

Training program:

Application and Systems Architecture – Architectural Patterns for application architects

Info:

Name:	Application and Systems Architecture – Architectural Patterns for application architects
Code:	Arch-patterns
Category:	Architectural patterns
Target audience:	architects
Duration:	3 days
Format:	50% lecture / 50% workshop

This training presents selected Design and Architectural Patterns in a practical and non-textbook approach, set in the context of designing web applications, platforms, systems, and frameworks. It includes practical examples drawn from real-world systems such as ERP, visual tools, distributed systems, and servers.

Participants will gain integrated knowledge of modern software engineering achievements, enabling them to create advanced systems. The discussed topics underpin modern frameworks and technologies, enhancing understanding and facilitating conscious usage. Techniques for combining patterns into higher-order structures are also presented.

The training is intended for designers and architects seeking to expand their competencies in standard architectural styles.

The program serves as a general framework; its full implementation exceeds 3 days. During the pre-training analysis, we select patterns beneficial for the team and focus on them during the training. Typically, we can address about 70% of the listed patterns within 3 days.

It's all about the content.

- Modern architectures (CQRS – supporting DDD)
- Possibility of basing examples on Kafka
- Matching solution class to problem class

Training program

1. Documenting Architecture in the 4C Approach - Context, Containers, Components, Classes

1.1. Context

1.1.1. The context in which the system will operate

1.1.2. Main actors

1.1.3. Organization-level processes involving the system

1.1.4. Role of other systems

1.2. Containers

1.2.1. Technical containers in which the system will be deployed

1.2.2. Integration patterns

1.2.3. Communication protocols

1.3. Components

1.3.1. Components and Services in the SOA context

1.3.2. Canonical data exchanged between Services

1.3.3. API design

1.3.4. Single source of truth principle

1.4. Classes

1.4.1. Introduction to design techniques and patterns at the class level

2. System Architecture - Overview of Approaches

2.1. Modularization

2.1.1. Levels of separation

2.1.2. Techniques for integrating modules

2.2. Micro Kernel Architecture

2.3. Command + Command Handler

2.4. Ports and Adapters

2.5. Restful

2.5.1. Four maturity levels

2.5.2. Rest as an application protocol, not a transport protocol

2.6. Event Driven Architecture

2.6.1. Architectures based on events

2.6.1.1. Events broker

2.6.1.2. Events Bus

2.6.1.3. Eventual Consistency

2.6.2. Temporal dependencies between events

2.6.3. Sagas – orchestration of events

3. Application Architectures

3.1. Layered Approach

3.1.1. Differences between Layer and Tier

3.1.2. Separation of application logic and domain logic

3.1.2.1. Application Logic

3.1.2.2. Modeling Use Case/User Story

3.1.2.3. Stateful or stateless

3.1.2.4. Challenges in module API design

3.1.2.5. Domain Logic and Domain Driven Design Techniques - Building Blocks

3.1.2.6. Aggregates

3.1.2.7. Entities

3.1.2.8. Value Objects

3.1.2.9. Domain Services

3.1.2.10. Policies

3.1.2.11. Specifications

3.1.2.12. Factories

3.1.2.13. Repositories

3.1.2.14. Modeling invariants

3.1.2.15. Model levels: Capacity, Operations, Policy – model tuning, Decision Support

3.1.3. Strategic Layer Testing – general approach overview

3.1.3.1. Combinatorial explosion of test cases problem

3.1.3.2. Testing upper layers using end2end approach

3.1.3.3. Testing lower layers using unit approach

3.2. Practical use of Inversion of Control techniques (DI, Events, AOP)

3.2.1. Dependency Injection – basic IoC technique

3.2.1.1. Use instead of factory patterns

3.2.1.2. Building specific Strategies, Decorators, etc. depending on application state (context, configuration)

3.2.1.3. Openness to extension via Strategy patterns

3.2.2. Event-driven systems – stronger IoC technique

3.2.2.1. Use for creating extensible plugin-based architectures

3.2.2.2. Use for creating scalable high-performance systems (using queues, e.g. JMS)

3.2.2.3. Sagas - modeling complex event-driven processes

3.2.3. Aspect Oriented Programming

3.2.3.1. Introduction to AOP

3.2.3.2. Interceptor techniques

3.2.3.3. Examples of AOP usage

3.3. Designing extensible systems

3.3.1. plugins

3.3.2. listeners

3.3.3. injecting extensions

3.3.4. aspect-oriented approach

4. Data access layer patterns

4.1. DAO

4.2. Repository

4.3. Active Record

4.4. Object-relational Mapping - using Java Persistence API (Hibernate)

4.5. ORM Issues

4.5.1. Paradigm mismatch

4.5.2. Object boundary mapping

4.5.3. Composition or aggregation - mapping

4.5.4. 3 approaches to optimistic locking

4.5.5. Versioning

4.5.6. Locking objects that serve as bases for others (READ)

4.5.7. Ensuring consistency of object graphs via root locking (WRITE)

5. Modularization

5.1. Separation of Bounded Contexts

5.1.1. Techniques for defining module boundaries

5.1.2. Techniques for module encapsulation

5.1.2.1. API

5.1.2.2. DTO

5.2. Module integration - approaches

5.2.1. Events

5.2.2. API calls

5.2.3. Business process engines

6. Command-query Responsibility Segregation – extended layered architecture

6.1. Support for Domain Driven Design

6.2. Solving ORM mismatch problems for data viewing in grids

6.3. Oriented toward scalability and extensibility

6.4. Creating dedicated models: for reading, for business operations

6.4.1. Solving performance issues

6.4.2. External index architecture using noSQL

7. noSQL

7.1. Use cases

7.2. Types of databases - matching to the problem

8. Testability – designing test-oriented application architectures

8.1. Strategic testing

8.1.1. Combinatorial explosion of test cases

8.1.2. Mapping application layers to the testing pyramid

8.1.3. Strategy

8.1.3.1. Domain layer - unit testing

8.1.3.2. Service layer - end2end testing

8.2. Striving to run logic outside the server – increasing productivity, reducing redeployment time

8.3. Architecture's susceptibility to testing: problems and pitfalls

8.4. Unit testing techniques: dummy, fake, stub, mock

8.5. Unit and integration testing tools (ex: JUnit, Mockito)

8.6. 3 levels of tests

8.6.1. Specification by Example - business goals

8.6.2. Flow - User Story

8.6.3. Automating UI interaction - Agents as abstraction over test scripts