BO·TT·EGA
IT minds

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
Grupa docelowa:	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.
JS można krytykować za błędy projektowe, natomiast faktem jest, że jest to dominujący język w przeglądarkach i coraz popularniejszy język po stronie serwera. Czas go poznać na głębszym poziomie. Język, nie kolejne API, narzędzie, framework czy bibliotekę.

Budując ekspercką intuicję, będziesz mógł/mogła skoncentrować się na problemach biznesowych a nie składni i idiomach języka. Te ostatnie staną się Twoją drugą naturą.
Niczym kierowca Formuły 1 rozumiejący doskonale to co siedzi pod maską samochodu, będziesz rozumieć co siedzi pod maską języka JS oraz najpopularniejszych bibliotek open source.
I w końcu wyrobisz sobie własny styl programowania oparty o pełniejsze zrozumienie języka a nie od lat powtarzane mantry w stylu "zawsze rób X", "nigdy nie rób Y".

Efekty uczenia

- Będziesz bardziej krytycznie patrzeć na programowanie obiektowe w JS z użyciem new, prototype i this
- Docenisz prostotę ludzkich części programowania funkcyjnego w JS
- Nagniesz swój mózg dotykając "nieludzkich" części programowania funkcyjnego
- Będziesz bardziej świadomie dobierać zaawansowane wzorce programowania asynchronicznego do problemu
- Będziesz lepiej rozumieć reguły koercji typów w JS
- Porównasz 4 generacje zarządzania modułami w kodzie JS
- Podniesiesz ekspresywność swojego kodu dzięki wybranym elementom ES6
- Poznasz możliwości i niebezpieczeństwa zaawansowanego metaprogramowania
- Zaczniesz patrzeć na JS w większej rozdzielczości i dostrzegać subtelne niuanse

Sposób nauki

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. Ponad 60 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 czegokolwiek, implementując to samemu.

W części teoretycznej omawiamy przygotowane przez mnie wcześniej testy jednostkowe opisujące mechanizmy języka.

BO·TT·EGA
IT minds

Czym to szkolenie NIE jest

- kursem node.js
- kursem z API przeglądarkowych (DOM, XMLHttpRequest, HTML5)
- kursem z frameworka do robienia Single Page Apps
- kursem z architektury aplikacji
- wprowadzającym kursem językowym dla nowicjuszy
- pozbawionym opinii trenera nijakim tutorialem

Zalety szkolenia:

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

BOTTEGA
IT minds

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. Naprawiamy przypadki brzegowe ===

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

2.1. Tworzymy zaawansowane modele koncepcyjne dla programów opartych o funkcje konstruktora/klasy

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 kopiowania z Object.assign()

2.6. Przechodzimy po obiektach z użyciem Object.keys()/getOwnPropertyNames() /getOwnPropertySymbols() oraz Reflect.ownKeys()

2.7. Dodajemy mechanizm mixinów do języka

2.8. 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

BO·TT·EGA
IT minds

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. Zastanawiamy się czym tak na prawdę jest programowanie funkcyjne

4.2. Implementujemy funkcje wyższego rzędu dostępne w JS (map/filter/reduce/find)

4.3. Implementujemy przydatne funkcje wyższego rzędu niedostępne w JS (m.in. flatMap, zip czy takeWhile)

4.4. Dogłębnie analizujemy sposoby zapobiegania modyfikacji obiektów (seal, freeze, deepFreeze itd.)

4.5. Optymalizujemy wywołania funkcji z użyciem technik tj. memoization i tail-call optimization

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 Promisów: finally, timeout, first, retry, collapse

5.4. Implementujemy własne obiekty zgodne z protokołem Iterable, aby lepiej zrozumieć generatory

5.5. Używamy generatorów aby zbudować własną abstrakcję do pisania synchronicznie wyglądającego kodu asynchronicznego

6. Modularność - maksymalizujemy nasze opcje w temacie zarządzanie modułami

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

6.2. Implementujemy własny Event Emitter aby lepiej zrozumieć wzorzec Observer

6.3. Wprowadzamy elementy choreografii do istniejącej aplikacji z użyciem Event Emitter'a

BOTTEGA IT minds

6.4. Programistycznie kontrolujemy kolejność ładowania modułów inspirując się składnią AMD

6.5. Budujemy dokładniejszy model koncepcyjny require, exports i module.exports z CommonJS

6.6. Doświadczamy tzw. JS fatigue refaktorując naszą przykładową aplikację na natywne moduły ES6 (tu będziemy zmuszeni użyć dodatkowych bibliotek)

7. Struktury danych - powrót na studia i notacja wielkiego O

7.1. Implementujemy pełny interfejs Map z ES6 aby lepiej zrozumieć po co je wprowadzono

7.2. Optymalizujemy prędkość odczytów z naszej mapy

7.3. Konwertujemy iterowalne obiekty i obiekty tablicopodobne na tablice

7.4. Robimy destrukturyzację nietrywialnych struktur

8. Programowanie funkcyjne II - rozrywka intelektualna dla fanów programowania funkcyjnego i zrozumiesz czym są monady

8.1. Poznajemy różnice pomiędzy partial application i currying

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

8.3. Poznajemy wady i zalety stylu programowania point-free

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

8.5. Zastępujemy null/undefined komponowalnym Maybe

8.6. Zastępujemy try/catch komponowalnym Either

9. Metaprogramowanie i Domain Specific Languages - podróż w rzadziej używane fragmenty języka aby zrozumieć go na jeszcze głębszym poziomie

9.1. Poznajemy możliwości i niebezpieczeństwa programowania z użyciem ES6 proxy poprzez przykłady

9.2. Budujemy DSL z użyciem tagged template literals i porównujemy je ze zwykłymi template literals