

# **Web technológiák 2**

Féléves beadandó feladat

Miskolci Egyetem

2023

Készítette: Nyeste Ágoston

Neptunkód: WH85ZH

## -Bevezetés

Elsődlegesen bemutatom a felhasználói felületet és funkciókat utána részletezem a megvalósítási folyamatot.

## -Konceptió

Egy labdarúgó csapat kommunikációját segítő webalkalmazás.

Posztok megosztása, módosítása törlése és hozzászólás.

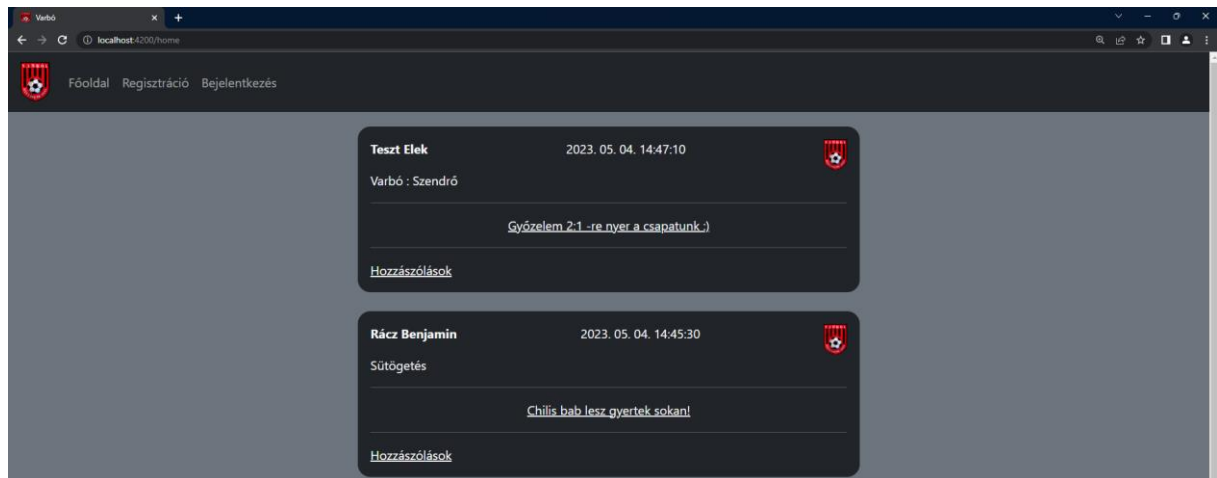
Bejelentkezés rendszer.

## -Technológia

A webalkalmazás MEAN stack-el készült el, az adatbázishoz MongoDB-t, a backendhez NodeJs-t, Frontendhez Angular-t használtam

## -Felület

### *Főoldal*




A főoldalon találhatóak a posztok amit a regisztrált felhasználók hoztak létre.

A hozzászólás gombra kattintva láthatjuk a kommenteket.

Kommentet csak regisztrált felhasználó adhat hozzá bejelentkezés után.

Nyeste Agoston

2023. 05. 04. 14:44:07



Meccs

Szombat 15:00 találkozó az öltözőben ne késsetek!

Hozzászólások

Rendben

Küldés

[A kommenteléshez jelentkez be.](#)

-Teszt Elek: nemjo :(

-Rác Benjamin: Leszek

-Nyeste Agoston: Leszek

Kommentek

## Regisztráció

Ezen a felületen lehet új felhasználót regisztrálni.

Felhasználónév

Email

Jelszó

Regisztráció

## Bejelentkezés

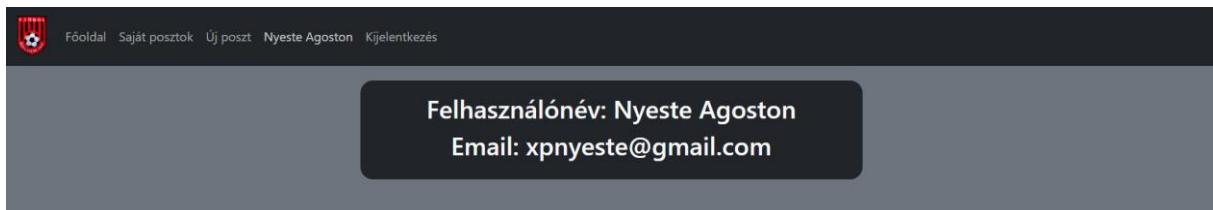
Itt lehet bejelentkezni

Felhasználónév

Jelszó

Bejelentkezés

Bejelentkezés után a navigációs sávon megjelennek a saját posztok és az új poszt oldal.



