BOTTEGA IT minds

Program szkolenia:

TDD dla Node.js

Informacje:

Nazwa:	**TDD dla Node.js**
Kod:	**craft-test-Node**
Kategoria:	Testowanie automatyczne
Grupa docelowa:	developerzy architekci
Czas trwania:	2 dni
Forma:	20% wykłady / 80% warsztaty

TDD - praktyka, narzędzie, strategia wytwarzania oprogramowania. Pomaga utrzymać skupienie i projektować dobre API w skali mikro. Jak każde inne narzędzie czy praktyka sprawdza się w wybranych kontekstach.

W trakcie 1 dniowego szkolenia Test-Driven node.js nauczysz się projektować aplikacje node.js z użyciem testów. Będziemy praktykować tzw. Londyńską szkołę TDD znaną również jako outside-in development. Wielkość problemu to ok. 150 linii kodu produkcyjnego i 300 linii testów. Nauka odbywa się w 90% przez ćwiczenia praktyczne, wprowadzamy tylko minimum teorii niezbędnej do rozwiązywania problemów. Każda wersja tego szkolenia jest inna. To jak będzie wyglądała skończona aplikacja zależy od decyzji projektowych grupy.

Uczestnicy będą zachęcani do robienia tzw. ping pong pair programming.

Po każdym kroku trener wrzuca referencyjne rozwiązanie na github'a według tego jakie decyzje podjęliśmy jako grupa.

Nie będzie żadnych skrótów, szkolenie jest symulacją codziennej pracy z TDD w rzeczywistych projektach.

Czego się nauczysz

- Projektować czysty i testowalny kod sterowany małymi, skupionymi na jednej rzeczy testami.
- Patrzeć na testy przez pryzmat ROI (Return on Investment), unikać testowania tego samego po kilka razy i nadmiernej specyfikacji.
- Projektować testowalny kod z "trudnymi" zależnościami tj. system plików, HTTP czy standardowe wyjście.
- Wybierać najbardziej efektywny sposób testowania kodu asynchronicznego, gdzie część programu wykonuje się teraz, a część później.
- Uzupełniać taktyczne decyzje TDD, strategicznymi decyzjami przy tablicy oraz zbierać potrzebne informacje za pomocą spike'ów.
- Słuchać feedback'u od testów i odpowiednio poprawiać strukturę kodu oraz API które wystawiamy naszym kolaboratorom.
- Zastępować biblioteki i narzędzia, praktykami i technikami dostępnymi na poziomie mechanizmów języka i platformy node.js. Np. mock, stub, spy bez narzędzi/bibliotek.
- W łatwiejszy sposób przekazywać intencje kodu produkcyjnego oraz kodu testów z wykorzystaniem nowych elementów ES6 i decydować kiedy ich użycie ma sens.

BO·TT·EGA
IT minds

- Stosować najbardziej przystępne elementy programowania funkcyjnego aby zmaksymalizować nasze opcje i wyjść poza programowanie z this/new/klasami.
- Kwestionować wiele mainstreamowych narzędzi i praktyk. Wytłumaczę np. dlaczego nie używam bibliotek tj. nock, sinon, chai albo dlaczego DRY w testach jest przereklamowane.

Zalety szkolenia:

- Londyńska szkoła TDD
- Duży nacisk na elementy programowania funkcyjnego
- Nietrywialne przykłady

BO·TT·EGA
IT minds

Szczegółowy program:

1. TDD

1.1. Praktyka, Narzędzie, Strategia - patrzenie na TDD z różnych perspektyw

1.2. TDD jako narzędzie taktyczne - projektowanie prostych i rozszerzalnych API w skali mikro

1.3. TDD a decyzje strategiczne - czyli dlaczego TDD nie zastępuje a wzbogaca tablicę i markery

1.4. Spikes - rozpoznawanie natury problemu zanim zaczniemy pisać testy

2. Testy

2.1. ROI testów - patrzenie na testy przez pryzmat zysku z inwestycji i jak pisać testy które są tanie w utrzymaniu

2.2. Testy jako mechanizm feedbacku - jak testy podpowiadają nam które fragmenty kodu poprawić

2.3. Testy jednostkowe vs testy integracyjne - praktyczne przykłady pokazujące jak dobierać rodzaj testu w zależności od problemu

2.4. Happy path vs unhappy path - testowanie sytuacji wyjątkowych i przypadków brzegowych

2.5. Jak wybierać pomiędzy weryfikacją stanu a weryfikacją interakcji w zależności od kontekstu

3. Trudne zależności

3.1. Jak radzić sobie z trudnym do testowania kodem dotykającym operacji wejścia/wyjścia

3.2. Testowanie zależnośći od systemu plików

3.3. Testowanie interakcji po HTTP

3.4. Testowanie kodu który wypisuje dane na standardowe wyjście

3.5. Używanie zamienników testowy tj. mock, spy, stub z pomocą narzędzi i bez

4. TDD a kod asynchroniczny

4.1. Jak radzić sobie z dodatkową złożonością kodu asynchronicznego w testach

4.2. Done pattern

4.3. Promisy - jak projektować łatwe do testowania asynchroniczne API

BOTTEGA IT minds

4.4. Promisy i generatory w testach - pisanie testów kodu asynchronicznego, które respektują sekwencyjne działanie naszego mózgu

5. TDD z ES6

5.1. Podnosimy ekspresywność naszego kodu produkcyjnego i testowego z użyciem nowych elementów języka

5.2. Generatory

5.3. Promisy

5.4. Arrow functions

5.5. Destructuring

5.6. Block scope

6. TDD z elementami programowania funkcyjnego

6.1. Myślenie w kategorii przepływu danych

6.2. Pisanie prostszego kodu z wykorzystaniem "ludzkich" części programowania funkcyjnego

6.3. Funkcje wyższego rzędu

6.4. Domknięcia

6.5. Kompozycja

6.6. Pure functions

7. Narzędzia

7.1. Jak ograniczyć użycie narzędzi do testowania i jak sam język daje nam bardzo przydatne mechanizmy do testów

7.2. Mocha

7.3. co-mocha

7.4. Node assert

7.5. testdouble.js