



**Wyższa Szkoła Ekonomii
i Informatyki w Krakowie**

Unit testy - Angular

Testy jednostkowe w Angular

- Otoczenie, narzędzia: [stateofjs](#)
- Kultura testów
- Nowy projekt? Czy biznes wie czego chce? TDD?
- Code review testów
- Code coverage. 100%? Focus na jakości, później ilości
- Spróbuj inaczej - nie patrz na % całego kodu, ale procent pokrycia commita - tu zrób 95%:)
- Drabinka Google: we offer the general guidelines of 60% as “acceptable”, 75% as “commendable” and 90% as “exemplary.”
- Uwaga na sposób mierzenia code coverage. To że pokrywamy kod if-a, nie znaczy że testujemy jego wszystkie edge-cases.

Jasmine/Karma

- Jasmine
 - Javascriptowy framework testowy
 - współpracuje z wszystkimi popularnymi bibliotekami do testów
 - bazowe pakiety są instalowane automatycznie z nowym projektem
 - wymaga test-runnera. W projektach angularowych domyślnie jest nim Karma
 - inne popularne biblioteki/frameworki: Jest, Testing library, Mocha,
-
- Karma
 - Javascriptowy test runner

Testy - szybki start

- struktura folderów i nomenklatura: pliki .spec.ts
- Pojedynczy test set: describe, it, xit, xdescribe, fit, fdescribe
- Praca ze środowiskiem przed/po teście: beforeAll, beforeEach, afterAll, afterEach

Mockowanie angularowych modułów

- Testy są izolowane, nie ma inicjalizacji modułów.
- Moduł jest tworzony dynamicznie przez środowisko testowe za każdym razem od nowa.
- W konfiguracji konieczne jest dostarczenie wszystkich zależności, których wymaga testowany element
- Angular dostarcza mocki swoich bazowych modułów, np.:
 - HttpClient -> HttpClientTestingModule
 - RouterModule -> RouterTestingModule
 - ...lub robimy własne mocki:)
- Uwaga na zależności angularowych serwisów - jeśli np. w teście brakuje ActivatedRoute - dołączamy moduł (w wersji dla testów) który ten serwis dostarcza (tutaj: RouterTestingModule)

Fixture w testach - przyspieszanie testów

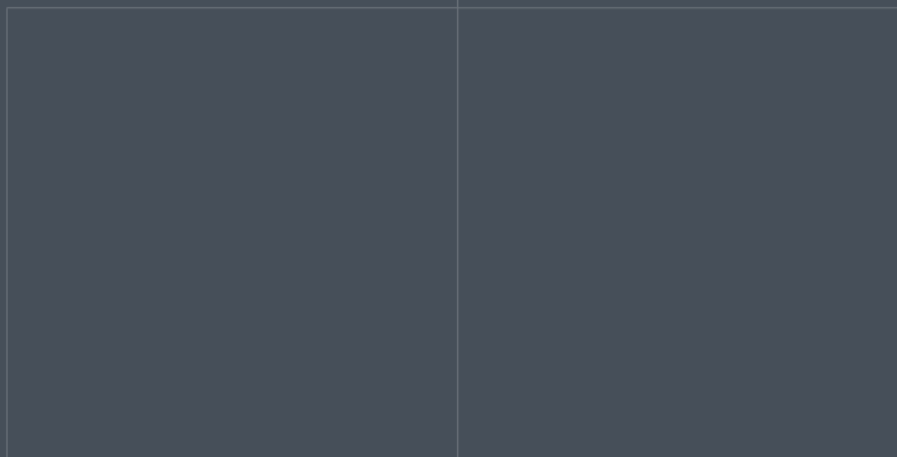
- Fixture nie zawsze jest potrzebne. A jak nie potrzebujemy to nie chcemy żeby zabierało nam czas.
- W szczególności - nie potrzebujemy do wstrzykiwania serwisów oraz testowania logiki komponentów (bez widoków)
- Wstrzykiwanie bez fixture: `TestBed.inject(Dashboard)`
- Inicjowanie komponentu:
 - `TestBed.inject(RxjsComponent)`
 - wtedy trzeba dodać komponent do providers: `[RxjsComponent]` (tak jak serwisy - bo traktujemy go podobnie jak serwis)
- W powyższym tracimy lifecycle, np trzeba ręcznie odpalić `ngOnInit` na komponentcie
- ...ale zyskujemy dużo czystsze i szybsze testy

Fixture - c.d.

- Możemy pójść dalej - nie używamy TestBed.inject() do tworzenia komponentu
- po prostu `const c = new RxjsComponent(myHeroMockService)`
- ..bo koniec końców testujemy KLASĘ komponentu. Nie widok.
- Będzie jeszcze szybciej:)
- Nie można tego zrobić jeśli korzystasz z modułów Angularowych (np. HttpClientModule, RouterModule - te są dostępne jedynie przez DI)
- Powyższe „obcinanie” testów angularowych to tzw. Isolated Unit Test
- Problem z raportem CodeCoverage - może nie pokazywać realnego pokrycia
- Ale nie cyferki są ważne;)

Testy - Visual regression tests

- Testy oparte o snapshoty html/png
- jasmine-snapshot / jest snapshot
- wychwytywanie zmian w wyglądzie
- różnice w wyglądzie w różnych przeglądarkach
- Przykładowe narzędzia:
 - lambdatest
 - backstop
 - rainforest
 - cypress + puppeteer



Wyższa Szkoła Ekonomii
i Informatyki w Krakowie

