

DOKUMENTÁCIÓ

Web technológiák 2

Készítette: **Nyíri Beáta**

Neptunkód: **I40FDC**

Dátum: **2023.06.29.**

Tartalom

A feladat leírása	1
Backend kezdeti beállításai, adatbázis létrehozása	1
Felhasználók kezelése (Backend)	2
Regisztráció	3
Bejelentkezés.....	3
Jelszó megváltoztatása	4
Állatok és kategorizálásuk (Backend)	5
DLC.....	5
Állatok.....	6
Felhasználók kezelése (Frontend)	7
Regisztráció	7
Bejelentkezés.....	8
Jelszó megváltoztatása	9
Kijelentkezés.....	9
Menü	10
Dashboard	11
DLC-k oldala	11
Állatok oldala	13

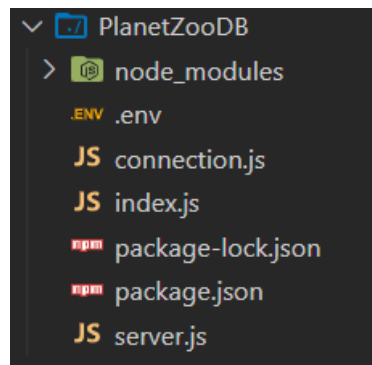
A feladat leírása

A beadandóm témájaként ugyanazt választottam, amit a Web technológiák 1 beadandóhoz is, ez pedig az egyik kedvelt videójátékom, a Planet Zoo. A feladathoz a játékban szereplő állatokhoz készítettem egy nyílvántartó rendszert.

Ehhez a kiadott feladat leírásában feltüntetett technológiákat alkalmaztam (Angular, NodeJS) az adatbáziskezelőn kívül. Én ugyanis MongoDB helyett MySQL-t használtam. A programot Visual Studio Code-ban írtam.

Backend kezdeti beállításai, adatbázis létrehozása

A programot a backend megírásával kezdtem. Feltelepítettem a szükséges csomagokat, létrehoztam egy adatbázist, majd létrehoztam a kapcsolatot az adatbázissal.



Az .env fájlban a globális változókat tárolom, connection.js fájl segítségével pedig az adatbázishoz csatlakoztam, amit MySQL 8.0 Command Line Client-tel kezeltem.

```
PlanetZooDB > .ENV .env
1 //Server
2 PORT = 8080
3
4 //Connection
5 DB_PORT = 3306
6 DB_HOST =localhost
7 DB_USERNAME = root
8 DB_PASSWORD = 1234
9 DB_NAME = planetzoodb
```

```
PlanetZooDB > JS connection.js > ...
1 const mysql = require('mysql');
2 require('dotenv').config();
3
4 var connection = mysql.createConnection({
5   port: process.env.DB_PORT,
6   host: process.env.DB_HOST,
7   user: process.env.DB_USERNAME,
8   password: process.env.DB_PASSWORD,
9   database: process.env.DB_NAME
10 });
11
12 connection.connect((err) =>{
13   if(!err){
14     console.log("Connected");
15   }
16   else{
17     console.log(err);
18   }
19 });
20
21 module.exports = connection;
```

Azért pedig, hogy ne manuálisan kelljen frissítenem a servert minden változtatásnál, nodemon-t használtam. Ehhez a package.json fájlban hoztam létre egy új "start" scriptet.

```
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1",
8     "start": "nodemon server.js"
9   },
```

Létrehoztam egy index.js fájlt is, ami a projekt haladásával folyamatosan bővült.

Felhasználók kezelése (Backend)

Az oldalt úgy építettem fel, hogy kétféle bejelentkezési lehetőség legyen. Be lehet lépni adminként és egyszerű felhasználóként. A különbség az, hogy az admin minden menüpontot lát és elér, a felhasználók pedig nem mindet, valamint ha egy felhasználó be szeretne lépni (Sign In), akkor azt az adminnak jóvá kell hagynia.

