

PRF kötelező program

Fejlesztési napló

Potyesz Máté

1. A webapp rövid leírása és elindítása

Az alkalmazásom egy egyszerű futballcsapat és focista listázós webapp. Az egyszerű user tudja böngészni a csapatokat.

A csapatokra kattintva meg tudja nézni a csapat részletes információit. A csapat játékosai gombra kattintva pedig az adott csapat játékosait tudja megnézni. Azokra kattintva pedig a játékosok részletes információit tudja megnézni.

A navbar-nál van egy profile gomb, amelyre kattintva egy egyszerű Profil ablakra érünk, ahol a bejelentkezett user a felhasználónevét és emailcímét tudja megnézni.

Az admin role-al rendelkező felhasználónál (alapértelmezetten bootstrappelve newAdmin nevű felhasználó az admin123 jelszóval) megjelenik három lehetőség a navbarban pluszban. Felhasználó/játékos és csapat módosítás. Ezekre kattintva megjelenik az adott modell Admin felülete. Mindegyiken lehet törölni, módosítani és hozzáadni adott játékost/csapatot/felhasználót. Az admin tudja tehát a user role-ját később módosítani, így adminná rakni valakit.

A felhasználónak először be kell regisztrálnia, majd bejelentkeznie ahhoz hogy a weboldalra belépjen.

A továbbiakban pedig lépésről lépésre megyünk a követelményeken és screenshootokkal bemutatom azok megvalósítását.

A webapp elindításához add ki először az npm install parancsot. Majd futtasd a szerveret (pl. *nodemon index.js* parancsal). Ha a frontendet is akarod tesztelni, akkor nyisd meg a frontend pappában lévő angular projektet és add ki ott is az npm install parancsot (vagy biztonság kedvéért add ki mindenképp). A szerver mivel statikusan hostolja a frontendet ezért elég ha a localhost:3000-es portot írod be (mindenképp csak így írd be, ekkor lesz jó: *localhost:3000*). A mongodb link lentebb van, de a kódban is benne van, szóval nem kell bemásolgatni.

Github repo: <https://github.com/potyeszmate/PRF>

2. Backend

A backend statikusan hostolja a frontendet:

A frontendet lebuildeltem az `ng build --configuration=production` parancsal, majd az elkészült fájlokat átmásoltam a backend public mappájába. Ezután pedig az `index.js`-be beírtam a következő parancsot:

```
app.use(express.static(path.join(__dirname, 'public')));
```

A backend tehát így már statikusan hosztolja a frontendet.

Az alkalmazás kapcsolódik egy mongodb instance-hoz :

Az alkalmazás az alábbi módon csatlakozik a mongodb instance-hoz:

```
16 const dbUrl = 'mongodb+srv://potyeszmate:password1234@prf-claster.wbtvog6.mongodb.net/?retryWrites=true&w=majority';  
17  
18 mongoose.connect(dbUrl);
```

Tehát a pojekthez mongoDB Azlast használtam, melynek linkje már szerepel az `index.js`-ben:

'mongodb+srv://potyeszmate:password1234@prf-claster.wbtvog6.mongodb.net/?retryWrites=true&w=majority';

Az atlasban engedélyezve van, hogy bármilyen IP címről lehessen rá csatlakozni.

Az alkalmazás képes bootstrappelni, vagyis MongoDB-t alap userekkel feltölteni:

A `bootstrap.js` felelős ezért. Alapértelmezetten létrehoz egy admin-t ha az nincs a db-ben a szervez elindultakor (`newadmin – admin123`)

```

async function ensureAdminExists() {
  try {
    const admin = await User.findOne({ accesLevel: 'admin' });
    if (admin) {
      console.log('Az admin felhasználó már megtalálható az adatbázisban!');
    } else {

      const username = 'newAdmin';
      const existingUser = await User.findOne({ username });
      if (existingUser) {
        console.log('A megadott felhasználónév már létezik!');
      } else {
        const newAdmin = new User({
          username,
          email: 'admin@admin.com',
          password: 'admin123',
          accesLevel: 'admin',
        });
        await newAdmin.save();
        console.log('Az admin felhasználó sikeresen létrehozva!');
      }
    }
  } catch (error) {
    console.error('Hiba történt az admin ellenőrzése vagy létrehozása során: ', error);
  }
}

```

A szerver megvalósít legalább két modellt, melyek sémája egyértelműen definiált:

A szerverben 3 modell szerepel. csapat, játékos és user modell. Egy modell így épül fel:

```

var playerSchema = new mongoose.Schema({
  playername: { type: String, unique: true, required: true},
  value: { type: Number, required: true},
  goals : { type: Number, required: true},
  country : { type: String, required: true},
  teamname : { type: String, required: true},
}, {collection: 'player'});

