Crashtest

Tomasz Janusz

Chief Technology Officer at Indoorway (Daftcode Group)

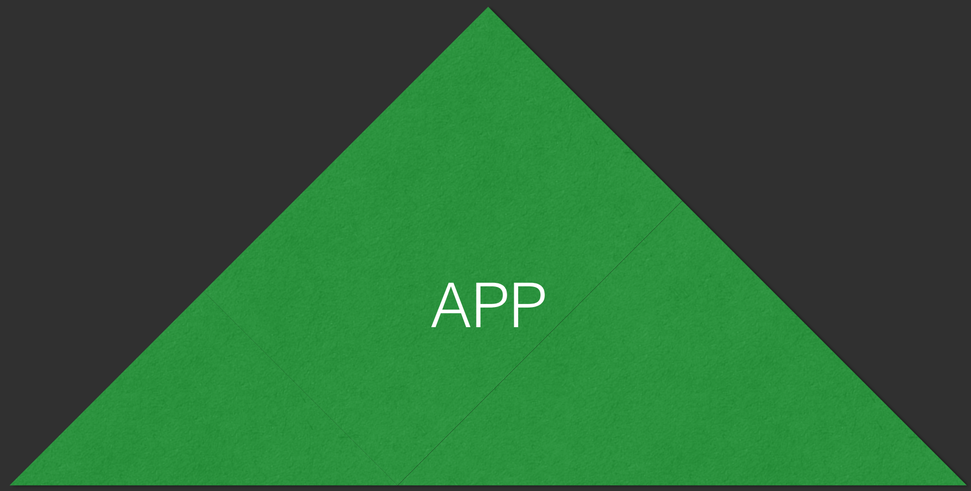
Testy automatyczne

Dlaczego testować kod?

- Obiektywny i zaplanowany tok działania aplikacji
- Aplikacja może być testowana pod kątem podawania błędnych parametrów do metod
- Przyspieszanie rozbudowę aplikacji
- Skracanie czasu weryfikacji zmian

Rodzaje testów

- End-to-end
- Integracyjne
- Jednostkowe
- Statyczne



End-to end

Czy workflow działa poprawnie?

- Ostatni etap testów przed akceptacyjnymi
- Skupiają się na testach określonych scenariuszy (Czy użytkownik może się zalogować i wylogować?)
- Przykładowe narzędzia: Cypress, Protractor

Integracyjne

Czy jednostki współpracują ze sobą?

- Dopuszczane jest testowanie komunikacji np. Z API, interfejsami użytkownika, bazami danych etc.
- Testy zazwyczaj używane w środowisku stagingowym
- Przykładowe narzędzia: ?

Statyczne

- Najczęściej zautomatyzowana analiza kodu pod kątem możliwych zagrożeń czy błędów w architekturze.
- Runnery posiadają zaimplementowane wcześniej zestawy reguł
- Testy zazwyczaj używane w środowisku programistycznym
- Przykładowe narzędzia: … :)

Jednostkowe

.

- Skupiają się na testowaniu w odizolowanym środowisku wydzielonych modułów (klas, metod, etc).
- Testy często wykorzystują niskopoziomowe API udostępniane przez runnery czy frameworki.
- Testy zazwyczaj używane w środowisku programistycznym
- Przykładowe narzędzia: Jest, Mocha, type

Dobre techniki

- Testuj tylko jedną metodę na raz
- Pisz testy tak, aby każdy z nich mógł być uruchomiony osobno
- Testuj działanie funkcji na różne typy danych
- Zawsze sprawdzań wynik działania funkcji
- Test Driven Development

To możesz pominąć

.