Azzal kezdtem ezt a fázist, hogy létrehoztam egy table.sql fájlt, amiben a user tábla létrehozásához és kezeléséhez szükséges SQL parancsok vannak, és feltöltöttem pár teszt adatot.

id	name	email	password	status	role
1	Admin	admin@gmail.com	admin	true	admin
2	Bea Nyiri	beanyiri@gmail.com	1234	false	user

Regisztráció

Bevezettem a route-okat és elsőként a regisztrációhoz készítettem egy API-t. A user.js fájlban az ehhez szükséges kód van. Az email alapján ellenőrzöm a felhasználókat, ha egy email már szerepel az adatbázisban, akkor ezt jelzi a program egy üzenettel, ha pedig még nem szerepel és jó adatokat adott meg a regisztrálni kívánó felhasználó, akkor az adatai bekerülnek az adatbázisba és egy üzenet jelenik meg, miszerint a regisztráció sikeres volt.

```

PlanetZooDB > routes > JS user.js > ...
1  const express = require('express');
2  const connection = require('../connection');
3  const router = express.Router();
4
5  router.post('/signup', (req, res) =>{
6      let user = req.body;
7      query = "select email,password,role,status from user where email=?"
8      connection.query(query,[user.email],(err,results) =>{
9          if(!err){
10             if(results.length <=0){
11                 query = "insert into user(name,email,password,status,role) values(?,?,?, 'false', 'user')";
12                 connection.query(query,[user.name,user.email,user.password],(err,results) =>{
13                     if(!err){
14                         return res.status(200).json({message: "Successfully Registered"});
15                     }
16                     else{
17                         return res.status(500).json(err);
18                     }
19                 })
20             }
21             else{
22                 return res.status(400).json({message: "Email Already Exists."});
23             }
24         }
25         else{
26             return res.status(500).json(err);
27         }
28     })
29 })
30
31
32 module.exports = router;

```

Bejelentkezés

A user.js fájlba ezután a bejelentkezéshez szükséges részt írtam meg. Bejelentkezéskor az emailt és a jelszót kell megadni és azt vizsgálom, hogy az email alapján már van-e ilyen felhasználó, jól lettek-e beírva az adatok (nincs elgépelés), illetve, hogy ki szeretne bejelentkezni (sima felhasználó, vagy admin). Sikeres bejelentkezés esetén egy belépési token generálódik.

```
//Bejelentkezés
router.post('/login',(req,res) =>{
  const user = req.body;
  query = "select email,password,role,status from user where email=?";
  connection.query(query,[user.email] ,(err,results) =>{
    if(!err){
      if(results.length <=0 || results[0].password !=user.password){
        return res.status(401).json({message:"Incorrect Username or Password"});
      }
      else if(results[0].status === 'false'){
        return res.status(401).json({message:"Wait for Admin approval"});
      }
      //token generálás
      else if(results[0].password == user.password){
        const response= { email: results[0].email, role: results[0].role}
        const accessToken = jwt.sign(response,process.env.ACCESS_TOKEN,{expiresIn:'8h'})
        res.status(200).json({token:accessToken});
      }
      else{
        return req.status(400).json({message:"Something went wrong. Please try again later"});
      }
    }
    else{
      return res.status(500).json(err);
    }
  })
})
})
```

