1. Funktional Reaktives Programmieren [U-150]

> *Als Entwickler möchte ich einfache, anonyme Nutzerdaten über Seitenaufrufe erheben um Nutzungsverhalten besser einschätzen zu können.*

1. Lernziele

- ✓ Sie nutzen Observables von Angular um eigene Funktionalität zu erstellen.
- ✓ Sie führen asynchrone Serveranfragen aus einer Observablenkette aus.
- ✓ Sie behandeln Fehler in der Ausführung und sorgen dafür, dass die Analysefunktionalität auch nach einem Fehler noch funktioniert.

2. Ergebnis

Die Todo Applikation sendet URLs und Tageszeiten an eine API, wenn in der Anwendung navigiert wird. Eventuelle Fehler bei der Serveranfrage werden abgefangen, sodass eine kontinuierliche Funktionalität gegeben ist.

3. Benötigte Dateien

- `src/app/app.component.ts`
- `src/app/analytics.service.ts`

4. Anleitung

Schritt 1: `NavigationEnd`-Events aus `router.events` filtern und die geladene URL mit Datum als Objekt bereitstellen

1. In `app.component.ts` erweitern Sie die Liste der Imports aus `@angular/core` um `OnInit`.
2. Fügen Sie `AppComponent` eine Methode `ngOnInit()` hinzu.

3. In `ngOnInit`, subscriben Sie zu der `this.router.events`-Observable des Routers mit der Methode `subscribe()`. Übergeben Sie als `onNext` eine Funktion, die den erhaltenen Wert auf der Konsole loggt. Übergeben Sie als `onError` eine Funktion. die den Wert an `console.error()` weitergibt.
4. Wenn Sie jetzt navigieren, sollten Sie mehrere, verschiedene Events in der Konsole sehen. Da nur Events, weche anzeigen, dass eine Navigation erfolgt ist, von Interesse sind, müssen Sie diese nach Navigationsevents filtern. Importieren Sie `NavigationEnd` von `@angular/router`.
5. Importieren Sie `filter` und `map` aus `rxjs/operators`
6. Erstellen Sie eine Funktion `isNavigationEndEvent` die einen `RouterEvent` erhält und `true` zurückgibt, wenn der Event vom Typ `NavigationEnd` ist. In allen anderen Fällen gibt die Funktion `false` zurück.
7. Vor dem Aufruf von `.subscribe()` fügen Sie eine Funktion `.pipe()` ein, der Sie die verschiedenen Rx Operatoren als Parameter übergeben.
8. Fügen Sie einen `filter()` Operator in `.pipe()` ein, welchem Sie die `isNavigationEndEvent` Funktion übergeben. Sie sollten an dieser Stelle, nur noch ein Event pro Seitenaufruf erhalten. Nun bereiten wir das Event auf, um die Zeit und die URL an den Server zu senden.
9. Fügen Sie nach `filter()` einen weiteren Parameter an, den `map()` Operator, dem Sie eine Funktion übergeben die den Navigationsevent in ein Objekt mit zwei Objektvariablen folgender Typen umwandelt:
    - `url: string`
    - `date: Date`

Schritt 2: Erstellen eines `AnalyticsService` mit einer Methode namens `log()` , welche die Objekte aus dem vorigen Schritt an die `/analytics`-Route des Backends schickt

10. Erstellen Sie eine neue Datei `analytics.service.ts` in `src/app-services`

11. Exportieren Sie eine neue Klasse `AnalyticsService` aus der Datei und fügen den Dekorator `@Injectable` hinzu, mit dem Objekt `{providedIn: 'root'}` als Parameter.
12. Importieren Sie die `HttpClient`-Klasse von `@angular/common/http` und erstellen Sie einen Konstruktor, der den `HttpClient` injiziert bekommt.
13. In `analytics.service.ts`, definieren Sie ein Interface namens `PageNavigationInfo` mit den folgenden Objektvariablen:

    ```
    url: string
    date: Date
    ```
14. Fügen Sie `„analytics“: []` in `db.json` ein
15. Erstellen Sie eine `log()`-Methode in der Klasse, welche einen Parameter vom Typ `PageNavigationInfo` als Parameter übergeben bekommt und ein `Observable` vom Typ `PageNavigationInfo` zurückgibt..
16. Schreiben Sie die Funktionalität für `log()`, die den Parameter vom Typ `PageNavigationInfo` via einer HTTP POST Anfrage an `http://localhost:3000/analytics` sendet. Geben Sie den Rückgabewert des Aufrufs zurück.

