

# JEGYZŐKÖNYV

Web-Technológiák II.

Féléves feladat

Raktárnyilvántartó rendszer

Készítette: Varga Bence

Neptunkód: CKFEC9

A modern üzleti világban a hatékony készletgazdálkodás kulcsfontosságú a vállalatok számára, hogy versenyképesek maradjanak és ügyfeleik igényeit kielégítsék. E cél elérése érdekében kifejlesztettem egy raktárnyilvántartó rendszert Angular programnyelven, amely nemcsak a készletkezelést teszi egyszerűbbé és átláthatóbbá, hanem hozzájárul a vállalat működésének optimalizálásához is.

Az Angular választása több szempontból is előnyös volt a projekt számára. Az Angular egy erőteljes és rugalmas front-end keretrendszer, amely lehetővé teszi a fejlesztők számára, hogy moduláris és karbantartható webalkalmazásokat hozzanak létre. A keretrendszer használata során a következő előnyöket tapasztaltam:

- **Modularitás és Újrafelhasználhatóság:** Az Angular moduláris felépítése lehetővé tette számomra, hogy különálló komponenseket hozzak létre a rendszer különböző funkcióihoz. Például külön komponensek készültek a készlet listázására, a részletek megjelenítésére és az új áruk bevitelére. Ez a struktúra nemcsak a kód újrafelhasználhatóságát tette lehetővé, hanem jelentősen megkönnyítette a fejlesztést és a karbantartást is.
- **Szolgáltatások és Üzleti Logika:** Az Angular szolgáltatásai lehetővé tették az üzleti logika elkülönítését a felhasználói felülettől. Az összes készletkezelési műveletet egy külön szolgáltatás kezelte, amely biztosította az adatok konzisztenciáját és a kód átláthatóságát.
- **Reszponzív Felhasználói Felület:** Az Angular támogatja a rezponzív webdizájnt, ami biztosította, hogy az alkalmazás különböző eszközökön, például asztali számítógépeken, táblagépeken és okostelefonokon egyaránt jól működjön.

A projekt során a következő technológiákat és eszközöket használtam:

- **TypeScript:** Az Angular TypeScript alapú, ami szigorú típusellenőrzést és modern JavaScript funkciókat kínál. Ez lehetővé tette a hibák korai észlelését és a kód minőségének javítását.
- **Angular CLI:** Az Angular CLI egy parancssori eszköz, amely segítette az alkalmazás gyors és hatékony fejlesztését. Az eszköz támogatja a projekt

létrehozását, a modulok és komponensek generálását, valamint az alkalmazás építését és tesztelését.

- **HttpClient:** Az Angular HttpClient modulját használtam a szerverrel való kommunikációhoz REST API-kon keresztül. Ez lehetővé tette az adatok hatékony és biztonságos továbbítását a szerver és a kliens között.
- **Reactive Forms:** Az Angular Reactive Forms modulja erős támogatást nyújtott a formok kezeléséhez és validációjához, ami különösen fontos volt az adatok bevitele és kezelése során.