Jelszó megváltoztatása

A weboldalamon lehetőség van a jelszavak megváltoztatására, ehhez a régi jelszót kell tudni, majd megadni, hogy mi legyen az új jelszó. Ehhez azonban nem minden felhasználónak van joga, ezért bevezettem az autentikációt és egy olyan funkciót, ami a role-okat ellenőrzi. Egy új mappát készítettem "services" névvel és ide került az authentication.js és a checkRole.js fájl.

```
PlanetZooDB > services > JS authentication.js > ...
1  ...require('dotenv').config();
2  const { response } = require('express');
3  const jwt = require('jsonwebtoken');
4
5  //A token létezik a header-ben vagy nem
6  function authenticateToken(req,res,next){
7    const authHeader = req.headers['authorization'];
8    const token = authHeader && authHeader.split(' ')[1];
9    if(token == null){
10     return res.sendStatus(401);
11    }
12
13    jwt.verify(token,process.env.ACCESS_TOKEN,(err,response) =>{
14      if(err){
15        return res.sendStatus(403);
16      }
17      res.locals = response;
18      next();
19    })
20  }
21
22  module.exports = { authenticateToken: authenticateToken }
```

```
PlanetZooDB > services > JS checkRole.js > ...
1  const { response } = require('express');
2
3  require('dotenv').config();
4
5  function checkRole(req,res,next){
6    if(res.locals.role == process.env.USER){
7      response.sendStatus(401);
8    }
9    else{
10     next();
11    }
12  }
13
14  module.exports = { checkRole: checkRole }
```

```
//Jelszó megváltoztatás
router.post('/changePassword',auth.authenticateToken,checkRole.checkRole,(req,res) =>{
  const user = req.body;
  const email = res.locals.email;
  var query = "select *from user where email=? and password=?";
  connection.query(query,[email,user.oldPassword] ,(err,results) =>{
    if(!err){
      if(results.length <=0){
        return res.status(400).json({message:"Incorrect Old Password"});
      }
      else if(results[0].password == user.oldPassword){
        query = "update user set password=? where email=?";
        connection.query(query,[user.newPassword,email] ,(err,results) =>{
          if(!err){
            return res.status(200).json({message:"Password updated successfully"})
          }
          else{
            return res.status(500).json(err);
          }
        })
      }
      else{
        return res.status(400).json({message:"Something went wrong. Please try again later"});
      }
    }
    else{
      return res.status(500).json(err);
    }
  })
})
})
```

Végül pedig készítettem három további funkciót, ami a user.js fájlba került. Az egyik a get, amivel az összes felhasználót meg lehet jeleníteni az adminon kívül. A második az update, amivel a felhasználók státuszát lehet megváltoztatni. A harmadik pedig a checkToken, amivel a token ellenőrzöm.

Állatok és kategorizálásuk (Backend)

Mivel a Planet Zoo-t választottam témámnak, ezért a játékban fellelhető állatokat lehet kezelni, valamint azokat a DLC-ket, amikbe ezek az állatok tartoznak. Készítettem egy dashboard-ot is (./routes/dashboard.js), aminek annyi a feladata, hogy összeszámolja az összes adatbázisban szereplő DLC-t és állatot, az adatokat pedig később megjelenítem a frontend írásakor.

DLC

Először a DLC-ket hoztam létre, amiket id alapján több állathoz is lehet párosítani. Ehhez a routes mappában készítettem egy dlc.js fájlt, amiben az ehhez tartozó API-k vannak. Ezekkel új DLC kategóriát lehet hozzáadni, az összeset ki lehet listázni és frissíteni/változtatni lehet az adokon.

```
//Új DLC hozzáadás
router.post('/add',auth.authenticateToken,checkRole.checkRole,(req,res,next) =>{
  let dlc = req.body;
  query = "insert into dlc (name) values(?)";
  connection.query(query,[dlc.name] ,(err,results) =>{
    if(!err){
      return res.status(200).json({message:"DLC added successfully"});
    }
    else{
      return res.status(500).json(err);
    }
  })
})
```

```
//DLC-k kilistázása
router.get('/get',auth.authenticateToken,(req,res,next) =>{
  var query = "select *from dlc order by name";
  connection.query(query,(err,results) =>{
    if(!err){
      return res.status(200).json(results);
    }
    else{
      return res.status(500).json(err);
    }
  })
})
```

```
//DLC frissítés
router.patch('/update',auth.authenticateToken,checkRole.checkRole,(req,res,next) =>{
  let animal = req.body;
  var query = "update dlc set name=? where id=?";
  connection.query(query,[animal.name,animal.id],(err,results) =>{
    if(!err){
      if(results.affectedRows ==0){
        return res.status(404).json({message:"DLC id is not found"});
      }
      return res.status(200).json({message:"DLC updated successfully"});
    }
    else{
      return res.status(500).json(err);
    }
  })
})
```