## Új poszt

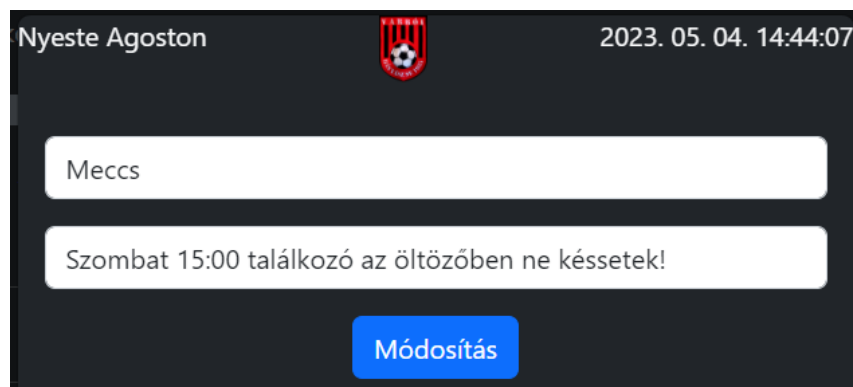
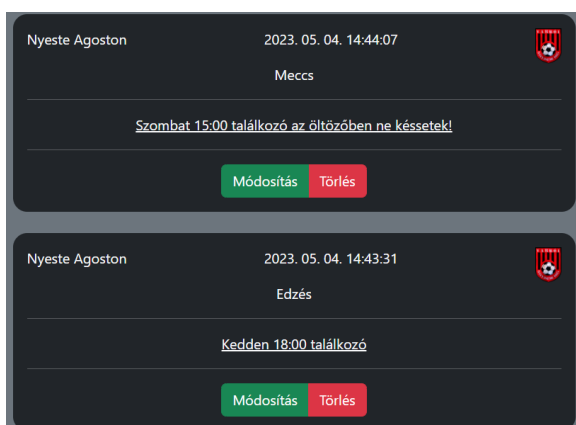
A tárgy és szöveg megadásával létrehozhatunk egy új posztot.

A dark-themed form with two white input fields. The first field is labeled 'Tárgy' and the second is labeled 'Üzenet'. Below the fields is a blue button labeled 'Küldés'.

## Saját posztok

Itt találhatóak a bejelentkezett felhasználó által létrehozott posztok.

A törlés gomb megnyomásával törölhetjük, a módosítással pedig módosíthatjuk a posztot.



# -Megvalósítás

## -Backend

A backend nevű mappában telepítettem a szükséges csomagokat:

- *npm init*: telepíti a node csomagokat
- *npm i -g nodemon*: minden mentésnél újraindul a server
- *npm i express*: a routingban segít
- *npm i mongoose*: MongoDB kapcsolatok
- *npm i jsonwebtoken*: biztonságos JSON továbbítás
- *npm i bcrypt*: jelszó titkosítás
- *npm i cors*: klienssel való összekötés
- *npm i cookie-session*: a user sessiont cookiban tárolja

Az adatbázis három táblából áll: *User, Post, Comment*

Ezekhez létrehoztam a modelleket a mongoose csomag segítségével.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const commentSchema = new Schema({
  _id:{type:Schema.Types.ObjectId, auto: true},
  post_id:{type:String, required: true},
  username:{type:String, required: true},
  text:{type:String, required: true},
  date:{type:String, required: true},
},{
  versionKey: false
});

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const postSchema = new Schema({
  _id:{type:Schema.Types.ObjectId, auto: true},
  username:{type:String, required: true},
  date:{type:String, required: true},
  title:{type:String, required: true},
  text:{type:String, required: true},
},{
  versionKey: false
});

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const userSchema = new Schema({
  _id:{type:Schema.Types.ObjectId, auto: true},
  username:{type:String, required: true},
  password:{type:String, required: true},
  email:{type:String,required: false},
},{
  versionKey: false
});
```

Létrehoztam az adatbázis modellt: *db.js*

```
const mongoose = require('mongoose');

const mongoDB_Url = process.env.MongoDB_Url;

mongoose.connect(mongoDB_Url);

mongoose.connection.on('connected',()=>{
  console.log('Connected succesfully!');
});

mongoose.connection.on('error',(err)=>{
  console.log('valami szar');
});
```

Egy *.env* fileban meghatároztam a MongoDB forrását

