FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Labo Gebruikersinterfaces
Reeks 3: Reactive programming met RxJS

Doel: Het kunnen toepassen van reactive programming principes via RxJS.
Voorkennis: HTML en Javascript.

1 Inleiding

Reactive programming is een programmeerparadigma gebaseerd op het verwerken van asynchrone data streams. Asynchroon wijst op het feit dat data buiten de main program flow wordt verwerkt. Deze data streams kunnen praktisch gezien alles zijn (bv. input van de gebruiker, GUI events, numerieke data, etc.).

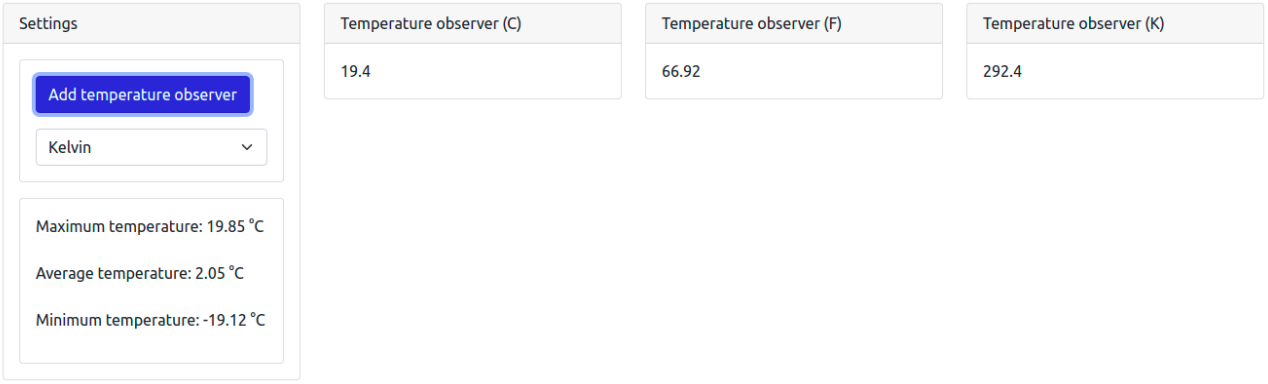
RxJS is een Javascript bibliotheek die ons in staat stelt reactieve programma's te schrijven aan de hand van `Observables`. Naast de kracht van reactive programming biedt RxJS een groot aantal `operators` die kunnen gebruikt worden om bewerkingen uit te voeren op de eerder vermelde data streams.

In deze labosessie zullen we verschillende `Observables` aanmaken die de waardes van een temperatuursensor simuleren. Om vervolgens de temperatuur te verkrijgen zullen we subscriben op deze `Observables`. De structuur en opbouw (HTML) van de website is gegeven en kan je terug vinden op Ufora. De focus van dit labo ligt dan ook op het leren en toepassen van principes uit reactive programming via RxJS. Het is de bedoeling dat wanneer we op de knop *Add temperature observer* klikken, een `Card` wordt toegevoegd die de temperatuur weergeeft. Praktisch zal dit overeenstemmen met subscriben op een `Observable` en eventueel het uitvoeren van een aantal operaties. Iedere `Card` komt dus eigenlijk overeen met een `Observer`. Daarnaast zullen we ook de minimum, gemiddelde en maximum temperatuur bepalen. Op figuur 1 zie je een voorbeeld van hoe de website er uiteindelijk uit moet zien. RxJS is geïmporteerd aan de hand van een CDN (Content delivery network) onderaan het bestand *index.html*. Zoek dat stukje code even op. En als je hier meer over wil weten, gebruik de juiste zoektermen.

Om klassen en functies van RxJS en RxJS Operators te importeren dien je de volgende code te gebruiken:

```
const { Observable } = rxjs;
const { map } = rxjs.operators;
```

Gaandeweg zal je de lijstjes startend met `Observable` en `map` moeten aanvullen. Let op: dit zijn de *eerste* en *enige* regels in het js-bestand die zorgen voor de juiste rxjs-imports; ga dus



Figuur 1: Voorbeeld van de finale website

geen extra regels code toevoegen die starten met `import`.

Sla er op tijd de theorieslides en de officiële documentatie op na via docs.w3cub.com/rxjs/.

2 Constante temperatuur

(a) Open het IntelliJ startproject dat te vinden is op Ufora. Open het bestand *index.html*, rechtermuisklik en kies voor *Run index.html*. Als deze webpagina in je browser wordt geopend zie je een knop *Add temperature observer* en een dropdownlijst waar een eenheid van temperatuur kan worden gekozen.