Ezek segítségével (miután létrehoztam az új adatbázis táblát) feltöltöttem a táblát néhány adattal.

```
mysql> desc dlc;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id    | int           | NO   | PRI | NULL    | auto_increment |
| name  | varchar(255)  | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.03 sec)

mysql> select *from dlc;
+----+-----+
| id | name      |
+----+-----+
| 1  | Australia |
| 2  | Konzerváció |
| 3  | Twilight  |
+----+-----+
```

Állatok

A projekt egyik legfontosabb eleme az állatok kezelése, az ehhez kapcsolódó kódot az animal.js fájlban írtam meg. Minden állatnak 5 tulajdonsága van, az id-je, ami automatikusan generálódik, a neve, a hozzá tartozó DLC id-je, vagyis hogy melyik DLC-ben található meg, egy rövid leírás róla és egy státusz, ami vagy "true" vagy "false".

Először az új állat hozzáadását tettem lehetővé, hogy a táblába tölthessek adatokat, amiket a többi funkcióval aztán lehessen manipulálni.

```
//Új állat hozzáadás
router.post('/add',auth.authenticateToken,checkRole.checkRole,(req,res) =>{
  let animal = req.body;
  var query = "insert into animal (name,dlcId,description,status) values(?,?,?, 'true')";
  connection.query(query,[animal.name,animal.dlcId,animal.description] ,(err,results) =>{
    if(!err){
      return res.status(200).json({message:"Animal added successfully"});
    }
    else{
      return res.status(500).json(err);
    }
  })
})
})
```

Feltöltés után a tábla a következőképpen nézett ki:

```
mysql> select *from animal;
```

id	name	dlcId	description	status
1	Koala	1	Phascolartos cinereus	true
2	Siamang	2	Symphalangus syndactylus	true
3	Red Kangaroo	1	Macropus rufus	true

A többi állatokhoz kapcsolódó API pedig a következő:

- Összes állat kilistázása
- Állatok kilistázása egy bizonyos DLC szerint
- Egy állat megjelenítése id szerint
- Állat frissítése
- Állat törlése id szerint
- Állat státuszának megváltoztatása

Felhasználók kezelése (Frontend)

Miután végeztem a backend megírásával, elkezdtem a frontendet. Ehhez leszedtem egy template-et az internetről kiindulási alapnak, amit később kicsit átformáztam és a saját részeimmel megtoldottam. Feltelepítettem a szükséges csomagokat, valamint az environments.ts fájlba beleírtam a backend url-jét, hogy összekössem a kettőt.

Az app mappában létrehoztam egy új mappát services névvel, majd az "ng g s user" paranccsal két fájl került ide, egy user.service.ts és egy user.service.spec.ts fájl. Majd ugyanebbe a mappába még két fájlt hoztam létre (snackbar.service.ts,snackbar.service.spec.ts), amikkel az üzeneteket iratom ki a felhasználónak (error, success, stb.). Ehhez például már tartozik előre megírt formázás a styles.css fájlban. A globális konstans változóknak a shared mappában készítettem egy global-constants.ts fájlt, ide került az általános error üzenet, valamint a regex-ek (felhasználónevekhez, emailekhez).

Regisztráció

A regisztráció megírásához az app mappába került egy signup komponens ("ng g c signup") négy fájlal (.html, .scss, .spec.ts, .ts). A user.service.ts fájlba a signup-hoz tartozó metódus került.


```
signup(data:any){
  return this.httpClient.post(this.url+"/
  user/signup",data,{
    headers: new HttpHeaders().set
    ('Content-Type',"application/json")
  })
}
```

A [home.component.html](#) fájlba beleírtam a hozzá tartozó részt: egy kattintható link a regisztrációhoz, majd a [home.component.ts](#) fájlba azt a kódot, ami kattintásra felnyitja a regisztrációs ablakot.

```
<ul>
  <li><a (click)="signupAction()">Signup</a></li>
</ul>
```

```
signupAction(){
  const dialogConfig = new MatDialogConfig();
  dialogConfig.width = "550px";
  this.dialog.open(SignupComponent,dialogConfig);
}
```

