DOKUMENTÁCIÓ

Web technológiák 2

Készítette: **Nyíri Beáta**

Neptunkód: **I40FDC**

Dátum: **2023.06.20.**

Tartalom

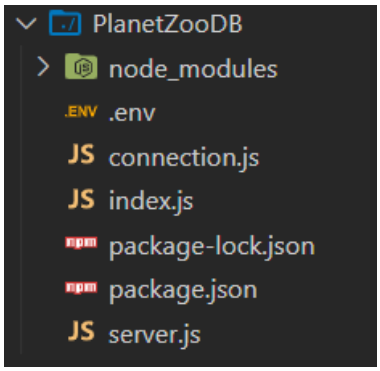
A feladat leírása

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

Ehhez a kiadott feladat leírásában feltüntetetett technológiákat alkalmaztam (Angular, NodeJS) az adatbáziskezelön 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 fájban a globális változókat tárolom, connection.js fájl segítségével pedik az adatbázishoz kapcsolódtam, 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 bövült.

Felhasználók kezelése (Backend)

Az oldalt úgy építettem fel, hogy kétféle bejelentkezési lehetőség legyen. Be lehet lépni adminként és egyszerü 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 elsöként a regisztrációhoz készítettem egy API-t. A user.js fájban az ehhez szükséges kód van. Az email alapján ellenörzöm 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 regisztralni kívánó felhasználó, akkor az adatai bekerülnek az adatbázisba és egy üzenet jelenik meg, miszerint a regisztráció sikeres volt.

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 authentikációt és egy olyan funkciót, ami a role-okat ellenörzi. Egy új mappát készítettem “services” névvel és ide került az authentication.js és a checkRole.js fájl.

PlanetZooDB > services > JS authentication.js > ...

```
require('dotenv').config();
const { response } = require('express');
const jwt = require('jsonwebtoken');

//A token létezik a header-ben vagy nem
function authenticateToken(req,res,next){
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];
    if(token == null){
        return res.sendStatus(401);
    }

    jwt.verify(token,process.env.ACCESS_TOKEN,(err,response) =>{
        if(err){
            return res.sendStatus(403);
        }
        res.locals = response;
        next();
    })
}

module.exports = { authenticateToken: authenticateToken }
```

PlanetZooDB > services > JS checkRole.js > ...

```
const { response } = require('express');

require('dotenv').config();

function checkRole(req,res,next){
    if(res.locals.role == process.env.USER){
        response.sendStatus(401);
    }
    else{
        next();
    }
}

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 tokent ellenörzöm.

Állatok és kategorizálásuk (Backend)

Mivel a Planet Zoo-t választottam témámnak, ezért a játékban fellelhetö 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 szereplö DLC-t és állatot, az adatokat pedig késöbb megjelenítem a frontend írásakor.

DLC

Elöször 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 | Conzervation |
|  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 melyhik DLC-ben található meg, egy rövid leírás róla és egy státusz, ami vagy “true” vagy “false”.

Elöször az új állat hozzáadását tettem lehetövé, hogy a táblába tölthessek adatokat, amiket a többi funkcióvan 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 következöképpen 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 következö:

- 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 internetröl kiindulási alapnak, amit késöbb kicsit átformáztam és a saját részeimmel megtoldottam. Feltelepítettem a szükséges csomagokat, valamint az environments.ts fájlba beleírtam a backend url-jét, hogy összekössem a kettöt.

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 elöre 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 fájllal (.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 következö 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 tarozó szövegdobozok (amikbe ha belekattint a felhasználó, de nem ír bele semmit vagy rossz a formátum és kikattint belöle, akkor hibaüzenetet kap), egy vissza gomb ami bezárja az ablakot és egy regisztráció gomb, ami megnyomására lefut a handleSubmit kód.

Bejelentkezés

Csakúgy, mint a regisztációnál, 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 belerakram 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.conponent.ts fájlban megírtam a formot, a constructort és a handleSubmit() metódust (itt már eltárolom a belépéskor generált tokent).

A belépés html-jét is ugyanúgy írtam meg, mint a regisztrációjét. Vannak szövegdobozok, 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