Warsztat Ekspercki TypeScript

Ducin IT Consulting

Statyczne Typowanie na Frontendzie

Ducin IT Consulting - Program szkolenia

Czas trwania: **3 dni**
Formuła: **30% teoria, 50% ćwiczenia, 20% praca w grupie**

Trening przygotowuje programistów do efektywnego stosowania TypeScriptu w projektach poprzez szukanie balansu pomiędzy prostotą, elastycznością i bezpieczeństwem typów w kodzie.

Szkolenie przeznaczone zarówno dla programistów frontendowych jak i backendowych. Zakres obejmuje - w zależności od potrzeb grupy - od podstawowych problemów i ich rozwiązań, poprzez zaawansowane, aż po zagadnienia eksperckie.

Szkolenie kładzie nacisk na statyczne typowanie jako alternatywę do dynamicznego JavaScriptu. Porównujemy implementacje dynamiczne i statyczne, przyglądając się zaletom płynącym z większej kontroli nad typami danych oraz kosztom, jakie niesie ze sobą stosowanie TypeScriptu w projektach o różnej skali. Poznajemy wzorce projektowe stosowane we frontendzie oraz implementują niektóre z nich. Wiele ćwiczeń realizowanych jest na podstawie zautomatyzowanych testów w stylu TDD.

Szkolenie przewiduje także zadania związane z projektowaniem aplikacji. Uwzględniane czynniki to elastyczność, łatwe refaktoringi w przyszłości i testowalność.

Kluczowe Aspekty:

- Zrozumienie celów i strategii TypeScripta, ułatwiające intuicyjne stosowanie języka
- Świadome posługiwanie się typami z uwzględnieniem elastyczności kodu oraz unikaniu dziur w type-safety
- Praktyczny TypeScript w średnich/dużych projektach, wzorce projektowe i architektura

Ducin IT Consulting

Program szkolenia:

1. (Optional) Prerequisites
    1.1. Chosen Elements of ECMAScript
2. Introduction
    2.1. TypeScript Design Goals and Philosophy
    2.2. Compile-time & Runtime
    2.3. Type Safety
    2.4. TS Breaking Changes
3. Type System
    3.1. Static vs Dynamic Typing
    3.2. Strong vs Weak Typing
    3.3. Implicit vs Explicit Typing, Type Inference
    3.4. Structural vs Nominal Typing, Duck Typing
    3.5. Soundness and Completeness
    3.6. Variance (Invariance, Covariance, Contravariance, Bivariance)
4. Fundamental & Advanced Types
    4.1. Primitive Types
    4.2. Top and Bottom Types
    4.3. Enums, String Literals, Tuples
    4.4. Unions, Intersections, Index Signatures
    4.5. Function Types
    4.6. Mapped Types
    4.7. Conditional Types
5. Generics

- 5.1. Generic Types
- 5.2. Class Generics
- 5.3. Function Generics
- 5.4. Required vs Inferred Generics

6. Type Compatibility

- 6.1. Primitives
- 6.2. Object Types
- 6.3. Function Types
- 6.4. Unions and Intersections, Mixed Types
- 6.5. Aliases, Interfaces, Classes, Objects

7. Object-Oriented Programming with TS

- 7.1. Applying SOLID Principles to TS
- 7.2. Encapsulation
- 7.3. Structural Polymorphism
- 7.4. Classes and Inheritance
- 7.5. Reusable Decorators (Parameters, Properties, Methods, Classes)

8. Functional Programming with TS

- 8.1. Typing Higher Order Functions with Generics
- 8.2. Functional Composition
- 8.3. Overloading Function Signatures

9. Patterns

- 9.1. Type-safe Error Handling
- 9.2. Opaque/Brand Types
- 9.3. Value Objects
- 9.4. Data Transfer Objects
- 9.5. Domain Logic in frontend layer

Ducin IT Consulting

10. Contract-First Design

- 10.1. TS interfaces contracts
- 10.2. Runtime Validation
- 10.3. JSON Schema
- 10.4. RAML/swagger-based contracts

11. Ecosystem

- 11.1. Compiler Flags
- 11.2. Incremental Builds
- 11.3. Automation: Webpack, Parcel
- 11.4. Handling dependencies
  - 11.4.1. DefinitelyTyped, npm @types
  - 11.4.2. writing custom .d.ts declarations

12. (Optional) TypeScript and legacy code

- 12.1. Incremental Migration
- 12.2. Adjusting TypeScript Config & Compiler Flags
- 12.3. Study case