ANGULAR FRONTEND FEJLESZTÉS

7. forduló

A kategória támogatója: TCS - TATA Consultancy Services

RENDELKEZÉSRE ÁLLÓ IDŐ:

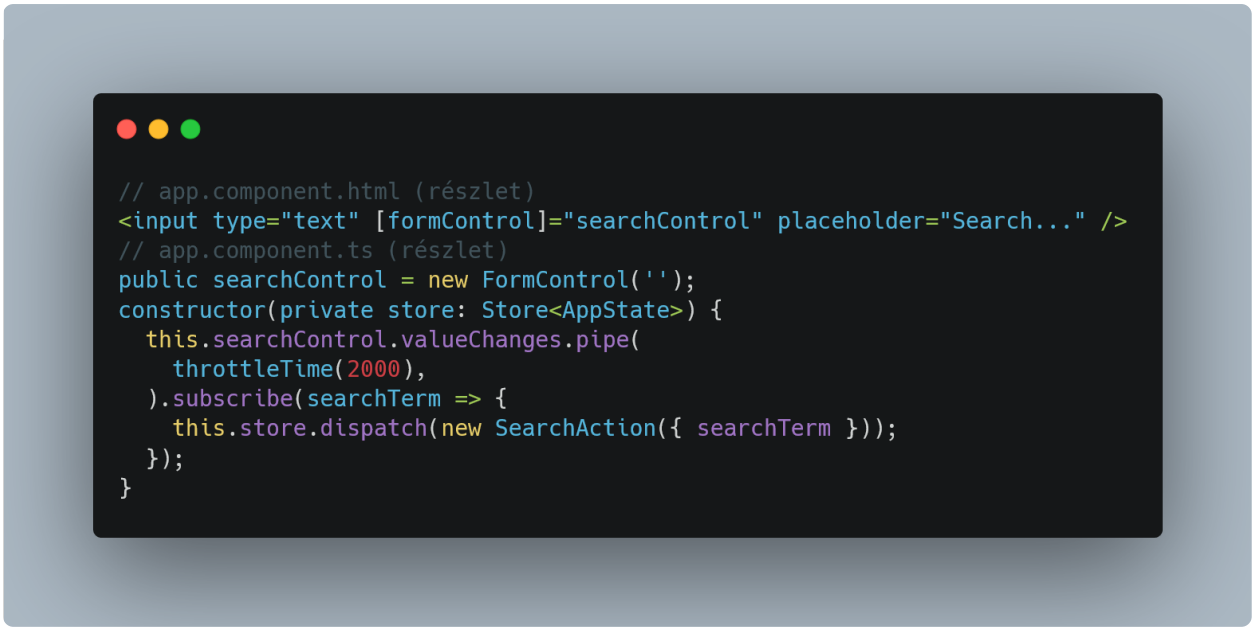
20:00

Ismertető a feladathoz

Felhasznált idő: 00:57/20:00 **Elért pontszám:** 0/20

1. feladat 0/4 pont

Implementáltunk egy nagyon kényelmes keresés funkciót. Egyetlen beviteli mező van, amibe a felhasználó beírja a keresési sztringet, és nem kell sehová kattintania, a találatok hamarosan automatikusan megjelennek. A keresés nem frontend oldalon zajlik, ezért nem szerettük volna, hogy minden egyes billentyűleütés után kimenjen a kérés a szerverhez. Csak akkor kellene kimennie, ha megváltozik a keresési sztring, majd utána két másodpercig változatlan marad.



```
// app.component.html (részlet)
<input type="text" [formControl]="searchControl" placeholder="Search..." />
// app.component.ts (részlet)
public searchControl = new FormControl('');
constructor(private store: Store<AppState>) {
  this.searchControl.valueChanges.pipe(
    throttleTime(2000),
  ).subscribe(searchTerm => {
    this.store.dispatch(new SearchAction({ searchTerm }));
  });
}
```

Sajnos azt tapasztaltuk, hogy nem működik megfelelően, a **throttleTime(2000)** problémás. Mire cseréljük ki?