(b) Wanneer we nu op de knop *Add temperature observer* klikken gebeurt er nog niets. Als eerste willen we dat er een nieuwe `Card` wordt toegevoegd aan onze webpagina. De methode om een `Card` toe te voegen is reeds gegeven, `addCard`. We moeten er dus enkel nog voor zorgen dat deze methode uitgevoerd wordt wanneer op de knop wordt geklikt. Doe dit door gebruik te maken van een `Observable` en dus NIET a.d.h.v. volgende methodes:

```
object.onclick = myFunction;
object.addEventListener("click", myFunction);
```

We houden nog geen rekening met de temperatuurschaal (de dropdownlijst is immers nog niet interactief), dus elke kaart die we maken toont temperaturen in Celsius. En de waarde van de temperatuur is voorlopig `0`.

Merk op: er wordt nu dus een Observable aangemaakt, wat in dit geval staat voor een collectie van toekomstige events. Bij elke gepaste actie van de gebruiker (nl. een klik op de knop) zal er een event aan deze collectie toegevoegd worden. En als reactie op elk nieuw event zal er een nieuwe card aangemaakt worden.

(c) We zullen nu een `Observable` aanmaken die iedere `Card` de temperatuur zal meedelen. Doe dit door gebruik te maken van de constructor van de klasse `Observable`. In deze eerste fase kan je een constante waarde gebruiken om de temperatuur voor te stellen; er wordt dus maar één (hardgecodeerde) waarde doorgegeven aan de observers.

Merk op: er wordt nu dus een Observable aangemaakt, wat in dit geval staat voor een collectie van reeds gekende waarden. De collectie is trouwens ook erg klein: er is maar

één element te bespeuren. Er is ook nog geen enkele Observer die ingeschreven heeft op deze Observable, dus gebeurt er voorlopig niets met dat ene element.

(d) In deze laatste stap gaan we subscriben op de net aangemaakte `Observable` in plaats van meteen een `Card` toe te voegen wanneer op de knop geklikt wordt. Zorg ervoor dat er een `Card` toegevoegd wordt wanneer onze `Observable` een nieuwe waarde ter beschikking stelt.

Merk op: het toevoegen van de Card zat al verweven in de code van deel (b). Het is dus op die plaats dat je het toevoegen van de Card vervangt door het subscriben op de Observable uit deel (c). Gelukkig bevat die Observable maar één element, zodat er maar één actie zal voortkomen uit dat subscriben. Controleer dat: voeg in deel (c) een extra element toe aan de Observable (liefst een ander hardgecodeerd getal dan het eerste), en ga na hoeveel kaarten er dan aangemaakt worden per keer dat er op de knop geklikt wordt.

3 Constante temperatuur met ruis

We passen de geschreven code gradueel aan. Beslis zelf of je de oude code bewaart om later met de oplossing te kunnen vergelijken. (Dat kan door de oude code in een apart bestand op te slaan, of door de nieuwe Observables onder de oude te schrijven, en ze een aangepaste naam te geven.)

(a) Voeg ruis toe aan de hardgecodeerde temperatuur door een random waarde tussen -0.5 en 0.5 er bij op te tellen.

(b) Gebruik een RxJS-operator om de temperatuur af te ronden tot op 1 cijfer na de komma. Zoek in de theorieslides naar de juiste RxJS-operator, en in de online documentatie naar uitleg en voorbeelden.

Merk op: als je op een Observable een RxJS-operator toepast, dan krijg je een nieuwe Observable als resultaat. De collectie van (reeds gekende of toekomstige) gegevens wordt dus omgezet naar een collectie van gegevens die uit de oorspronkelijke berekend worden.

(c) Is de temperatuur voor iedere `Card` (`Observer`) dezelfde? Wilt dit zeggen dat je een cold of een hot observable hebt gemaakt? Indien je een hot observable gemaakt hebt, probeer er dan nu een cold observable van te maken. Indien je aanvankelijk een cold observable hebt gemaakt, maak er dan nu een hot observable van.

(d) Zorg er nu voor dat de `Observable` iedere seconde een update stuurt van de temperatuur. Gebruik hiervoor de JavaScript-methode `setInterval`. Let op want nu zal je in de functie `subscribe` moeten bepalen welke `Card` je moet updaten. Wat zijn hier de verschillen tussen een hot en cold observable en waarom?