- Nie testuj metod, które komunikują się z zewnętrznymi modułami bez mocków
- Nie testuj wykorzystywanych bibliotek ^^’
- Omijaj ‘common’ code
- **Nie twórz testów bez sensu – wydłużają one proces CI**

Jest!

https://jestjs.io

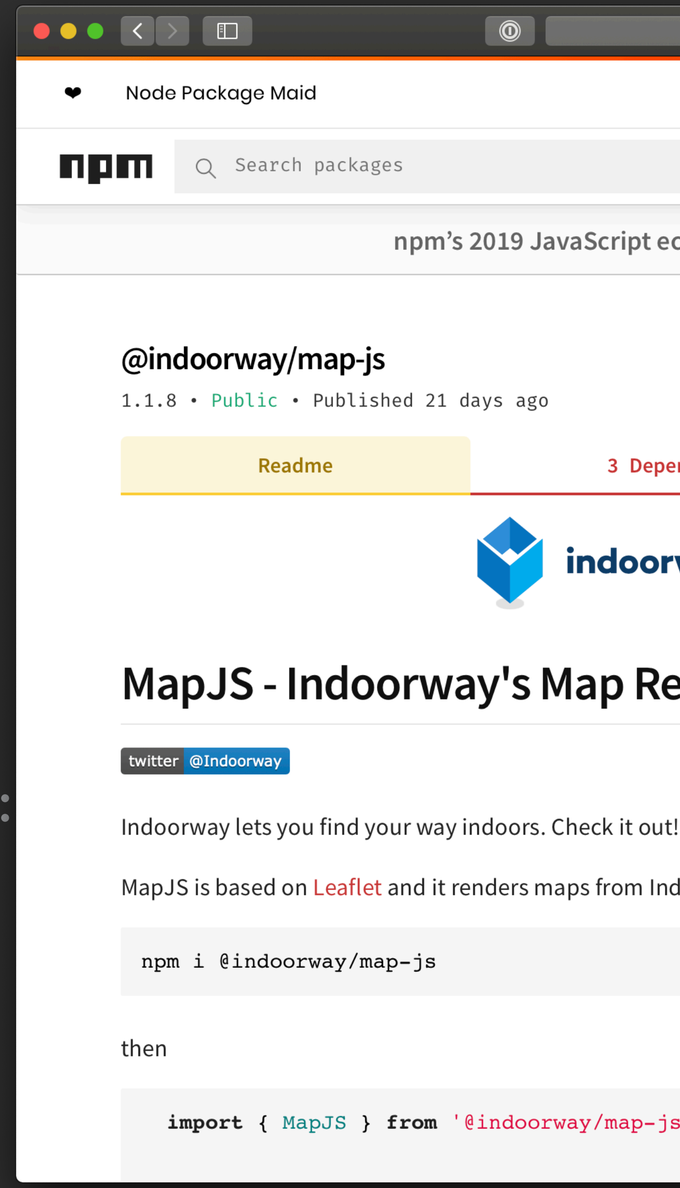
- Szybki - testy wykonują się równolegle
- Minimalna konfiugracja
- Wsparcie programisty - kolorowe i sformatowane komunikaty błędów
- Działa dobrze w dużych środowiskach - testy mogą być wznawiane w miejscu, gdzie zatrzymały się wcześniej

Jest API
- w zasadzie nie da się prościej 🤌

- **Describe** - jak Jest rozumie moduł
  więcej na: https://jestjs.io/docs/en/api#describename-fn
- **Except** - moduł z praktycznie każdym możliwym wariantem sprawdzania wyniku działania funkcji
  więcej na: https://jestjs.io/docs/en/expect
- **Mock** - kompletnie API do tworzenia mocków danych
  więcej na: https://jestjs.io/docs/en/mock-functions