```
MongoDB_Url = "mongodb://127.0.0.1:27017/webtech2"
PORT = 3000
```

Mind háromhoz létrehoztam a controllereket amik a *létrehozás, törlés, frissítés, olvasás* operációkat végzik.

*comment.controller.js*

```
const httpStatusCode = require('http-status-codes');
const comment = require('../models/comment.model');
//get comments by Post id
exports.getComments = (req,res) =>{
  comment.find({post_id: req.body.post_id}).then(docs=>{
    res.send(docs);
  }).catch(err=>{
    res.status(httpStatusCode.INTERNAL_SERVER_ERROR).send(err);
  })
}
//get all comments
exports.getAll = (req,res) =>{
  comment.find().sort({date: -1}).then(docs=>{
    res.send(docs);
  }).catch(err=>{
    res.status(httpStatusCode.INTERNAL_SERVER_ERROR).send(err);
  })
}
//create comment
exports.createComment = (req,res) =>{
  const obj = req.body;
  comment.create(obj).then(doc=>{
    res.status(httpStatusCode.CREATED).send(doc);
  }).catch(err=>{
    res.status(httpStatusCode.INTERNAL_SERVER_ERROR).send(err);
  })
};
```

*post.controller.js* példa

a `getPost` a jsonwebtoken-ben tárolt jelenleg bejelentkezett felhasználóhoz tartozó posztokat adja vissza, ez a saját posztok oldalhoz kell.

```
//get posts by id
exports.getPost = (req,res) =>{
  let id = "null";
  let token = req.session.token;
  let name = "null";
  jwt.verify(token, config.secret, (err, decoded) => {
    id = decoded.id;
  });
  User.findOne({_id: id}).then(docs =>{
    post.find({username: docs.username}).sort({date: -1}).then(docs=>{
      res.send(docs);
    }).catch(err=>{
      res.status(httpStatusCode.INTERNAL_SERVER_ERROR).send(err);
    })
  });
};
//create new post
exports.post = (req,res) =>{
  const obj = req.body;
  post.create(obj).then(doc=>{
    res.status(httpStatusCode.CREATED).send(doc);
  }).catch(err=>{
    res.status(httpStatusCode.INTERNAL_SERVER_ERROR).send(err);
  })
};
```

*user.controller.js*

Regisztrációnál a *bcrypt* csomagot használva titkosítva tárolom el a jelszót.

```
//create new user
exports.signup = (req,res) =>{
  const obj = new User({
    username: req.body.username,
    email: req.body.email,
    password: bcrypt.hashSync(req.body.password, 8),
  });
  User.create(obj).then(doc=>{
    res.status(httpStatusCode.CREATED).send(doc);
  }).catch(err=>{
    res.status(httpStatusCode.INTERNAL_SERVER_ERROR).send(err);
  })
};
```

Kijelentkezésnél a sessiont üresre állítom.

```
//sign user out
exports.signout = async (req, res) => {
  try {
    req.session = null;
    return res.status(200).send({ message: "You've been signed out!" });
  } catch (err) {
    this.next(err);
  }
};
```

Bejelentkezésnél ellenőrzöm, hogy létezik e a felhasználónév, utána a jelszó egyezőségét. Ezután beállítom a *cookie-session*t az adott user azonosítójával.

```
//bejelentkezés
exports.signin = async (req, res) => {
  try {
    const user = await User.findOne({ username: req.body.username });
    if (!user) {
      return res.status(404).send({ message: "No such user" });
    }
    const passwordIsValid = bcrypt.compareSync(req.body.password, user.password);
    if (!passwordIsValid) {
      return res.status(401).send({ message: "Invalid Password!" });
    }
    const token = jwt.sign({ id: user.id }, config.secret, { expiresIn: 86400 });
    req.session.token = token;
    res.status(200).send({
      id: user._id,
      username: user.username,
      email: user.email
    });
  } catch (err) {
    res.status(500).send({ message: err.message });
  }
};
```

Két *middleware* file-t is létrehoztam, az egyik azt figyeli, hogy létezik-e a *token session*, azaz be van-e jelentkezve egy felhasználó.

