Angular 16+ programozás

Masterfield TRAINING
IT COURSES FOR PROFESSIONALS

Mohó vs. lusta betöltés

- Alapértelmezés szerint az NgModules betöltése mohó módon történik.
- A sok útvonallal rendelkező nagy alkalmazásoknál fontolja meg a lusta betöltést – egy olyan tervezési mintát, amely szükség szerint betölti az NgModules-t.
- A késleltetett betöltés segít a kezdeti kötegméretek kisebb tartásában, ami viszont csökkenti a betöltési időt.

Modulok betöltése

- Alapértelmezés szerint a modulok betöltése során az Angular mohó betöltést alkalmaz.
- Komplex, sok útvonallal rendelkező alkalmazások esetén a mohó betöltés a teljesítményt ronthatja.

Az útvonalleírások elején betöltjük a modult


```
import { EmpListComponent } from './emp/emp-list/emp-list.component';

const routes: Routes = [
  {
    path: 'emp',
    component: EmpListComponent
  }
];
```

Lusta betöltés

- Az Angularban van lehetőségünk lusta betöltés használtára, csak akkor töltjük be a modult, ha arra szükség van a futás során.

Az útvonalleírások elején betöltjük a modult


```
//import { EmpListComponent } from './emp/emp-list/emp-list.component';

const routes: Routes = [
  {
    path: 'emp',
    //component: EmpListComponent
    loadChildren: () => import('./emp/emp.module')
                        .then(m => m.EmpModule)
  }
];
```

Signal

- Az Angular v17-es verziójában mutatták be.
- Reaktív primitívek készíthetőek a komponensek kezelése során.
  - értéket képvisel
  - szabályozott módon változtatható meg
  - a változás nyomon követhető, detektálható
- Megkülönböztetünk írható és olvasható signalokat.
- Célja a változásészlelés és a tartalomgenerálás optimalizálása.

Signal használata

- Egy-egy esemény bekövetkezése esetén
  - Alapértelmezett változásészlelés esetén nem tudjuk pontosan, hogy mi változott ➔ a teljes komponensfát, minden kifejezésével együtt át kell vizsgálni.
  - Szignálok használata esetén pontosan megállapítható, hogy melyik adat változott, hiszen értesítést kapunk ➔ megállapítható, hogy mely komponenseket, és azokon belül mely kifejezéseket kell megváltoztatni.

Írható signal használata

```
export class CounterComponent {
   counter = signal(0);

   inc(){this.counter.update(actValue => actValue + 1)}
   dec(){ this.counter.update(actValue => actValue - 1)}

   reset(){ this.counter.set(0);}
}
```

Incializáláshoz a **signal()** függvényt kell meghívni.

Közvetlen értékbeállítás a **set()** függvényen keresztül történik.

Ha a signal aktuális értékét fel szeretnénk használni a beállításhoz, akkor az **update()** függvényt kell meghívni


```
Érték: {{counter()}}
<button (click)="dec()">--</button>
<button (click)="reset()">Reset</button>
<button (click)="inc()">++</button>
```

Érték kiolvasáshoz a szignált függvényként hívjuk meg.

Származtatott értékek

- computed(): olyan signal, amely egy vagy több más signal értékéből kerül kiszámításra. ➔ Ha a forrás signal frissül, a számított érték is.
- A számított érték gyorsítótárba kerül és lusta módon kerül kiértékelésre.
- Csak olvasható.

A Farenheit értéket a Celsius érték alapján számítjuk ki.

```
export class TemperatureComponent {
  celsius = signal(10.3);
  farenheit = computed(() => this.celsius() * 1.8 + 32);

  inc(){this.celsius.update(actValue => actValue + 0.1)}
  dec(){ this.celsius.update(actValue => actValue - 0.1)}
  reset(){ this.celsius.set(0);}
}
```

Computed értékek nyilvántartása

- Nincs szükség megfigyelő használatára, nincs feliratkozás.
  1. A származtatott érték meghatározása legalább egyszer megtörténik, az inicializálás során.
  2. A származtatott érték számítása során az Angular felderíti, hogy mely forrásszingáloktól függ az érték.
- A számított szignálok esetén a forrásszignálok kezelés dinamikus. Ha a számítás során nem hívjuk meg a forrásszignált, akkor megszűnik a függőség a szignálok között!

Computed értékek

- untracked(): számított szignál a forrás szignál értékének kiolvasása függőséglétrehozás nélkül.
- Ritkán, csak indokolt esetben használjuk!

```
export class FilteredListComponent {
  items = signal([...]);
  nameFilter = signal('');
  categoryFilter = signal('computers');

  filteredItems = computed(() => {
    const nameF = this.nameFilter();
    const catF = untracked(this.categoryFilter);
    return this.items().filter(item =>
      item.name.includes(nameF) && item.category == catF
    );
  });
}
```

A kategória szűrőt csak akkor vesszük figyelembe, ha más signal is változik.

Összetett adatszerkezetek

- A signal értéke lehet összetett adatszerkezet (objektum, tömb) is, ekkor is a korábban bemutatott függvényeket kell használni.

```
console.log(this.user().id)
console.log(this.user().name)
this.user.set({id:12, name:"John Smith"})
```

- Az alapértelmezett egyenlőségvizsgálatot indokolt esetben felül lehet írni!

```
user = signal(
  {id: 1, name: "Tom Jones"},
  {equal: (a, b) => { return a.id === b.id ;},}
);
```

Mellékhatások

- Az effektusok segítségével mellékhatásokat tudunk rendelni a signalban történő változáshoz.
- Aszinkron módon, a változásészlelési folyamat során futnak le.
- A signaltól való függés az utolsó lefutástól függ.
- Az effect() függvénnyel hozhatjuk létre injektálásra alkalmas környezetben.
- Ne használjuk az állapotváltozások továbbítására!

```
effect(() => {
  console.log(`Actual counter value: ${this.counter()}`);
});
```

Callback-ként adjuk át a mellékhatást.

Minden counter értékváltozásra lefut

Mellékhatások használata

- Alapértelmezetten, ha a mellékhatáson belül a signal értékeket meg akarjuk változtatni, akkor hibát kapunk.
- Ha erre mégis szükség van, akkor azt külön engedélyezni kell a mellékhatás létrehozásakor.

```
effect(() => {
  this.count.set(1);
},{allowSignalWrites: true});
```


- Csak kivételes esetekben használjuk!

Állapotkezelés

- A webfejlesztés egyik legfontosabb kihívása az alkalmazás állapotának hatékony kezelése.
- Az állapotkezelés határozza meg, hogy az adatok tárolása, lekérése és frissítése az alkalmazás különböző összetevői között hogyan történik.
- Több állapotkezelési lehetőség áll rendelkezésre.
  - Komponens állapot
  - Szolgáltatások és RxJS
  - NgRx tárolók
- A signal koncepció megjelenésével az állapotok kezelése újabb alternatívát kapott.