

# Architektura JavaScript

*Program szkolenia, Ducin IT Consulting*

Czas trwania: 3-5 dni

Formuła: 33% wykłady, 33% ćwiczenia, 33% projektowanie

Szkolenie ma na celu ugruntowanie oraz poszerzenie wiedzy z zakresu designu oraz architektury oprogramowania - ze szczególnym naciskiem na platformę JavaScript. Przegląd narzędzi oraz technik odbywa się poprzez omawianie zarówno powodów, dla których dane rozwiązanie należy zastosować, jak i kluczowych różnic, jakie między nimi występują. To z kolei kładzie nacisk na rozumienie konsekwencji dokonywanych przez architektów decyzji. Przykłady oraz implementacja koncentrują się na języku JavaScript, jednak omawiane zagadnienia (wzorce projektowe, architektoniczne, pryncypia) są uniwersalne; niektóre dotyczą także backendu.

Uczestnicy szkolenia poznają szerszy kontekst wokół narzędzi, których na co dzień używają. Dzięki ćwiczeniom z projektowania w grupach zwiększają rozumienie problemów, które należy rozwiązać. Szkolenie nie uczy kodowania w konkretnym frameworku/ narzędziu. Przygotowuje do roli architekta - osoby odpowiedzialnej za zrozumienie klas problemów i wybór klas rozwiązań a także rozumienia konsekwencji podejmowania decyzji (ich zakresu oraz czasu).

## Zakres

Poniższy program jest ramowy, tj. uwzględnia szeroki zakres, niekoniecznie możliwych do realizacji w ciągu 3 dni. Uczestnicy szkolenia w dużej mierze na bieżąco mają wpływ na wybór zagadnień, na których realizacji najbardziej im zależy. W praktyce, docelowy program formułuje się sam w trakcie szkolenia.

## Zalety Szkolenia:

- Przegląd klasycznych problemów z jakimi mierzą się aplikacje frontendowe wraz z możliwymi rozwiązaniami
- Style architektoniczne, ich zastosowanie, zalety i wady
- Wzorce projektowe

# Program szkolenia:

## 1. Architektura - od strony miękkiej

- 1.1. Cele architektury
- 1.2. Architekt - rola w projekcie, obowiązki i wyzwania
- 1.3. Projektowanie architektury
- 1.4. Poszukiwanie cech rozwiązań - oraz samych rozwiązań
- 1.5. Zarządzanie decyzjami architektonicznymi
- 1.6. Identyfikowanie potencjalnych problemów
- 1.7. Planowanie i realizacja refaktoringów

## 2. Concepts

- 2.1. Coupling
- 2.2. Cohesion
- 2.3. Reusability
- 2.4. Inversion of Control
- 2.5. Caching

## 3. Principles

- 3.1. SOLID
- 3.2. DRY
- 3.3. KISS
- 3.4. YAGNI
- 3.5. Law of Demeter
- 3.6. The Hollywood Principle

## 4. Design Patterns

- 4.1. Singleton

4.2. Observer

4.3. Pub-Sub

4.4. Factory

4.5. Command

4.6. Strategy

4.7. Memento

4.8. State Machine

## 5. Programming Models

5.1. Sync vs Async vs Parallel vs Distributed

5.2. Parallel vs Concurrent

5.3. Fallacies of Distributed Computing

## 6. Backend Architectural Styles

6.1. Monoliths vs Microservices

6.2. CRUD

6.3. CQRS

6.4. Event Sourcing

6.5. Ports and Adapters (Hexagonal)

6.6. Scalability

## 7. Frontend Architectural Concerns

7.1. Monoliths vs Microservices

7.2. Dataflow

7.2.1. One-time binding

7.2.2. One-way data flow

7.2.3. Two-way data binding

7.3. State Management

7.4. Runtime, Patterns and Design

7.4.1. jQuery

7.4.2. AngularJS

7.4.3. React

7.4.4. Angular

7.4.5. Svelte

7.4.6. WebComponents

7.5. Microfrontends

## 8. Frontend Application Features

8.1. Notifications

8.2. Realtime Operations

8.3. Optimistic Updates

8.4. Caching

8.5. Lazy Loading

8.6. Hot-Module Replacement

8.7. Time-travelling

## 9. WebServices

9.1. SOAP

9.2. REST

9.3. API Contracting

## 10. Browser WebServices

10.1. Protocols: HTTP, Comet, WebSockets, SSE

10.2. Webservice APIs: XHR, fetch, events, socket.io

10.3. Async APIs: callbacks, events, promises, async/await, CSP,  
reactive streams

## 11. Sessions

11.1. Server Sessions

11.2. JSON Web Tokens