```
verifyToken = (req, res, next) => {
  let token = req.session.token;
  if (!token) {
    return res.status(403).send({ message: "No token provided!" });
  }
  jwt.verify(token, config.secret, (err, decoded) => {
    if (err) {
      return res.status(401).send({ message: "Unauthorized!" });
    }
    req.userId = decoded.id;
    next();
  });
};
```

A másik regisztrációnál figyeli, hogy a felhasználónév és email már foglalt-e.

```
checkDuplicateUsernameOrEmail = (req, res, next) => {
  // Username
  User.findOne({ username: req.body.username })
    .then(user => {
      if (user) {
        res.status(400).send({ message: "Failed! Username is already in use!" });
        return;
      }
      // Email
      User.findOne({ email: req.body.email })
        .then(user => {
          if (user) {
            res.status(400).send({ message: "Failed! Email is already in use!" });
            return;
          }
          next();
        })
        .catch(err => {
          res.status(500).send({ message: err });
        });
    })
    .catch(err => {
      res.status(500).send({ message: err });
    });
};
```

A *route* fájlokban megterveztem, hogy a *controllerek* adott elemei milyen elérési útvonalon legyenek elérhetőek.

A posztok útvonalainál látható, hogy a *create update delete*n parancshoz bejelentkezés szükséges, mivel a *verifyToken middleware* itt meg van hívva.

```
module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "Origin, Content-Type, Accept"
    );
    next();
  });
  app.get("/api/post/all", controller.get);
  app.post("/api/post/create", [authJwt.verifyToken], controller.post);
  app.put("/api/post/update", [authJwt.verifyToken], controller.put);
  app.post("/api/post/delete", [authJwt.verifyToken], controller.delete);
  app.get("/api/post/search", controller.getPost);
  app.post("/api/post/single", controller.getSinglePost);
};
```



A kommenteknél csak a létrehozáshoz kell érvényes *token*.

```
app.post("/api/comment/get", controller.getComments);
app.get("/api/comment/all", controller.getAll);
app.post("/api/comment/create",[authJwt.verifyToken],controller.createComment);
```

A felhasználóknál a regisztrációnál használom az felhasználónév, email *middleware*-et.

```
app.post([
  "/api/auth/signup",[verifySignUp.checkDuplicateUsernameOrEmail],
  controller.signup]);
app.post("/api/auth/signin", controller.signin);
app.post("/api/auth/signout", controller.signout);
```

A szervert a *server.js* fileal indítható el a nodemon server parancsal. Itt kell beállítani a környezetet, adatbázist, cookie-át, és a routingot. A szerver a 3000-es porton fut.

```
const express = require('express'),
cors = require('cors');
const cookieSession = require("cookie-session");
//setup environment
require('dotenv').config();
//database
require('./server/config/db');
const app = express();
var corsOptions = {
  origin: ["http://localhost:4200"],
  credentials: true,
  optionSuccessStatus:200
}
app.use(cors(corsOptions));
// parse requests of content-type - application/json
app.use(express.json());
// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));
app.use(
  cookieSession({
    name: "lupilu-session",
    secret: "martyxmo", // should use as secret environment variable
    httpOnly: true
  })
);
app.use(express.json());
//routing user login
require("./server/routes/auth.routes")(app);
require("./server/routes/user.routes")(app);
require("./server/routes/post.routes")(app);
require("./server/routes/comment.routes")(app);
//routing posts
const port = process.env.PORT || 3000;
app.listen(port,()=>{console.log('Server is running at http://localhost:3000')});
```

## -Client

A client mappában kiadtam a *ng new client* parancsot ami létrehozta a projectet, az *ng serve*-el pedig elindítottam a 4200-as porton. A designhoz *bootstrap*et használtam amit hozzáadtam a projecthez.

Létrehoztam az táblákhoz tartozó *interface*eket.

```
export interface Comments {  
  _id?: string;  
  post_id?: string;  
  username?: string;  
  text?: string;  
  date?: string;  
}
```

```
export interface Posts {  
  _id?: string;  
  username?: string;  
  date?: string;  
  title?: string;  
  text?: string;  
}
```

Létrehoztam a *service*-eket amik a backend-el kommunikálnak.

*auth.service*, először eltárolom a szerveren erre a célra beállított útvonalat

```
const AUTH_API = 'http://localhost:3000/api/auth/';
```

Itt a felhasználóhoz tartozó műveletek vannak.