```

Adott legalább két olyan adatbázis hook, amelyek a modellek mentése vagy lekérése közben futnak le:

Sok ilyen példa van, például a user regisztrálásnál, vagy user loginnál, bootstrapnél (ha az is annak számít), stb. Ilyen például a regisztrációnál a titkosítás és a role adása:

```

13     userSchema.pre('save', function(next) {
14         const user = this;
15         if(user.isModified('password')) {
16             if(user.username === 'newAdmin')
17             {
18                 user.accessLevel = 'admin';
19             }else{
20                 user.accessLevel = 'user';
21             }
22         }
23         bcrypt.genSalt(10,function(err, salt) {
24             if(err)
25             {
26                 console.log('error in gen salt')
27                 return next(err);
28             }
29             bcrypt.hash(user.password, salt,function(error, hash) {
30                 if(error){
31                     console.log('error in hash');
32                     return next(error);
33                 }
34                 user.password = hash;
35                 return next();
36             });
37         });
38     });
39 }
40 }
41 else{
42     return next();
43 }
44 }
45 })
46

```

A szerver megvalósít egy lokális autentikációs stratégiát:

A szerver a passportjs autentikációs stratégiát használja.

index.js:

```

57
58 passport.use('local', new LocalStrategy(async function(username, password, done) {
59     try {
60         const user = await userModel.findOne({ username: username });
61         if (!user) return done(null, false, { message: 'User not found' });
62         user.comparePassword(password, function(error, isMatch) {
63             if (error) return done(error, false);
64             if (!isMatch) return done(null, false, { message: 'Wrong password' });
65             return done(null, isMatch);
66         });
67     } catch (err) {
68         return done('hiba a lekeres során', null);
69     }
70 });
71
72 passport.serializeUser(function(user, done) {
73     if (!user) return done('nincs megadva beleptetheto user', null);
74     return done(null, user);
75 });
76
77 passport.deserializeUser(function(user, done) {
78     if (!user) return done('nincs kileptetheto user', null);
79     return done(null, user);
80 });
81

```

A szerver kezeli a login sessiont:

Fentebb is mutattva, a szerver kezeli a login sessionömet. Még kód példar rá a fentén kívül:

user.model.js:

```
userSchema.methods.comparePassword = function(password, nx) {  
  bcrypt.compare(password, this.password, function(err, isMatch) {  
    nx(err, isMatch);  
  });  
}
```

route.js: itt ha a státust le tudjuk kérni a session közbe egy api hívással a szervertől és megkapjuk az éppen bejelentkezett usert.

```
36  
37 //user status  
38 router.route('/status').get((req, res, next) => {  
39   if(req.isAuthenticated()) {  
40     return res.status(200).send(req.session.passport);  
41   }else {  
42     return res.status(403).send('Nem volt bejelentkezve');  
43   }  
44 });  
45
```

A szerver rendelkezik a két kezelt modell CRUD interfészeivel, illetve egy login, logout, register route-tal:

A szerver CRUD route-jai, illetve a login/logout/register route-jai a route.s fájlban vannak. Példáik:

Login/Logout:

```
12 //login  
13 router.route('/login').post((req, res, next) => {  
14   if (req.body.username, req.body.password) {  
15     passport.authenticate('local', function(error, user) {  
16       if(error) return res.status(500).send(error);  
17       req.login(user, function(error) {  
18         if(error) return res.status(500).send(error);  
19         return res.status(200).send('Bejelentkezés sikeres');  
20       });  
21     })(req, res);  
22   } else {  
23     return res.status(400).send('Hibas keres, username es passw kell');  
24   }  
25 });  
26  
27 //logout  
28 router.route('/logout').post((req, res, next) => {  
29   if(req.isAuthenticated()) {  
30     req.logout();  
31     return res.status(200).send('Kijelentkezés sikeres');  
32   } else {  
33     return res.status(400).send('Nem volt bejelentkezve');  
34   }  
35 });
```

Register:

```
48 router.route(['/user'])
49 .get((req, res, next) => {
50   userModel.find({})
51   .then((users) => {
52     res.status(200).send(users);
53   })
54   .catch((err) => {
55     res.status(500).send('DB error');
56   });
57 })
58 //register a user
59 .post((req, res, next) => {
60   if (req.body.username && req.body.email && req.body.password) {
61     userModel.findOne({ username: req.body.username })
62     .then((user) => {
63       if (user) {
64         return res.status(400).send('Ez a felhasználónév már létezik');
65       }
66
67       const usr = new userModel({
68         username: req.body.username,
69         email: req.body.email,
70         password: req.body.password,
71       });
72
73       return usr.save().then(() => {
74         res.status(202).send('Successful save ');
75       });
76     })
77     .catch((err) => {
78       console.error(err);
79       res.status(500).send('Error while saving user');
80     });
81   } else {
82     res.status(400).send('There is no username, email or password');
83   }
84 })
```