Ezután a `signup.component.ts` fájlban létrehoztam a regisztrációs formot, validálom a beírt adatokat (jó-e a formátum) és a `handleSubmit()` funkcióval lekezelem a regisztrációt, azaz ha rákattint a felhasználó a regisztráció gombra, akkor hibát dob, vagy sikeres volt és továbbengedi a következő oldalra.

```
handleSubmit(){
  var formData = this.signupForm.value;
  var data = {
    name: formData.name,
    email: formData.email,
    password: formData.password
  }
  this.userService.signup(data).subscribe((response:any) =>{
    this.dialogRef.close();
    this.responseMessage = response?.message;
    this.snackbarService.openSnackBar(this.responseMessage,"");
    this.router.navigate(['/']);
  },(error) =>{
    if(error.error?.message){
      this.responseMessage = error.error?.message;
    }
    else{
      this.responseMessage = GlobalConstants.genericError;
    }
    this.snackbarService.openSnackBar(this.responseMessage, GlobalConstants.error);
  })
}
```

A `signup.html` fájljában végül a regisztrációs ablak kinézetét írtam meg. Ide tartozik egy cím, a beírandó adatok és hozzájuk tartozó szövegdozok (amikbe ha belekattint a felhasználó, de nem ír bele semmit vagy rossz a formátum és kikattint belőle, akkor hibüzenetet kap), egy vissza gomb ami bezárja az ablakot és egy regisztráció gomb, ami megnyomására lefut a `handleSubmit` kód.

Bejelentkezés

Csakúgy, mint a regisztrációnál, a bejelentkezéshez is írtam egy metódust a `user.service.ts` fájlba.

```
login(data:any){
  return this.httpClient.post(this.url+
    "/user/login/",data,{
      headers: new HttpHeaders().set('Content-Type',"application/json")
    })
}
```

Ezután login komponenseket generáltam (ng g c login), majd a homepage html-jébe beleszereltem a bejelentkezés gombját, és a ts fájlba a hozzá tartozó részt.

```
<ul>
  <li><a (click)="loginAction()">Login</a></li>
  <li><a (click)="signupAction()">Signup</a></li>
</ul>
```

```
loginAction(){
  const dialogConfig = new MatDialogConfig();
  dialogConfig.width = "550px";
  this.dialog.open(LoginComponent,dialogConfig);
}
```

Ugyanúgy, mint a regisztrációnál, a login.component.ts fájlban megírtam a formot, a constructort és a handleSubmit() metódust (itt már eltárolom a belépéskor generált token).

A belépés html-jét is ugyanúgy írtam meg, mint a regisztrációjét. Vannak szövegdozok, amiket helyesen kell kitölteni és utána rá lehet kattintani a Bejelentkezés gombra, ami aztán átirányítja a felhasználót a Dashboard-ra (ha nem kell az admin elfogadására várnia).

Jelszó megváltoztatása

Bejelentkezés után a Dashboard oldal fogadja a felhasználót, az oldal headerjében pedig található egy ikon, amire ha rákattintunk, akkor egy lenyíló ablak két lehetőséget kínál.

Az egyik a regisztráció, amire ha rákattintunk, akkor a regisztrációs ablak megnyílik, a szövegdozokat ki lehet tölteni, de az Update gomb sajnos nem működik. Így végül idő hiányában a jelszó megváltoztatását angularban nem sikerült kiviteleznem.

```
<button [matMenuTriggerFor]="profile" mat-icon-button class="m-r-5">
  <mat-icon>account_circle</mat-icon>
</button>
<mat-menu #profile="matMenu" class="mymegamenu">
  <button mat-menu-item (click)="changePassword()">
    <mat-icon>password</mat-icon> Change Password
  </button>
  <button mat-menu-item (click)="logout()">
    <mat-icon>exit_to_app</mat-icon> Sign Out
  </button>
</mat-menu>
```