```
login(username: string, password: string): Observable<any> {  
  return this.http.post(  
    AUTH_API + 'signin',  
    {  
      username,  
      password,  
    },  
    httpOptions  
  );  
}  
  
register(username: string, email: string, password: string): Observable<any> {  
  return this.http.post(  
    AUTH_API + 'signup',  
    {  
      username,  
      email,  
      password,  
    },  
    httpOptions  
  );  
}  
  
logout(): Observable<any> {  
  return this.http.post(AUTH_API + 'signout', { }, httpOptions);  
}
```

A post service-ben először létrehozok két *Subject*et a *post interface* segítségével.

```
private posts$: Subject<Posts[]> = new Subject();
private post$: Subject<Posts> = new Subject();
```

Private metódusokkal elértem az adatokat, pl.: az alábbi visszaadja az összes posztot

```
private refreshPosts() {
  this.http.get<Posts[]>(API_URL + 'all')
    .subscribe(posts => {
      this.posts$.next(posts);
    });
}
```

Ezt egy publikus metódus pedig meghívja, ezt a metódust hívhatják meg a komponensek.

```
getPosts(): Subject<Posts[]> {
  this.refreshPosts();
  return this.posts$;
}
```

Itt található még az *update*, *create*, és egyéb *get* metódusok is.

A comment szervice hasonló felépítésű mint a post service, itt is *Subject*eket hozok létre az interfacel.

Storage service-ben a bejelentkezést lehet állítani

```
public saveUser(user: any): void {
  window.sessionStorage.removeItem(USER_KEY);
  window.sessionStorage.setItem(USER_KEY, JSON.stringify(user));
}

public getUser(): any {
  const user = window.sessionStorage.getItem(USER_KEY);
  if (user) {
    return JSON.parse(user);
  }

  return {};
}

public isLoggedIn(): boolean {
  const user = window.sessionStorage.getItem(USER_KEY);
  if (user) {
    return true;
  }

  return false;
}
```

- *saveUser* metódus elmenti a bejelentkezett felhasználót
- *getUser* visszatér a bejelentkezett felhasználó adatait
- *isLoggedIn* pedig megmondja, hogy jelenleg be van-e jelentkezve valaki

Ezeket követően létrehoztam az oldalakhoz tartozó komponenseket

```
> board-user
> create-post
> home
> interfaces
> login
> profile
> register
```

Példa Service használatra

Az alábbiakban a felhasználó regisztráció segítségével fogom bemutatni egy service működését.

A komponenshez tartozó html kódban egy form-control segítségével kérem be az adatokat, kezelve a hibás inputokat is. pl.: a felhasználónév minimum 3 karakter hosszú

```
<form
  *ngIf="!isSuccessful"
  name="form"
  (ngSubmit)="f.form.valid && onSubmit()"
  #f="ngForm"
  novalidate
>
  <div class="form-group mt-3">
    <input
      placeholder="Felhasználónév"
      type="text"
      class="form-control"
      name="username"
      [(ngModel)]="form.username"
      required
      minlength="3"
      maxlength="20"
      #username="ngModel"
      [ngClass]="{ 'is-invalid': f.submitted && username.errors }"
    />
    <div class="invalid-feedback" *ngIf="username.errors && f.submitted">
      <div *ngIf="username.errors['required']">A felhasználónév kötelező</div>
      <div *ngIf="username.errors['minlength']">
        A felhasználónév minimum 3 karakter hosszú
      </div>
      <div *ngIf="username.errors['maxlength']">
        A felhasználónév maximum 20 karakter hosszú
      </div>
    </div>
  </div>
</div>
```

A komponens typescript filejában létrehozok egy form propetit a kívánt tagokkal. A form véglegesítésénél lefut az *onSubmit* metódus és a konstruktorban beállított *authService* service-t használva végrehajtja a regisztrálás parancsot.

```
export class RegisterComponent implements OnInit {  
  form: any = {  
    username: null,  
    email: null,  
    password: null  
  };  
  isSuccessful = false;  
  isSignUpFailed = false;  
  errorMessage = '';  
  
  constructor(private authService: AuthService) { }  
  
  ngOnInit(): void {  
  }  
  //on form submit  
  onSubmit(): void {  
    const { username, email, password } = this.form;  
  
    this.authService.register(username, email, password).subscribe({  
      next: data => {  
        console.log(data);  
        this.isSuccessful = true;  
        this.isSignUpFailed = false;  
      },  
      error: err => {  
        this.errorMessage = err.error.message;  
        this.isSignUpFailed = true;  
      }  
    });  
  }  
}
```