Schritt 3: Importieren des `AnalyticsServices` in die `AppComponent` und das Einbinden der `log()`–Funktion, um Daten zum Server zu senden

17. Importieren Sie den soeben erstellten Service in `app.component.ts` und achten darauf, dass der Service injiziert wird. Alle weiteren Schritte werden in `AppComponent` vollzogen.
18. Erstellen Sie eine private Objektvariable namens `routingSubscription` für die `AppComponent` und weisen Sie dieser den Rückgabewert der Rx-Kette in `ngOnInit` zu,
19. Fügen Sie in der `ngOnInit()`-Methode in der `.pipe` nach dem `map` einen `switchMap()`-Operator ein (Import von `switchMap` nicht vergessen), dem Sie eine Funktion übergeben, die den übergebenen Parameter an die `log()`-Funktion des `AnalyticsService` weitergibt.

20. Navigieren Sie in der Applikation und beobachten Sie in der JavaScript-Konsole etwaige Fehlermeldungen. Außer der ersten Nachricht, versucht die Applikation weitere Anfragen an den Server zu senden?

Schritt 4: Bei Fehlern die Observable live halten mit Hilfe des `retry()`-Operators

21. Fügen Sie nach der **`switchMap()`** ein **`retry(2)`** ein. Wie ändert sich das Verhalten? Was passiert, nachdem Sie mehrmals in der Applikation navigiert haben?
22. Fügen Sie temporär nach der **`switchMap`** noch eine **`map()`**-Funktion ein, die eine Exception wirft. Schauen Sie sich das Verhalten der Anwendung nach wiederholtem Navigieren an. Was fällt Ihnen auf?

Schritt 5: Ersetzen des `retry()`-Operators mit `catch()`, welches bei Fehlern die originale Observable zurückgibt.

23. Entfernen Sie die **`map()`** mit der Exception wieder.
24. Mit dem **`retry(2)`** kann die Rx-Kette zwei Fehler akzeptieren, aber nach mehr als zwei Fehlern in Folge wird die Rx-Kette beendet. Wir berichtigen nun dieses Verhalten.
25. Entfernen Sie den **`retry()`**-Operator und korrespondierenden **`import`**.
26. Erweitern Sie die **`AppComponent`** mit einer privaten Variablen **`navEventObservable`** und weisen Sie die Rx-Kette dieser Variablen anstelle von **`routingSubscription`** zu.
27. Ändern Sie den **`.subscribe()`**-Aufruf so, dass er nicht mehr am Ende der Rx-Kette hängt, sondern sich an **`navEventObservable`** subscribed. Sie spalten mit dieser Änderung die Kette auf: Die Bearbeitung des Events wird als Observable an **`navEventObservable`** übergeben. An dieser Variablen erfolgt dann die Subscription.
28. Nach dem **`switchMap()`**-Operator, fügen Sie einen **`catchError()`**-Operator an, welchem Sie eine Funktion übergeben, die den Fehler als Parameter erhält und den Inhalt von **`navEventObservable`** zurückgibt.

29. Navigieren Sie in der Applikation und beobachten Sie die Javascript-Konsole.
30. Fügen Sie dem Objekt in der Datei `db.json` eine Objektvariable `analytics` zu, welche mit einem leeren Array initialisiert wird.
31. Ersetzen Sie die Funktion in `.subscribe()` für den `onNext`-Event mit einer leeren Funktion.
32. Navigieren Sie in der Applikation und beobachten Sie die in der `db.json`-Datei gespeicherten Einträge für Seitennavigation.

5. Kontrollfragen

- Was bewirkt der `switchMap`-Operator, und wie unterscheidet er sich vom `map`-Operator?
- Welche Operatoren gibt es, die nur auf Fehler reagieren? Was sind deren Eigenschaften?

6. Weiterführende Materialien

- RxJs Handbuch
  `http://reactivex.io/rxjs/manual/`
- Rx Marbles (interaktiv)
  `http://rxmarbles.com/`