# Fő oldál

## Raktárnyilvántartó rendszer

Raktár Gyártmány Megrendelések

Raktári egységek

Friestítés

Egyreőg hozzáadása

Alkatrész neve: Motor

Mennyiség: 1db

Raktári szám: #1

Bevételezés ideje: 2024-05-27

Törles

The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'src' directory containing various files and folders. The file 'warehouse.component.ts' is selected. The code editor displays the content of this file, which is a TypeScript class 'WarehouseComponent' implementing 'OnInit'. The class has a 'warehouseItems' array, a 'constructor' that takes 'WarehouseService' and 'ToastrService' as arguments, and a 'loadWarehouseItems()' method that fetches data from 'warehouseService' and displays it using 'toastrService'. The terminal at the bottom shows the output of the Angular CLI, indicating that the application bundle generation is complete and the application is running on localhost:4200.

```

src > app > warehouse > ts warehouse.component.ts > WarehouseComponent
1  import { Component, OnInit } from '@angular/core';
2  import { WarehouseItem } from '../models/warehouse-item';
3  import { WarehouseService } from '../services/warehouse.service';
4  import { ToastrService } from 'ngx-toastr';
5
6  @Component({
7    selector: 'app-warehouse',
8    templateUrl: './warehouse.component.html',
9    styleUrls: ['./warehouse.component.css']
10 })
11 export class WarehouseComponent implements OnInit {
12   warehouseItems: WarehouseItem[] = [];
13
14   constructor(private warehouseService: WarehouseService, private toastrService: ToastrService,) {}
15
16   ngOnInit(): void {
17     this.loadWarehouseItems();
18   }
19
20   loadWarehouseItems(): void {
21     this.warehouseService.getWarehouseItems().subscribe(items => {
22       this.warehouseItems = items.filter(item => item.quantity > 0);
23
24       // Törölés azokból az elemekből, amelyeknek a darabszáma 0
25       const deleteItems = items.filter(item => item.quantity === 0);
26       deleteItems.forEach(item => {
27         this.warehouseService.deleteWarehouseItem(item.id).subscribe(() => {
28           console.log('0-as darabszám miatt a(z) ' + item.name + ' raktári egység törölve lett!');
29         }, error => {
30           console.error('Hiba történt az elem törlésekor:', error);
31         });
32       });
33     });
34   }
35 }

```

PROBLEMS OUTPUT **TERMINAL** PORTS DEBUG CONSOLE

```

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.
✓ Browser application bundle generation complete.

5 unchanged chunks

Build at: 2024-05-27T16:33:42.104Z - Hash: d803f004e8cea9e1 - Time: 729ms

✓ Compiled successfully.

```

Az általam fejlesztett raktárnyilvántartó rendszer egyik kulcsfontosságú komponense az Angular alapú WarehouseComponent, amely a raktári elemek kezelését és megjelenítését végzi. Ez a komponens az ngOnInit életciklus metódusban hívja meg a loadWarehouseItems metódust, amely lekéri a raktári elemeket a WarehouseService segítségével, és szűri azokat, amelyek készlete meghaladja a nullát. Azokat az elemeket, amelyek készlete nulla, automatikusan törli a rendszer. Az új elemek hozzáadása és a meglévők törlése szintén ezen a komponensen keresztül történik, biztosítva a felhasználók számára a raktári készlet hatékony és valós idejű kezelését. A ToastrService használatával a rendszer értesítéseket küld a felhasználónak a sikeres műveletekről, így javítva a felhasználói élményt és a rendszer átláthatóságát.

## Gyártmány

### Raktárnyilvántartó rendszer

[Raktár](#) [Gyártmány](#) [Megrendelések](#)

Gyártmány felvétele

Gyártmány neve		
Motorháorgár		
Darabszám		
		1 db
<button>Mentés</button>		

Autóhoz szükséges alkatrészek		
Motor	Váz	Kerék
1 db	1 db	4 db

Motorháorgárhoz szükséges alkatrészek		
Motor	Váz	Kerék
1 db	1 db	2 db

Kamionhoz szükséges alkatrészek		
Motor	Váz	Kerék
1 db	2 db	6 db

Biciklihez szükséges alkatrészek		
Váz	Kerék	Csavarhészlet
1 db	2 db	1 db

Csavarhészlethez szükséges alkatrészek		
M2-es csavar	M5-os csavar	M12-es csavar
10 db	10 db	10 db

```

src > app > manufacturing-list > TS manufacturing-list.component.ts > ManufacturingListComponent
19
20 export class ManufacturingListComponent implements OnInit {
21
22
23
24   manufacturingForm = this.formBuilder.group({
25
26     id: this.formBuilder.control(''),
27     manufacturingName: this.formBuilder.control('', Validators.required),
28     quantity: this.formBuilder.control(1)
29   });
30
31
32
33   constructor(
34     private manufacturingService: ManufacturingService,
35     private warehouseService: WarehouseService,
36     private toastrService: ToastrService,
37     private activatedRoute: ActivatedRoute,
38     private formBuilder: FormBuilder) { }
39
40
41   ngOnInit(): void {
42     const id = this.activatedRoute.snapshot.params['id'];
43
44     if (id) {
45
46       this.manufacturingService.getOne(id).subscribe({
47
48         next: (item) => {
49           this.manufacturingForm.setValue(item);
50         },
51

