Wyższa Szkoła Ekonomii
i Informatyki w Krakowie

Programowanie reaktywne - RxJS

Programowanie imperatywne

```
let del = 1
let fin = 2
let delfin = del + fin

fin = 5
delfin = del + fin
```

WSEI

Programowanie reaktywne

```
let del = 1
let fin = 2
let always delfin = del + fin

fin = 5
// delfin = 6
```

WSEI

Programowanie reaktywne - what the f* is always?

- Natywna funkcjonalność języka
- Ukryta logika frameworka/kompilatora
- Opakowanie na zależności del i fin-a
- setTimeout ("imperatywna" reaktywność ;))
- Observable

WSEI

Kiedy pisać reaktywnie?

- Praca z danymi zmieniającymi się asynchronicznie
- Race conditions
- Aplikacje pracujące z dużą ilością asynchronicznych zdarzeń

- Na froncie pracujesz praktycznie zawsze w środowisku asynchronicznym.

WSEI

Programowanie reaktywne - observables

- Programowanie reaktywne bazuje nie na wartościach, ale na strumieniach wartości
- Kod jest powiadamiany i reaguje na:
  - fakt emisji w strumieniu (emisja może być "pusta")
  - kolejne wartości przychodzące w strumieniu danych
- Przykład: newsletter, tablica lotów, dzwonek szkolny, budzik na 15:50 w pracy;)

WSEI

Reactive Manifesto

- Główne założenia do systemu reaktywnego
  - Responsive - żadna operacja nie blokuje wątku wykonującego kod, system odpowiada „w rozsądnym” dla danej operacji czasie
  - Resilient - system pozostaje responsywny gdy wystąpi błąd/wyjątek
  - Elastic - system pozostaje responsywny przy nagłych zmianach obciążenia
  - Message Driven - operacje wyzwalane są przez emitowane zdarzenia, zdarzenia definiują granicę powiązań pomiędzy komponentami systemu
- Powyższe założenia pozwalają tworzyć systemu elastyczne z luźno powiązanymi zależnościami*

  *luźno powiązane elementy - to takie które rozmawiają za pomocą ogólnie znanego interfejsu
- Implementacja: ReactiveX i jego implementacje (w tym RxJS), Reactor, MobX, cycle.js, .net/reactive, […]

https://www.reactivemanifesto.org/

WSEI

Programowanie reaktywne - rozwój w JS/TS

- Rozwój programowania asynchronicznego w JS/TS:
  - callbacks
  - event listeners
  - promises
  - async/await
  - observables (zewnętrzne biblioteki)

WSEI

Wzorzec projektowy observer

- Wzorzec behawioralny
- Rozwiązuje problem komunikacji zmiany stanu/wystąpienia zdarzenia
- O tym czy odbiorca jest powiadamiany o zmianie stanu decyduje sam odbiorca
- Rozróżniamy role: Observable/Subject, Subscriber/Observer
  - Observable to element do którego zapisują się Observers
  - Subscriber/Observer to element chcący otrzymywać powiadomienia emitowane przez Observable/Subject
  - Producer - element emitujący dane do Observable/Subject

WSEI

Wzorzec projektowy observer

- Stosujemy gdy zmiany stanu Producera wpływa na Subskrybentów
- Zmiany stanu mogą być synchronicznie i asynchroniczne
- Subskrybenci mogą być dynamicznie dodawani i usuwani - komunikacja jeden do wielu lub jeden do jeden
- Nie ma gwarancji kolejności powiadamiania subskrybentów

WSEI

RxJS - klocki

- **Observable** - strumień który możemy obserwować
- **Observer** - obiekt przechowujący callbacki uruchamiane podczas emisji wartości ze strumienia. To Observer wie jak pobierać wartość (subskrypcja) ze strumienia. Observerem może być również pojedynczy callback
- **Subscription** - obiekt powstały w procesie subskrypcji. Przechowujemy by się wypisać;)
- **Operators** - czyste funkcje pracujące jako „middleware” w strumieniu. Zapisują się do strumienia, zwracają strumień. Podczas każdej emisji realizują swoją założoną funkcjonalność.
- **Subject** - obiekt który pozwala emitować wartości do wielu subskrybentów (hot observable)
- **Scheduler** - „dispatcher” strumienia. W zależności od rodzaju schedulera pozwala wyzwalać wartości ze strumienia w oparciu o m. in. czas, zdarzenie przerysowania ekranu, asynchroniczną operację