Válasz

- ○ throttleTime(2000, asyncScheduler, { leading: false, trailing: true })
- ○ throttleTime(2000, asyncScheduler, { leading: false, trailing: false })
- ◯ debounceTime(2000)
- ○ auditTime(2000)
- ○ ignoreElements(2000)
- ○ skipUntil(timer(2000))

Magyarázat

A **debounceTime** pont azt csinálja, amire szükségünk van, a **throttleTime** és **auditTime** nem fognak várni 2sec-et, csak késleltetik az emittálásokat.

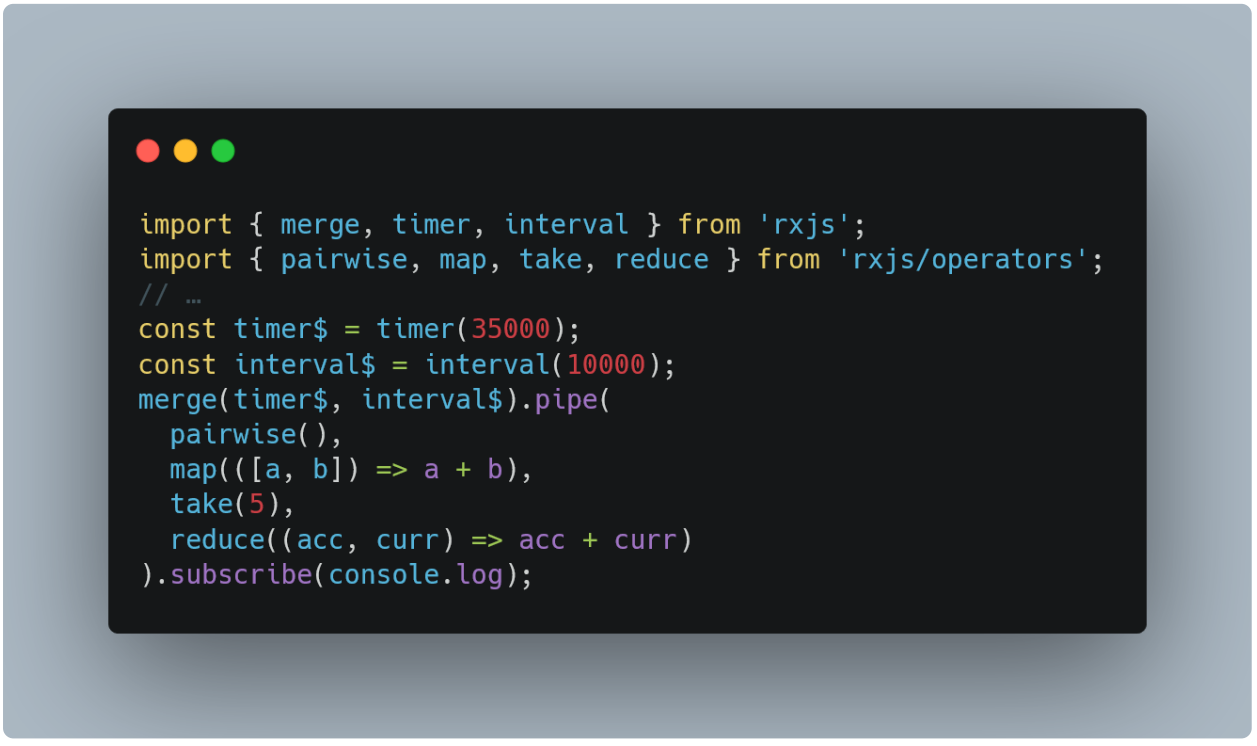
Összefoglaló cikk erről a háromról: https://medium.com/@jvdheijden/rxjs-throttletime-debouncetime-and-audittime-explained-in-examples-c393178458f3

Az **ignoreElements**-nek nincs is paramétere, mindent ignorál.

A **skipUntil** addig skippeli az értékeket, amíg a paraméterül kapott observable nem emittál, tehát 2sec-ig skippel mindent, de utána mindent átenged.

2. feladat 0/6 pont

Mi fog a konzolra íródni az alábbi kód lefuttatása után? (Természetesen minden timert megvárunk)



```
import { merge, timer, interval } from 'rxjs';
import { pairwise, map, take, reduce } from 'rxjs/operators';
// …
const timer$ = timer(35000);
const interval$ = interval(10000);
merge(timer$, interval$).pipe(
  pairwise(),
  map(([a, b]) => a + b),
  take(5),
  reduce((acc, curr) => acc + curr)
).subscribe(console.log);
```

Válasz

- ○ Error
- ○ NaN
- ○ 36
- ◯ 16
- ○ 10
- ○ 9
- ○ 3
- ○ 6

Magyarázat

A timer 0 értéket emittál 35sec múlva, majd complete-el.

