

Observables

<https://rxjs.dev/guide/overview>

Observables

```
import { Observable } from 'rxjs';
```

Kartais norime savo komponentams nuolat perduoti atnaujinamus duomenis, kad jie galėtų juos rodyti, formatuoti ar tikrinti.

Observables – tai objektai, kurie laikui bėgant perduoda reikšmes savo prenumeratoriams (apie tai vėliau).

Nors jie gali atrodyti kaip funkcijos, nes yra tingūs (neveikia, kol nėra užsiprenumeruojami) ir gražina reikšmes laikui bėgant, iš tikrujų jie yra **RxJS klasės Observable objektai**, turintys metodus kaip `.subscribe()`, `.pipe()` ir `.unsubscribe()`.

Jos leidžia lengvai naudoti **stebėtojo (observer)** šabloną programuojant su **JavaScript**.

Observer pattern in Javascript

- **Subject (Dispatcher)** – tai objektas, kuris perduoda pranešimus sukurtam **Observable** duomenų srautui.
- **Observer (Subscriber/Listener)** gali pradėti gauti pranešimus tik tada, kai jis užsiprenumeruoja šį **Observable**.
- Svarbu: pranešimai tampa matomi tik po prenumeratos inicijavimo – ankstesni duomenys **nėra** automatiškai perduodami naujiems prenumeratoriams (išskyrus tam tikrus subjektų tipus, pvz., **BehaviorSubject** ar **ReplaySubject**).

```
//Subject:    ---M---M---M---M---M  
//Observer:   -----^M---M---M---M
```



Observable subscriptions

Kad **Observable** perduotų reikšmes, **stebėtojas (observer)** turi būti užsiprenumeravęs.

Kol nėra aktyvios prenumeratos, **Observable** lieka neveikiantis ir neduoda jokių duomenų. Tik prenumeravus pradedami siųsti duomenys stebėtojui.

```
import { Observable } from "rxjs";

//Observable
const obs = new Observable((subject) => {
    //Paduodam žinutes Observer
    subject.next(1);
    subject.next(2);
    subject.complete();
});

//Apaščiom Observer
function observer(message) {
    console.log(message);
}

//Prenumeruojam Observable su Observer.
obs.subscribe({
    next: observer,
    error: (err) => console.error(err),
    complete: () => console.log("Observable completed"),
});

//Output: 1, 2, "Observable completed"
```

Subjektai (Subjects)

Multikastuojami (Multicasted) observables – tai **Observable** objektai, kuriuos gali stebėti keli prenumeratoriai vienu metu.

Yra keli **Subject** tipai:

1. **Subject** – paprasčiausias subjektas, perduodantis reikšmes tik prenumeratos metu.
2. **ReplaySubject** – išsaugo ankstesnes reikšmes ir perduoda jas naujiems prenumeratoriams.
3. **BehaviorSubject** – tai **ReplaySubject**, kuris taip pat turi pradinę reikšmę ir ją iškart perduoda užsiprenumeravusiam stebėtojui.

```
import { Subject } from "rxjs";

const subject = new Subject();

const observerOne = (message) =>
  console.log(`Observer1: ${message}`);
const observerTwo = (message) =>
  console.log(`Observer2: ${message}`);

subject.subscribe(observerOne);

subject.next("This");
subject.next("Is");
console.log("Not");

subject.subscribe(observerTwo);

subject.next("Sparta");
/** Output:
 * Observer1: This
 * Observer1: Is
 * Not
 * Observer1: Sparta
 * Observer2: Sparta
 */
```

Observable operatoriai (operators)

Paprastos funkcijos, kurios suteikia papildomų galimybių dirbant su **Observables**.

Yra du pagrindiniai tipai:

1. **Kūrimo (Creation) operatoriai** – sukuria **Observables** iš pateiktų šaltinių arba sujungia kelis **Observables** į vieną.
2. **Vamzdiniai (Pipeable) operatoriai** – naudojami su `.pipe()` funkcija, kad modifikuotų arba transformuotų **Observables** ir grąžintų naujus duomenų srautus.

Pilną operatorių sąrašą galima rasti čia: learnrxjs.io

Operatorių tipai (<https://www.learnrxjs.io/>)

1. Kūrimo (Creation) operatoriai

- **of([1,2,3])** – sukuria **Observable**, kuriame visa masyvo reikšmė yra viena reikšmė.
- **from([1,2,3])** – sukuria **Observable**, kuris perduoda kiekvieną masyvo elementą kaip atskirą reikšmę.
- **fromEvent(document, 'click')** – sukuria **Observable**, kuris reaguoja į click įvykius dokumente.

2. Modifikavimo (Transformation/Filtering) operatoriai

- **filter** – blokuoja tam tikras reikšmes, kad jos nebūtų perduotos prenumeratoriui.
- **map** – keičia perduodamas **Observable** reikšmes prieš jos pasiekia prenumeratorių.

```
import { map, of, tap } from "rxjs";

of(1, 2, 3)
  .pipe(
    tap(console.log),
    map((value) => (value + 10).toString()),
    tap(console.log)
  )
  .subscribe(() => console.log("Success"));
```

Angular interface YRA Observable

<https://angular.dev/api/router/ActivatedRoute>

```
class ActivatedRoute {  
  constructor(urlSubject: BehaviorSubject<UrlSegment[]>, paramsSubject: BehaviorSubject<  
    snapshot: ActivatedRouteSnapshot;  
  readonly title: Observable<string | undefined>;  
  url: Observable<UrlSegment[]>;  
  params: Observable<Params>;  
  queryParams: Observable<Params>;  
  fragment: Observable<string | null>;  
  data: Observable<Data>;  
  override outlet: string;  
  override component: Type<any> | null;  
  readonly routeConfig: Route | null;
```

Observables

AsyncPipe

- Standartinis **Angular pipe**, skirtas prenumeruoti **Observables** (arba Promises) tiesiai template.
- **Automatiškai** atsisako prenumeratos, kai komponentas sunaikinamas.

```
@Component({
  selector: 'app-task',
  imports: [ AsyncPipe ],
  template: '<div>{{ data | async }}</div>',
  styleUrls: ['./task.component.scss'],
})
export class TaskComponent {
  data = new Subject();
  ngOnInit() {
    this.data.next('Hello');
  }
}
```

Observables

Task #1

Papildykite **TaskService Subject** naudojimu.

1. Pakeiskite **TaskService** taip, kad komponento viduje būtų galima rašyti **this.tasks\$ = this.#service.tasks\$** ir jūs visada gautumėte atnaujintą **tasks** sąrašą.
2. Naudokite **Subject** serviso viduje - jo turinj atnaujinkite **getTasks** metodu.
3. Naudokite **switchMap** pipe'able operatorių, kad pakviestumėte **getTasks** po naujo task pridėjimo servise.
4. (**Optional**): Sukūrė naują servisą, kuris ima duomenis iš <https://randomuser.com/api> - naudokite **combineLatest** ir pridėkite "užduoties kūrėja" prie kiekvienos užduoties.

Lifto Muzika.