```

Az `ManufacturingListComponent` az `OnInit` életciklus metódust használja a komponens inicializálásához. A komponens az Angular `FormBuilder` szolgáltatását használja egy reaktív űrlap létrehozásához, amely tartalmazza a gyártási tételek adatait, mint például az azonosítót, a gyártási nevet és a mennyiséget. Az űrlap validációs szabályokat is tartalmaz, például a gyártási név kötelező mezőként van beállítva.

## Adatok Betöltése és Hibakezelés

Az `ngOnInit` metódusban az `ActivatedRoute` segítségével lekérjük a URL-ből az azonosítót, ha az rendelkezésre áll. Ha az azonosító megtalálható, a komponens az `ManufacturingService` szolgáltatáson keresztül lekéri a megfelelő gyártási tétel adatait. Az adatok sikeres betöltése esetén a tétel adatai betöltődnek az űrlapba a `setValue` metódus segítségével. Ha hiba lép fel az adatok betöltésekor, a komponens a `ToastrService` használatával értesíti a felhasználót a hibáról.

A komponens több szolgáltatást is integrál a működése során:

- `ManufacturingService`: A gyártási tételek adatainak kezelésére szolgál. Lehetővé teszi a tételek lekérését és szerkesztését.
- `WarehouseService`: A raktári tételek kezeléséhez használható szolgáltatás.

- ToastrService: A felhasználói értesítések megjelenítésére szolgál. Sikeres és hibás műveletek esetén is értesítést küld a felhasználónak.
- ActivatedRoute: Az aktuális útvonal paramétereinek kezelésére szolgál, lehetővé téve az azonosítók dinamikus lekérését.

### Felhasználói Élmény és Hibakezelés

A komponens célja, hogy egy könnyen használható és megbízható felületet biztosítson a felhasználók számára a gyártási tételek kezeléséhez. A ToastrService használatával a rendszer valós idejű visszajelzést ad a felhasználóknak a műveletek eredményéről, ami jelentősen javítja a felhasználói élményt. A hibakezelés révén a felhasználók azonnal értesülnek a problémákról, és ezáltal gyorsan tudnak intézkedni.

## Raktárnyilvántartó rendszer

Raktár Gyártmány Megrendelések

### Rendelésfelvevél

Megrendelő neve	<input type="text"/>
Születési idő	<input type="text" value="éééé. hh. nn. --&gt;"/>
Gyártmány neve	<input type="text"/>
Válassz gyártmányt!	<input type="text"/>
Darabszám	<input type="text" value="1"/> db
<input type="button" value="Mentés"/>	

```
14 styleUrls: ['./order-list.component.css']
15 })
16
17
18
19 export class OrderListComponent implements OnInit {
20   orderForm = this.formBuilder.group({
21
22     id: this.formBuilder.control(''),
23     customerName: this.formBuilder.control('', Validators.required),
24     dateOfBirth: this.formBuilder.control(new Date()),
25     order: this.formBuilder.control('', Validators.required),
26     quantity: this.formBuilder.control(1)
27   });
28
29
30
31   constructor(
32     private orderService: OrderService,
33     private toastrService: ToastrService,
34     private activatedRoute: ActivatedRoute,
35     private formBuilder: FormBuilder) { }
36
37
38   ngOnInit(): void {
39     const id = this.activatedRoute.snapshot.params['id'];
40
41     if (id) {
42       this.orderService.getOne(id).subscribe({
43
44         next: (item) => {
45           this.orderForm.setValue(item);
46         },
47
48         error: (err) => {
49
50           console.error(err);
51           this.toastrService.error('Hiba a termékadatok betöltésekor.', 'Hiba');
52         }
53       });
54     }
55   }
56
57 }
```

Az OrderListComponent reaktív űrlapot használ az Angular FormBuilder segítségével, amely lehetővé teszi a felhasználók számára a rendelési tételek adatainak bevitelét. Az űrlap különböző mezőket tartalmaz, mint például az azonosítót, a vevő nevét, a születési dátumot, a rendelés leírását és a mennyiséget. A vevő neve és a rendelés leírása kötelező mezőként van megjelölve a Validators.required használatával.

