Wyższa Szkoła Ekonomii
i Informatyki w Krakowie

Programowanie reaktywne
RxJS

Programowanie reaktywne

- Rozwój programowania asynchronicznego w JS/TS: callbacks -> event listeners -> promises + async/await -> observables
- Programowanie reaktywne bazuje nie na wartościach, ale na strumieniach wartości
- Kod jest powiadamiany i reaguje na kolejne wartości przychodzące w strumieniu danych
- Angular do komunikacji i wymiany danych za pomocą strumieni wykorzystuje bibliotekę RxJS
- W oparciu o obiekty Observables działa wiele wbudowanych w Angular kluczowych serwisów i obiektów, m.in:
- HttpClient,
- ReactiveForms
- Router
- EventEmitter

WSEI

Programowanie reaktywne

- 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 - subskrypcja do emitera

- Pipe async
- Observable.subscribe()
- operatory - share, merge, zip, connect, […]
- Observable.toPromise() -> firstValueFrom/lastValueFrom()
- Observable.forEach() - zwraca Promise, działa jak .subscribe
- Pobranie snapshot-a z BehaviorSubject: bs.value

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
- „Bolesnym” przykładem w Angularze może być HttpClient
- Cold observable jest emiterem unicastowym

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 - 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 - 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 strumienia do zmiennej?

minusy $

- 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?

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 - operatory standalone i operatory pipeable pracujące na wielu strumieniach

- race, raceWith - zwraca pierwszy ze strumieni który zrobił next | err | complete
- zip, zipWith - sekwencyjnie i jednocześnie emituj wartości z każdego źródła w postaci tablicy
- combineLatest, combineLatestWith - dla każdej emisji z dowolnego źródła wyemituj komplet ostatnich wartości
- forkJoin - wyemituj ostatnie wartości gdy wszystkie observables się zakończą
- merge - złącz emisje w jeden strumień
- concat, concatWith - po zakończeniu pierwszego observabla emituj drugi, następnie kolejny itd
- switchMap, mergeMap, concatMap, exhaustMap - dla każdej emisji A, zapisz się do B

WSEI

RxJS - własne operatory

- Na podstawie pipe()
  Najczęstszy case - mamy zestaw operatorów w pipe() który jest wspólny dla wielu observabli
  Przykład: rxjs/many-> sumujIWyswietl

- Od zera. Tylko po co:)
  Funkcja musi implementować interfejs
  (obs: Observable<T>) => Observable

  Należy pamiętać o pełnej implementacji Observable (next, error, complete)

WSEI

RxJS - unsubscribe

- .unsubscribe()
- async pipe
- .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 robisz sub-a na poziomie „root” (np. AppComponent, AppModule, serwisy providedIn: root)
  - …ale łatwo zapomnieć o .unsubscribe() przy refaktorze:(

- Uwaga na „zawieszone” Promise pochodzące np. .toPromise, firstValueFrom(), lastValueFrom()

WSEI

Higher order observable

- Higher order observable to taki observable którego wartościami są inne observable, np:

```
subject.next(of(„Jan”, „Nowak”, „Kraków”))
subject.next(of(„Katarzyna”, „Kowalska”, „Gdańsk”))
subject.next(of(„Magdalena”, „Wiśniewska”, „Wrocław”))
```

- operatory:
  - concatAll
  - combineLatestAll
  - switchAll
  - mergeAll
  - exhaustAll
  - zipAll

WSEI

RxJS - helper

https://rxjs-dev.firebaseapp.com/operator-decision-tree

Uwaga: część operatorów w sugestiach jest już oznaczona jako deprecated

WSEI