Kijelentkezés

A felhasználó ikonra kattintva a második lehetőség a kijelentkezés. Ennek a funkciónak a megírását azzal kezdtem, hogy "/material-component/dialog" mappába generáltam egy confirmation komponenst. A kijelentkezés gombra kattintva egy felugró ablak fogad, ahol a felhasználótól megkérdezi az oldal, hogy biztosan ki akar-e lépni. Először megírtam azt a kódot, ami akkor szükséges, ha az "igen" gombot választjuk.

```
handleChangeAction(){
  this.onEmitStatusChange.emit();
}
```

```

<mat-dialog-content class="mat-typography">
  <h5>Are you sure you want to {{ details.message }} ?</h5>
</mat-dialog-content>
<mat-dialog-actions align="end">
  <button mat-raised-button color="primary" (click)="handleChangeAction()">Yes</button>
  <button mat-raised-button mat-dialog-close>No</button>
</mat-dialog-actions>

```

Majd a header.components.ts fájlba került bele a logout() funkció, ami végrehajtja a kijelentkezést (visszairányít a homepage-re és törli a bejelentkezéskor generált token). Ezt a funkciót html-ben hozzákötöttem a kijelentkezés gombhoz.

```

logout() {
  const dialogConfig = new MatDialogConfig();
  dialogConfig.data = {
    message: 'Logout',
  };
  const dialogRef = this.dialog.open(ConfirmationComponent,
    dialogConfig);
  const sub = dialogRef.componentInstance.onEmitStatusChange.subscribe(
    (user) => {
      dialogRef.close();
      localStorage.clear();
      this.router.navigate(['/']);
    }
  );
}

```

Menü

Bejelentkezés után bal oldalon található a menü, amivel a különböző oldalak között navigálhatunk. Először a menu-items.ts-t készítettem el, amiben a menüben kiválasztható elemeket formázom.

```

import { Injectable } from '@angular/core';

export interface Menu {
  state: string;
  name: string;
  icon: string;
  role: string;
}

const MENUITEMS = [
  { state: 'dashboard', name: 'Dashboard', icon: 'dashboard', role: '' },
];

@Injectable()
export class MenuItems {
  getMenuitem(): Menu[] {
    return MENUITEMS;
  }
}

```

Ezután a sidebar html fájljába írtam bele a menüpontok és azok ikonjainak dinamikus megjelenítéséhez szükséges sorokat (for ciklussal).

```

<mat-nav-list appAccordion>
  <mat-list-item appAccordionLink *ngFor="let menuitem of menuItems.getMenuitem()" routerLinkActive="selected" group="({ menuitem.state })">
    <a appAccordionToggle [routerLink]="['/pz',menuitem.state]" *ngIf="(menuitem.role === '' || menuitem.role === tokenPayload.role)">
      <mat-icon>{{ menuitem.icon }}</mat-icon>
      <span>{{ menuitem.name }}</span>
      <span fxFlex></span>
    </a>
  </mat-list-item>
</mat-nav-list>

```

Dashboard

A dashboard oldalon a DLC és az állat kategóriákat jelenítem meg, kiíratom az összes adatbázisban szereplő DLC és állat számát, valamint egy-egy gombot, amivel átirányít annak az oldalára, amelyiknek a gombját megnyomtuk.

Ehhez a services mappában készítettem dashboard ts fájlokat. A dashboard.service.ts-ben található a getDetails() funkció, amivel a backendben megírt dashboard/details kötöm össze, ami a DLC-k és az állatok összesített számát mondja meg.

```

getDetails() {
  return this.httpClient.get(this.url + '/dashboard/details');
}

```

Ezután a dashboard.component.ts fájlban írtam meg az ehhez kapcsolódó függvényt, aminek segítségével aztán a html fájlban megjelenítem a kapott adatokat. A két kártyához tartoznak a gombok, amikkel a megfelelő oldalra navigálhatunk.

```

constructor(
  private dashboardService: DashboardService,
  private snackbarService: SnackbarService
) {
  this.dashboardData();
}

dashboardData() {
  this.dashboardService.getDetails().subscribe(
    (response: any) => {
      this.data = response;
    },
    (error: any) => {
      console.log(error);
      if (error.error?.message) {
        this.responseMessage = error.error?.message;
      } else {
        this.responseMessage = GlobalConstants.genericError;
      }
      this.snackbarService.openSnackBar(
        this.responseMessage,
        GlobalConstants.error
      );
    }
  );
}