### Adatok Betöltése és Hibakezelés

Az ngOnInit metódusban a komponens az ActivatedRoute segítségével lekéri a URL-ből az azonosítót, ha az rendelkezésre áll. Ha az azonosító megtalálható, az OrderService szolgáltatáson keresztül lekéri a megfelelő rendelési tétel adatait, és azokat betölti az űrlapba a setValue metódus segítségével. Ha hiba lép fel az

adatok betöltésekor, a komponens a ToastrService használatával értesíti a felhasználót a hibáról.

### Rendelési Tételek Mentése

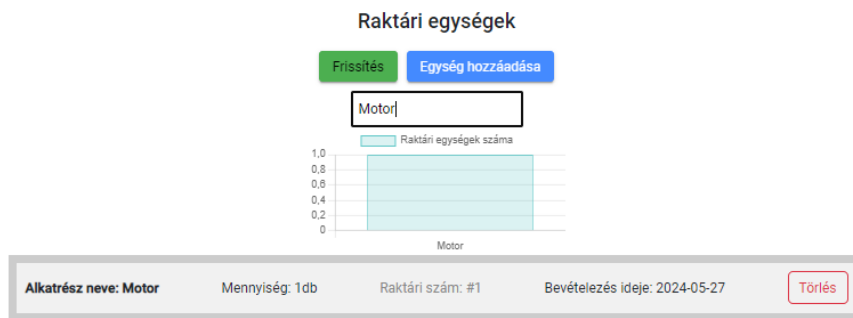
Az `saveOrder` metódus kezeli az új rendelési tételek mentését és a meglévő tételek frissítését. Ha az űrlap érvényes, a metódus először ellenőrzi, hogy létezik-e már rendelés az adott vevő névvel. Ezt a `OrderService` szolgáltatás `getOrders` metódusával teszi. Ha létezik már ilyen rendelés, a mennyiség frissítésre kerül, és az adatbázisban is frissül az elem. Ha nem létezik ilyen rendelés, az új rendelési tétel hozzáadásra kerül az adatbázishoz.





```
updateChartData(): void {  
  const labels = this.warehouseItems.map(item => item.name);  
  const quantities = this.warehouseItems.map(item => item.quantity);  
  
  this.chartData = {  
    labels: labels,  
    datasets: [  
      {  
        label: 'Raktári egységek száma',  
        data: quantities,  
        backgroundColor: 'rgba(75, 192, 192, 0.2)',  
        borderColor: 'rgba(75, 192, 192, 1)',  
        borderWidth: 1  
      }  
    ]  
  };  
}
```

Kiegészítettem egy diagramrendszerrel a beadandómat, amely mutatja a jelenleg elérhető raktári egységek darabszámát.



```
addWarehouseItem(newItem: WarehouseItem): void {
  this.warehouseService.create(newItem).subscribe(() => {
    this.loadWarehouseItems();
  }, error => {
    console.error('Hiba történt a raktári elem hozzáadásakor:', error);
  });
}

deleteWarehouseItem(itemId: number): void {
  this.warehouseService.deleteWarehouseItem(itemId).subscribe(() => {
    this.loadWarehouseItems();
    this.toastrService.success('Raktári egység sikeresen törölve!', 'Siker');
  }, error => {
    console.error('Hiba történt a raktári elem törlésekor:', error);
  });
}

searchWarehouseItems(): void {
  if (this.searchTerm.trim()) {
    this.filteredWarehouseItems = this.warehouseItems.filter(item =>
      item.name.toLowerCase().includes(this.searchTerm.toLowerCase())
    );
  } else {
    this.filteredWarehouseItems = [...this.warehouseItems];
  }
  this.updateChartData();
}
```

Hozzáadtam egy keresősávot a főoldalhoz, ahol különböző alkatrészekre tudunk keresni.