

Program szkolenia:

JavaScript dokładniej - dla tych, którzy chcą się zagłębić

Informacje:

Nazwa:	JavaScript dokładniej - dla tych, którzy chcą się zagłębić
Kod:	JS-deeper
Kategoria:	JavaScript
Odbiorcy:	developerzy
Czas trwania:	3-4 dni
Forma:	20% wykłady / 80% warsztaty

Wymagania co do znajomości JS zmieniły się przez ostatnie kilka lat. Pobieżna znajomość języka i API modnego dzisiaj frameworka już nie wystarczą. Poza tym frameworki się zmieniają, a język pozostaje na dłużej. W części praktycznej będziemy robić TDD (Test-Driven Development) najciekawszych ficzerów języka i idiomów, wzbogacone o tworzenie modeli koncepcyjnych, testy wydajnościowe oraz dyskusje grupowe w bezpiecznym środowisku. Dziesiątki bardzo starannie zaprojektowanych ćwiczeń pomoże Ci lepiej poznać tajniki języka i przygotuje do pisania nie tylko prostych aplikacji, ale także reużywalnych bibliotek. Założenie: najlepiej zrozumiesz działanie czegośkolwiek, implementując to samemu.

Zalety szkolenia:

- Bardzo szczegółowe spojrzenie na język JavaScript
- Porównanie podejścia obiektowego i funkcyjnego w kontekście JS
- Nauka z wykorzystaniem testów jednostkowych zamiast slajdów

Szczegółowy program:

1. Typy - dlaczego większość porad na temat koercji typów jest zbyt uproszczona

- 1.1. Implementujemy algorytmy konwersji dla operacji takich jak ToBoolean czy ToPrimitive
- 1.2. Budujemy ogólny model konwersji między typami
- 1.3. Wyrabiamy intuicję kiedy stosować koercję typów, a kiedy jej unikać
- 1.4. Budujemy algorytm siedzący pod maską ==
- 1.5. Dekomponujemy głęboko zagnieżdżone struktury danych z użyciem destrukuryzacji

2. Programowanie obiektowe - dlaczego klasyczne OO w JS jest niepotrzebnie skomplikowane

- 2.1. Tworzymy zaawansowane modele koncepcyjne dla programów opartych o funkcje konstruktora/klasę
- 2.2. Tworzymy zaawansowane modele koncepcyjne dla programów opartych o linkowanie obiektów
- 2.3. Rozróżniamy pomiędzy: .prototype, [[Prototype]], __proto__
- 2.4. Poznajemy new i instanceof od podszewki
- 2.5. Odkrywamy niuanse mixinów z Object.assign()
- 2.6. Rozpoznajemy sytuacje, w których abstrakcja klasy wycieka

3. Funkcje i zasięg widoczności - w pełni wykorzystujemy możliwości, które dają proste funkcje

- 3.1. Budujemy dokładniejszą reprezentację umysłową tego czym tak na prawdę jest hoisting, modelując kontekst wywołania
- 3.2. Budujemy dokładniejszą reprezentację umysłową funkcyjnego zasięgu widoczności na podstawie stosu kontekstów wywołania
- 3.3. Wyciągamy niuanse zasięgu blokowego i przyglądamy się sporom na temat użycia const, let i var
- 3.4. Rozkładamy funkcje na czynniki pierwsze (parametry vs argumenty, domyślne wartości, zmienna liczba argumentów)
- 3.5. Porównujemy 4 sposoby dynamicznego wiązania this

3.6. Testujemy i porównujemy wydajność i zużycie pamięci kodu opartego o domknięcia i o prototypy

3.7. Szukamy przypadków kiedy nowości z ES6 tj. arrow functions nie powinny być automatycznym zastępstwem dla kodu ES5

4. Programowanie funkcyjne I - zaczynamy używać pragmatyczne narzędzia programowania funkcyjnego bez doktoratu z matematyki, a Twój kod będzie prostszy i bardziej reużywalny

4.1. Implementujemy funkcje wyższego rzędu dostępne w JS (map/filter/reduce/flatMap)

4.2. Implementujemy przydatne funkcje wyższego rzędu niedostępne w JS (np. takeWhile)

4.3. Rozwiązujemy zaawansowane problemy przetwarzania danych z użyciem ciągu funkcji wyższego rzędu

4.4. Optymalizujemy wywołania funkcji z użyciem technik tj. memoization

4.5. Zapobiegamy modyfikacji obiektów przez deepFreeze i wzorce programowania immutable

5. Asynchroniczny JS - zaczniesz świadomie poruszać się w gąszczu opcji zarządzania asynchronicznością w JS

5.1. Analizujemy model współbieżności oparty o pętlę zdarzeń

5.2. Znajdujemy problemy z callbackami inne niż zagnieżdżenia

5.3. Budujemy przydatne utilsy dla Promise: timeout, first, retry

5.4. Używamy async/await aby uzyskać synchronicznie wyglądający kod asynchroniczny

5.5. Implementujemy własną wersję Observable/RxJS aby zrozumieć koncepcję strumieni

6. Modularność - nowoczesne środowiska JS nie potrzebują hacków do pisania modularnego kodu

6.1. Analizujemy kod programu opartego o wzorce Revealing Module oraz Dependency Injection

6.2. Używamy natywnych modułów ES6 w Node.js i w przeglądarce bez dodatkowych narzędzi

6.3. Implementujemy własny Event Emitter aby lepiej zrozumieć wzorzec Observer

6.4. Porównujemy wzorce komunikacji oparte o orkiestrację i choreografię

7. Programowanie funkcyjne II - rozrywka intelektualna dla fanów programowania funkcyjnego

7.1. Poznajemy różnice pomiędzy partial application i currying

7.2. Zastępujemy obiektowe Dependency Injection z konstruktorami, funkcyjnym Dependency Injection z curry

7.3. Poznajemy wady i zalety stylu programowania point-free

7.4. Budujemy abstrakcje do łączenia funkcji w użyteczne bloki: compose i pipe

7.5. Aplikujemy Funktory i Monady do rozwiązywania praktycznych problemów

7.6. Zastępujemy null/undefined komponowalnym Maybe

8. Metaprogramowanie - wzbogacamy możliwości języka

8.1. Dodajemy możliwość iterowania po obiektach

8.2. Budujemy bardziej wszechstronną wersję async/await z użyciem generatorów

8.3. Wzbogacamy możliwości języka z użyciem Proxy

8.4. Budujemy własny DSL z użyciem tagged template literals