```

```

<body>
  <mat-card>
    <b><span>Dashboard</span></b>
  </mat-card>
  <br>
  <div class="row">
    <div class="column">
      <div class="card">
        <div class="container">
          <h2 class="title">Total DLC:</h2>
          <h1 class="title">{{ data?.dlc }}</h1>
          <p><button class="button" [routerLink]="['/pz/dlc']">View DLC</button></p>
        </div>
      </div>
    </div>
    <div class="column">
      <div class="card">
        <div class="container">
          <h2 class="title">Total Animal:</h2>
          <h1 class="title">{{ data?.animal }}</h1>
          <p><button class="button" [routerLink]="['/pz/animal']">View Animal</button></p>
        </div>
      </div>
    </div>
  </div>
</body>

```

DLC-k oldala

A DLC-k menedzseléséhez készítettem egy service-t dlc névvel. Ide került az új hozzáadása, a frissítés és az összes kilistázása funkciók meghívása. A material.routing.ts fájlba leírtam, hogy ezt az oldalt csak az admin láthassa a sidebar-on.

Következőnek a material-component mappába generáltam egy manage-dlc komponenst, ahova a `tableData()` és az `applyFilter` funkció került (az adatok között lehet keresni a beírt adat alapján). Ide került még az új hozzáadása (`handleAddAction`) és a meglévő szerkesztése (`handleEditAction`) funkciók lekezelése.

```
handleAddAction() {
  const dialogConfig = new MatDialogConfig();
  dialogConfig.data = {
    action: 'Add',
  };
  dialogConfig.width = '850px';
  const dialogRef = this.dialog.open(DlcComponent, dialogConfig);
  this.router.events.subscribe(() => {
    dialogRef.close();
  });
  const sub = dialogRef.componentInstance.onAddDlc.subscribe
  ((response) => {
    this.tableData();
  });
}
```

```
handleEditAction(values: any) {
  const dialogConfig = new MatDialogConfig();
  dialogConfig.data = {
    action: 'Edit',
    data: values,
  };
  dialogConfig.width = '850px';
  const dialogRef = this.dialog.open(DlcComponent, dialogConfig);
  this.router.events.subscribe(() => {
    dialogRef.close();
  });
  const sub = dialogRef.componentInstance.onEditDlc.subscribe
  ((response) => {
    this.tableData();
  });
}
```

Az ehhez tartozó html-be bekerült az új hozzáadása gomb, a kereső, az összes DLC felsorolva, mellettük pedig egy szerkesztés ikon, amire rákattintva megnyílik a szerkesztő ablak.

A dialog mappába generáltam egy dlc komponenst, ami a felugró ablakokhoz tartozó kódot tartalmazza. Az add és edit ablakok ugyanúgy működnek és néznek ki, így egy html fájl elég volt mindkettőhöz, attól függ az ablakban megjelenített cím és "ok" gomb, hogy melyik funkció gombjára kattintottunk.

```

<mat-toolbar color="primary">
  <mat-toolbar-row fxLayout="row">
    <span class="title-center">{{ dialogData.action }} DLC</span>
  </mat-toolbar-row>
</mat-toolbar>

<mat-dialog-content class="mat-typography">
  <form [formGroup]="dlcForm">
    <div fxFlex fxLayout="column">
      <div fxLayout="row wrap">
        <mat-form-field appearance="fill" fxFlex>
          <mat-label>Name</mat-label>
          <input matInput formControlName="name" required />
          <mat-error *ngIf="dlcForm.controls.name.touched && dlcForm.controls.name.invalid">
            <span *ngIf="dlcForm.controls.name.errors.required">
              >This field is mandatory</span>
            </mat-error>
          </mat-form-field>
        </div>
      </div>
    </div>
  </form>
</mat-dialog-content>

<mat-dialog-actions align="center">
  <button mat-raised-button color="primary" type="submit"(click)="handleSubmit()"
    [disabled]="!(dlcForm.valid && dlcForm.dirty)">{{ action }}</button>
  <button mat-raised-button color="primary" mat-dialog-close="">Close</button>
</mat-dialog-actions>

```

Állatok oldala