Indoorway MapJS

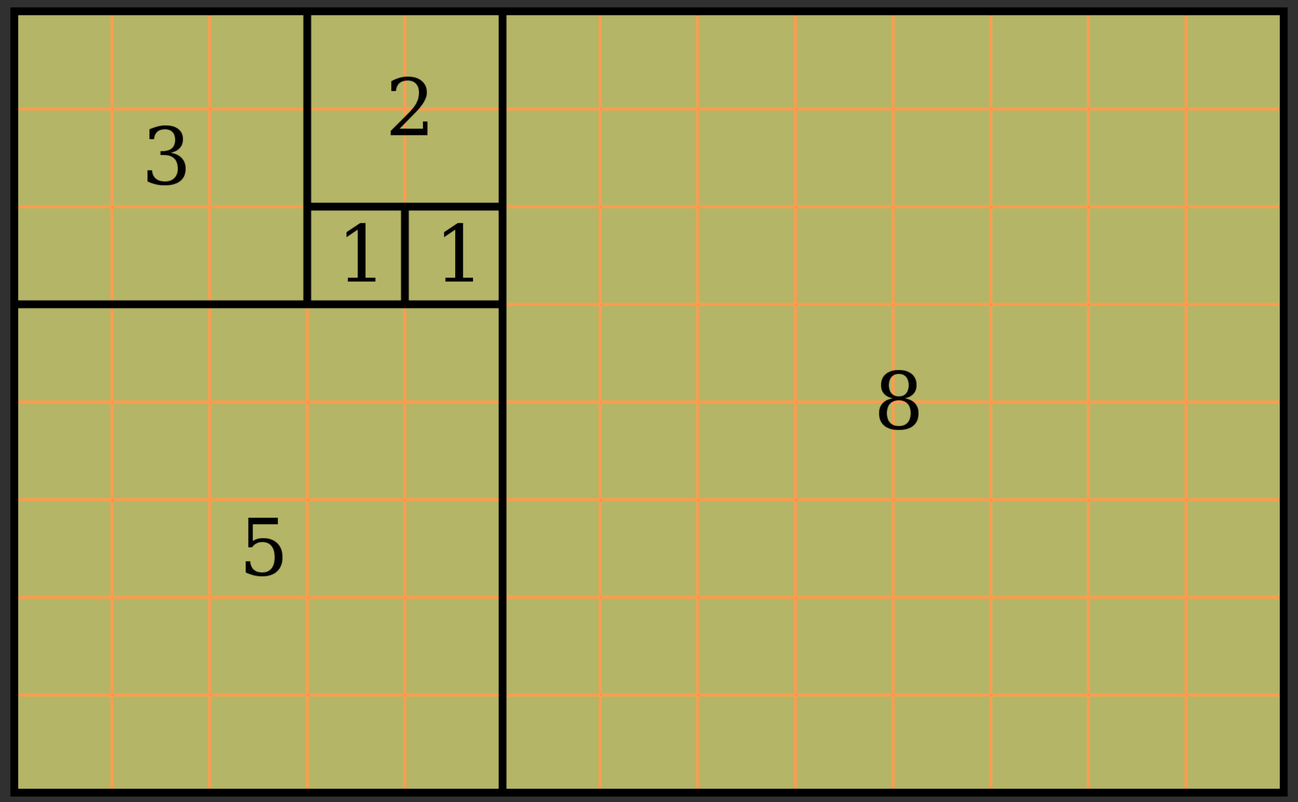
- Biblioteka Indoorway dla klientów zewnętrznych
- Umożliwia wyświetlanie danych na innych stronach internetowych
- Ma testy w Jest I możemy sobie przy nich posiedzieć :



DevTools

Zadanie domowe

?


3
2
1
1
8
5

0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597

Ciąg Fibonacciego

$$F(n) = F(n-1) + F(n-2)$$

ZADANIE 1:

1 pkt

Stwórz ciało funkcji dla poniższego interfejsu:

fib(integer num): [integer]

Funkcja fib() ma zaimplementować formalny wzór na Ciąg Fibonacciego. Parametrem wejściowym jest liczbą wyrazów zwróconych w tablicy po zakończeniu działania fubnkcji. Funkcja powinna sprawdzać poprawność danych wejściowych.

- Nie bój się modyfikować liczby parametrów funkcji. Twoja implementacja Ciągu może zakładałać istnienie więcej niż jednego parametru wejściowego. Ważne, aby dodatkowe parametry były opcjonalne i parametr num był pierwszy.

- Więcej o Ciągu Fibonacciego możesz przeczytać m.in. na Wikipedii

ZADANIE 2:

0 - 4 pkt.

Stwórz zestaw testów jednostkowych bazujący na Jest dla Twojej implementacji. Za każdy z napisanych i działających testów możesz zdobyć 1 pkt - kryterium będzie zasadność istnienia tego testu.

- w teście możesz osadzić komentarz tłumaczący dlaczego wybrałeś określone podejście przeprowadzenia testu

Dzięki i do zobaczenia!
8.05.2018

Dosłownie wszędzie: @TomaszJanusz