Tip: Om bij te houden welke `Card` je moet updaten kan je een teller bijhouden van het aantal `Observers`. Je kan dan een kopie van de teller meegeven met de functie `next`. De waarde van deze lokale kopie kan je dan ook verwerken in de `id` van de `Card` die je toevoegt - als dat al niet vanzelf gaat.

Tip: Je kan meerdere argumenten meegeven aan de functie `next` door deze te verpakken in een `Object`:

```
subscriber.next({subscriber: sub, value: temperature.value});
```

Merk op: omdat de Observable nu verschillende getallen 'spuit' (van het werkwoord spuien), zal de code hard omgegooid moeten worden. Anders zal er voor elk nieuw getal een nieuwe Card toegevoegd worden aan het document. Pas dus de code op twee plaatsen wezenlijk aan.

(1) Telkens er een nieuwe subscriber is voor de Observable zet je een waardenpaar in de lijst van waarden die de Observable genereert: het volgnummer van de subscriber, en een temperatuur. En dit krijgt elke seconde een update.

(2) Bij elke klik op de knop

i. voeg je een Card toe aan het document

ii. bepaal je wat er bij elke nieuwe temperatuur moet gebeuren, door te subscriben op de temperatuur-Observable (nl: haal de juiste Card uit het html-document, en pas daar de temperatuur aan).

Omdat de functie addCard de card wel toevoegt aan het html-bestand (of beter: de DOM-structuur van dat bestand) maar geen returnwaarde heeft, is er in het JavaScript-bestand geen variabele gekend die de net toegevoegde card (of alle toegevoegde cards) bevat. Daarom moet de card opgehaald worden uit de DOM-structuur.

4 Variërende temperatuur en andere temperatuurschalen

(a) Neem nu een kijkje in het script `Signals.js`. Dit script bevat een klasse `sineSignal`. Objecten van deze klasse hebben een veld `value` dat iedere 100 ms wordt bijgewerkt, bekijk gerust eens hoe dit werkt. Wanneer je deze waardes zou plotten, zou je zien dat deze waardes afkomstig zijn van een sinusfunctie. Er wordt één object van deze klasse geëxporteerd: `temperature`. Het veld `value` van dit object stelt dus de temperatuur voor waarvan de waardes fluctueren volgens een sinusfunctie. Importeer dit object in je eigen script om het te kunnen gebruiken.

(b) Vervang de constante temperatuur door het sinusvormig temperatuursignaal. Aangezien de data buiten de `Observable` wordt gegenereerd dienen alle `Observers` dezelfde data te zien (hot observable). Alle `Card`s dienen dus dezelfde temperatuur te tonen.

(c) Lees de eenheid die aangegeven wordt door de dropdownlijst in aan de hand van een Observable.

(d) In deze stap willen we de temperatuur omzetten naar een andere eenheid. Het is niet de bedoeling om een nieuwe `Observable` aan te maken voor iedere eenheid maar de omzetting te doen aan de hand van een RxJS operator.

5 Minimum, gemiddelde en maximum temperatuur

(a) De array `temperatureBurst` (gedefinieerd in "Signals.js") stelt een aantal opeenvolgende temperatuurmetingen voor. Creëer een `Observable` die alle elementen in deze array één voor één uitspuwt.

(b) Bereken de minimum, gemiddelde en maximum temperatuur en plaats deze op de webpagina in de daarvoor voorziene paragrafen. Gebruik een RxJS operator om deze drie

waardes te berekenen.

(c) Je zou deze opdracht ook kunnen uitvoeren aan de hand van de RxJS operator `of`, wat is het verschil?

6 Extra

(a) Maak oefening 5 opnieuw maar gebruik nu de array `temperatureBurstWithErrors`. Als je de waardes van deze array zou printen, zou je zien dat sommige elementen de waarde *error* hebben. Dit komt overeen met een fout tijdens het uitlezen van de temperatuur. Om een correct minimum, gemiddelde en maximum te berekenen moeten we deze *error*'s uit filteren. Gebruik hiervoor de RxJS operator `filter` https://rxjs-dev.firebaseapp.com/api/operators/filter.

(b) Verhoog de frequentie dat de `Observable` een update stuurt naar alle `Observers`. We willen echter dat de waardes op de webpagina maar één keer per seconde worden bijgewerkt. Gebruik hiervoor de RxJS operator `throttleTime` https://rxjs-dev.firebaseapp.com/api/operators/throttleTime.