu

##### 1.4.1. Command + Command Handler

##### 1.4.2. Ports and Adapters

###### 1.4.2.1. Integracja

###### 1.4.2.2. SOA ready

## 1.4.2.3. Strategia multiscreen

**2. Architektury aplikacji**

## 2.1. Podejście warstwowe

## 2.1.1. Różnice pomiędzy Layer a Tier

## 2.1.2. Ogólny podział - 3I

## 2.1.2.1. Interfejsy

## 2.1.2.2. Interakcje

## 2.1.2.3. Integracja

**3. Wzorce warstwy prezentacji z wykorzystaniem JSF**

## 3.1. MVC, MVP, MVVM - poszukiwanie odpowiedniej metafory dla Backing Bean

## 3.2. Front Controller i Page Controller

## 3.3. DTO - kiedy zastosowanie ma sens

## 3.4. Ekran komponentowy - dobór odpowiedniego zasięgu dla fragmentów modelu

**4. Wzorce warstwy logiki**

## 4.1. Podział na logikę aplikacji i logikę domenową

## 4.2. Logika aplikacji

## 4.2.1. Modelowanie Use Case/User Story

## 4.2.2. Stanowo czy bezstanowo

## 4.2.3. Wykorzystanie Managed Beans i dobór odpowiedniego zasięgu

## 4.2.4. Zarządzanie transakcjami - poziomy izolacji i propagacja transakcji

## 4.2.5. Transakcje aplikacyjne - wykorzystanie rozszerzonego trybu kontekstu persystencji

## 4.3. Logika domenowa i Techniki Domain Driven Design - Building Blocks

## 4.3.1. Agregaty - mapowanie relacyjno obiektowe zgodne z zasadami Object Oriented

## 4.3.2. Encje

4.3.3. Value Objects

4.3.4. Serwisy Domenowe

4.3.5. Polityki

4.3.6. Specyfikacje

4.3.7. Fabryki

4.3.8. Repozytoria - poprawne wykorzystanie JPA

4.3.9. Modelowanie niezmienników

4.3.10. Poziomomy modelu: Capacity, Operations, Policy – dostrajanie modelu, Decision Support

## 5. Wzorce warstwy dostępu do danych - wykorzystanie JPA

5.1. Shared Repository

5.2. DAO

5.3. Repository

5.4. Wzorce blokowania całych grafów obiektów (ochrona spójności zgodna z OO)

5.4.1. Optymistyczne - dobór podejścia do scenariusza

5.4.1.1. Wersjonowanie encji

5.4.1.2. Read (optimistic)

5.4.1.3. Write (optimistic\_force\_increment) - ochrona agregatów (grafów encji)

5.4.1.4. Pesymistyczne

5.4.1.5. Pułapki i najlepsze praktyki

5.5. Modelowanie encji

5.5.1. Wykorzystanie typów zagnieżdżonych do mapowania Value Objects

5.5.2. Modelowanie niezmienników

5.5.3. Dobór struktur danych w powiązaniu One-to-Many

5.5.3.1. Set

5.5.3.2. Bag

5.5.3.3. List

5.5.3.4. Unikanie zbędnych operacji odtwarzania i pobierania kolekcji

5.6. Mapowanie kolekcji

5.6.1. Tabele linkujące w powiązaniu One-to-Many

5.6.1.1. Kiedy stosować - kompozycja a agregacja

5.6.2. Strategie pobieranie w kontekście modelu biznesowego

5.7. Optymalizacja operacji

5.7.1. Pobieranie Proxy

5.7.2. Dynamiczny Update

5.8. Operacje kaskadowe a granica Agregatu

5.9. Problem identyfikacji encji - dobór podejścia

5.9.1. klucze główne

5.9.2. klucze naturalne

5.9.3. UUID

5.9.4. referencja w pamięci

5.10. Dziedziczenie

5.10.1. Kiedy dziedziczenie nie ma sensu w modelu biznesowym

5.10.2. Implementacje dziedziczenia - wady i zalety każdego z podejść

5.10.3. Pojedyncza tabela

5.10.4. Tabela per klasa

5.10.5. Tabele dla super klas

5.10.6. Alternatywa dla dziedziczenia

5.10.6.1. Party Archetype

5.10.6.2. Role Object Pattern

## 5.11. Wydajność dostępu do danych

### 5.11.1. Pułapki wydajności JPA

5.11.1.1. „n+1 Select problem” - wykrywanie i zapobieganie

5.11.1.2. Wykrywanie N+1SP

5.11.1.3. Wykrywanie automatyczne przy pomocy AOP

5.11.1.4. Sposoby naprawy N+1SP

5.11.1.5. JOIN FETCH

5.11.1.6. Batch size

### 5.11.2. Pułapki Lazy Loadingu oraz zbyt chciwego pobierania danych

5.11.2.1. Nadmierne pobieranie danych

5.11.2.2. Racjonalne wykorzystanie Lazy Loadingu

## 6. Strategiczne testowanie warstw – ogólny przegląd podejścia

6.1. Problem eksplozji kombinatorycznej przypadków testowych

6.2. Testowanie górnych warstw w podejściu end2end

6.3. Testowanie dolnych warstw w podejściu jednostkowym

## 7. Praktyczne wykorzystanie technik Inversion of Control do budowy frameworków i systemów

7.1. Dependency Injection – podstawowa technika IoC

7.1.1. Wykorzystanie zamiast wzorców fabrykujących

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

7.1.3. Otwartość na rozbudowę dzięki wzorcom Strategii

7.2. Systemy sterowane zdarzeniami – silniejsza technika IoC

7.2.1. Użycie do tworzenia rozszerzalnych architektur opartych o pluginy

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

7.2.3. Sagi - Modelowanie złożonych procesów zdarzeniowych

### 7.3. Aspect Oriented Programming

#### 7.3.1. Wykorzystanie interceptorów

#### 7.3.2. Wstęp do AOP

#### 7.3.3. Techniki Interceptorów

#### 7.3.4. Przykłady zastosowania AOP

### 7.4. Projektowanie systemów otwartych na rozbudowę

#### 7.4.1. Pluginy

#### 7.4.2. Listenery

#### 7.4.3. Wstrzykiwanie rozszerzeń

#### 7.4.4. Podejście aspektowe

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

### 8.1. Wsparcie dla Domain Driven Design

### 8.2. Rozwiązanie problemów z niedopasowaniem ORM do przeglądu danych w Gridach

### 8.3. Zorientowanie na skalowanie i rozszerzalność

### 8.4. Tworzenie dedykowanych modeli: do odczytu, do operacji biznesowych

#### 8.4.1. Rozwiązanie problemów z wydajnością

#### 8.4.2. Architektura zewnętrznego indeksu z użyciem noSQL

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

### 9.1. Strategiczne testowanie

#### 9.1.1. Problem eksplozji kombinatorycznej przypadków testowych

#### 9.1.2. Mapowanie warstw aplikacji na piramidę testów

#### 9.1.3. Strategia

##### 9.1.3.1. Warstwa domenowa - testowanie jednostkowe

##### 9.1.3.2. Warstwa serwisów - testowanie end2end

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

9.3. Zagadnienia podatności architektury na testy: problemy i pułapki

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

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

9.6. 3 poziomy testów

9.6.1. Specification by Example - cele biznesowe

9.6.2. Flow - User Story

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