WSEI

Programowanie reaktywne - RxJS

- Podstawowe klasy tworzące strumienie (i emitujące wartości) to **Observable, Subject, BehaviorSubject, ReplaySubject, AsyncSubject.**
- Dostępne są również dodatkowe funkcje tworzące strumienie z tablic, zdarzeń, obietnic czy np. emitujące wartości co określony czas.
- Observable może wyemitować wartość (next), błąd (error), lub zakończyć się (complete)
- Do pracy ze strumieniami RxJS oddaje szeroki zestaw dodatkowych operatorów. Np. mapowanie danych, filtrowanie, opóźnianie wartości, wybieranie tylko niektórych wartości ze strumienia, łączenie wielu strumieni itd.
- KONIECZNIE należy pamiętać o zakończeniu subskrypcji gdy już jej nie potrzebujemy

WSEI

RxJS - konwencje nazewnictwa Observable

- kolekcje danych: clients/jokes$,
- pojedyncze wartości: token/user$
- akcja na wartość ze strumienia: saveForm/animateCircle$
- **z $ lub bez. jak projekt woli.**

- plusy $
    - easy to see (huuuge)
    - wygodne jak pobieramy wartość do zmiennej: user$ --> user. Tylko czy chcemy pobierać wartość do zmiennej?
- minusy $
    - jak nazwiesz tablicę observabli?
    - dlaczego nie dajesz suffixu do np. promise, funkcji lub sub-a?
    - pójdziemy w kolejne suffixy? To już było (hungarian notation)!
    - (nie)wygoda pisania
    - czy jeśli funkcja zwraca Observable to też powinna mieć sufiks $?
    - czy rozróżniamy w $ Observable od Subject, BehaviorSubject itd?

- nie pytaj czy funkcja asynchroniczna powinna mieć prefix async... This. is. war.

WSEI

RxJS - Hot observable

- Producer pracuje niezależnie od subskrybcji Observabla
- Producer może pracować bez żadnego obserwatora
- Przykłady: Subject, BehaviorSubject, ReplaySubject, AsyncSubject
- Wszyscy subskrybenci dostają tę samą wartość - jeden producer jest współdzielony
- Hot observable to (prawie zawsze) emiter multicast
- Najczęstszy problem z hot - brak dostępu do wartości wyemitowanych przed zapisaniem się (rozwiązanie: operatory, ReplaySubject)

WSEI

RxJS - Cold observable

- Przykładem Cold Observable jest strumień z new Observable().
- Posiada jedno źródło emisji (producer),
- Producerem przekazywanym do observable jest funkcja
- Cold observable uruchamia producera dopiero w momencie pojawienia się obserwatora (np. callback subskrypcji).
- Każdy nowy obserwator uruchamia producer na nowo
- Cold observable jest emiterem unicastowym

WSEI

RxJS - Hot & cold observables

- Konwersja Cold->Hot: operatory connect, share
- Konwersja w poprzednich wersjach RxJS: operatory publish (+refCount), publishBehavior, publishReplay, publishLast - wszystkie wylatują w RxJS8

WSEI

RxJS - podziały strumieni

- Synchroniczne/asynchroniczne
- Hot/Cold
- Unicast/Multicast

WSEI

RxJS - subskrypcja do emitera

- Observable.subscribe()
- operatory - share, merge, zip, connect, […]
- Observable.toPromise() -> firstValueFrom/lastValueFrom()
- Observable.forEach() - zwraca Promise, działa jak .subscribe

- Pobranie snapshot-a z BehaviorSubject bez subskrypcji: bs.value

WSEI

RxJS - unsubscribe

- .unsubscribe()
- .complete() na observable
- własny dekorator na właściwości klasy lub na klasę (tricky)
- operatory timeout, take, takeWhile, takeUntil, first - śliska sprawa
- Subscription.add()/unsubscribe

- Gdzie nie musisz:
    - gdy observable się kończy (i mamy pewność otrzymania wartości)
    - gdy sami zakończymy observable
    - na poziomie „root” aplikacji
- Uwaga na „zawieszone” Promise pochodzące np. .toPromise, firstValueFrom(), lastValueFrom()

WSEI

RxJS - operatory

- Operatory to funkcje które pobierają Observable(s) i zwracają *nowy* Observable
- Pipeable operators - wywołujemy je w .pipe()
- Creation operators - używamy jak zwykłej funkcji.

- Operatory filtrujące: first, last, elementAt, skip, **filter**, sample, debounce(Time), distinctUntilChanged, **take**(While/Until), every, find(Index)
- Operatory transformujące: **map**, ~~**pluck**~~, scan, reduce
- Operatory tworzące: **from**, of, **fromEvent**, interval, timer, generate,
- Obsługa błędów: catchError, retry(When)
- Operatory warunkowe: iif, every, defaultIfEmpty
- Operatory pomocnicze: **tap**, delay, timeout(With), toArray, startWith, endWith ,defer
- Aktualna lista: https://rxjs.dev/guide/operators

WSEI

RxJS - helper

- https://rxjs-dev.firebaseapp.com/operator-decision-tree
  Uwaga: część operatorów w sugestiach jest już oznaczona jako deprecated

WSEI

Wyższa Szkoła Ekonomii
i Informatyki w Krakowie