Az interval 10sec-enként emittál 0tól kezdődően növekvő egész számokat.

A mergelt observable tehát az alábbi számokat emittálja sorrendben: 0, 1, 2, 0, 3, 4, 5, 6, 7, 8...

A pairwise kettessével emittálja őket tuple-ként: [0,1], [1,2], [2,0], [0,3], [3,4], [4,5], [5,6], [6,7], [7,8]...

A map összeadja a tuple-öket: 1, 3, 2, 3, 7, 9, 11, 13, 15....

A take(5) miatt csak az első öt számot vesszük.

A reduce összeadja őket: 1+3+2+3+7 = 16

3. feladat 0/10 pont

Adott az alábbi Angular modul és komponens:

```
// app.module.ts
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  bootstrap: [AppComponent]
})
export class AppModule {}
// app.component.ts
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  changeDetection: ChangeDetectionStrategy.OnPush,
  providers: [HttpClientModule]
})
export class AppComponent {
  public terrans$: Observable<Terran[]>;
  public showTerranList = false;
  constructor(private http: HttpClient) {
    this.terrans$ = this.http.get<Terran[]>(ApiEndpoint.GetTerrans);
  }
}
// app.component.html
<button (click)="showTerrans = !showTerrans">Show / Hide Terrans</button>
<div *ngIf="showTerrans">
  <div *ngFor="let terran of terrans$ | async">
    Name: {{ terran.name }} | Power: {{ terran.power }}
  </div>
</div>
```

(Az egyszerűség kedvéért a TS import-okat kihagytuk, de minden megfelelően importálva van, lefordul, de futás közben nem feltétlenül error mentes.)

Az alábbi állítások közül melyek igazak?

Válaszok

- [x] Errort kapunk, mert a **HttpClientModule**-t a modul imports tömbjébe kell tenni, nem a komponens **providers**-be
- [ ] Errort kapunk, mert a komponensből kihagytuk a **styleUrls** tömböt
- [ ] Errot kapunk az **async pipe**, ***ngIf**, ***ngFor**-ra, mert a modulunkba nincs importálva a **CommonModule**
- [ ] Errot kapunk a gombra kattintáskor; egy függvényt kellene hívnunk, nem állíthatjuk a **showTerrans** értékét szimplán a template-ből
- [ ] (Feltételezzük, hogy nincs már futás közben error) A http GET kérés kimegy akkor is, ha még nem kattintottunk a gombra egyszer sem
- [x] (Feltételezzük, hogy nincs már futás közben error) Ha sokszor kattintunk a gombra a http GET kérés kimegy minden alkalommal, amikor a **showTerrans** true-ra vált
- [ ] (Feltételezzük, hogy nincs már futás közben error) Az **OnPush ChangeDetection** miatt a Terranok nem jelennek meg a UI-on, a click handlerbe kell egy **markForCheck()/detectChanges()**

Magyarázat

a) A HttpClientModule nem Injectable hanem module, ezért importálni kell.

b) Nem kötelező a styleUrls-t megadni.

c) Ezek ugyan a CommonModule-ban vannak, de exportálja őket a BrowserModule is, ezért nem szükséges a CommonModule importálása a használatukhoz.

d) Lehet template-be expressionöket tenni, nem feltétlenül kell függvénybe szervezni ennyire egyszerű logikát.

e) A http.get egy Cold Observable-t ad vissza, az első subscriber-ig nem fog kimenni a kérés. Mivel az *nglf kezdeti értéke false azért az async pipe nem subscribe-ol rá amíg nem kattintunk.

f) A http.get minden új subscriber esetén újra kiküldi a kérést. Mivel a gombra való kattintás újra subscribe/unsubscribe-ol így a kérés többször fog kimenni. Ezt kiküszöbölhetjük pl. egy shareReplay() operarátorral.

g) Nem igaz, OnPush esetén is le fog futni a changeDetector, mert async pipe-ot használtunk. A pipe minden emittálás után magától hív egy markForCheck()-et.