3. Frontend

A frontend kommunikál a backenddel:

A frontend persze kommunikál a szerverrel. Az API endpointokat a servicekben hívom meg metódusokba, így ezeket felhasználva a különböző komponensekbe és kommunikálva a szerverrel működik a webapp. CORS-t használók egyébként a backenden, de csak az elején kellett hogy a fellőtt 4200-as portú fronted tudjon kommunikálni a 3000-es portú backendel, de mivel most már statikusan van hostolva ezért nem is kellene már. Egy ilyen api hívás:

```
1 export const environment = {
2   production: false,
3   custom: 'This is PRF dev',
4   url: 'http://localhost:3000/',
5 }
```

```
register(usernames: string, email: string, password: string){
  return this.http.post(environment.url + 'user', { username: usernames, email: email, password: password },
    {responseType: 'text'});
}
```

A frontend komponensei route-okkal érhetőek el, a navigáció megfelelően működik:

A route-okat elég volt az `app.routing.module.ts`-be kiszervezni, most erre a projektre nem láttam értelmét még a lazy loadingos megvalósításnak is. Szimplán így valósítottam meg:

```
22
23 const routes: Routes = [
24   {path: '', redirectTo: 'home', pathMatch: 'full'},
25   {path: 'login', component: LoginComponent},
26   {path: 'register', component: RegisterComponent},
27
28   //Teams and players
29   {path: 'home', component: HomeComponent, canActivate: [AuthGuardGuard]},
30   {path: 'players/:id', component: PlayersComponent, canActivate: [AuthGuardGuard]},
31   //Admin list
32   {path: 'admin-players', component: PlayerListComponent, canActivate: [AuthGuardGuard]},
33   {path: 'admin-teams', component: TeamListComponent, canActivate: [AuthGuardGuard]},
34   {path: 'admin-users', component: UserListComponent, canActivate: [AuthGuardGuard]},
35   //Admin add
36   {path: 'admin-players-add', component: PlayerAddComponent, canActivate: [AuthGuardGuard]},
37   {path: 'admin-teams-add', component: TeamAddComponent, canActivate: [AuthGuardGuard]},
38   {path: 'admin-users-add', component: UserAddComponent, canActivate: [AuthGuardGuard]},
39
40   //Admin update
41   {path: 'admin-players-update/:id', component: PlayerUpdateComponent, canActivate: [AuthGuardGuard]},
42   {path: 'admin-teams-update/:id', component: TeamUpdateComponent, canActivate: [AuthGuardGuard]},
43   {path: 'admin-users-update/:id', component: UserUpdateComponent, canActivate: [AuthGuardGuard]},
44
45   //Details
46   {path: 'teamdetails/:id', component: TeamDetailsComponent, canActivate: [AuthGuardGuard]},
47   {path: 'playerdetails/:teamid/:playerid', component: PlayerDetailsComponent, canActivate: [AuthGuardGuard]},
48   {path: 'profile', component: ProfileComponent, canActivate: [AuthGuardGuard]},
49
50   //Error 404
51   {path: '**', component: ErrorComponent}
52 ]
```

A navbarban (és máshol is) pedig a Router-ral navigálok a page-ek között:

Egy példa a navbar komponensnél:

```
41
42 protected goToAdminTeams() {
43   this.router.navigate(['/admin-teams']);
44 }
45
46 protected goToAdminPlayers() {
47   this.router.navigate(['/admin-players']);
48 }
49
50 protected goToAdminUsers() {
51   this.router.navigate(['/admin-users']);
52 }
53
54 protected goToHome() {
55   this.router.navigate(['/home']);
56 }
57
58 protected goToProfile() {
59   this.router.navigate(['/profile']);
60 }
61
```

A frontend rendelkezik legalább egy regisztráció, egy login, egy főoldal/terméklista, egy admin felület, egy termék részletező és egy egyéb komponenssel, melyek fel vannak töltve megfelelő tartalommal:

Ezt a pontot a 4. pontban fogom bemutatni, ahol minden egyes page-ről van egy képernyőkép.

A frontend a bejelentkezéshez a backend megfelelő végpontjait szólítja meg:

A login.service.ts-ben találhatóak a login/logout servicek, amik a szerver API endpointjait hívják meg:

```
9  export class LoginService {
10
11      constructor(private http: HttpClient) { }
12
13      login(usernames: string, password: string) {
14          return this.http.post(environment.url + 'login', { username: usernames, password: password },
15              { responseType: 'text' });
16      }
17
18
19      logout() {
20          return this.http.post(environment.url + 'logout', {}, { withCredentials: true, responseType: 'text' }, )
21      }
22  }
```

Majd a login komponensbe ezt felhasználom:

```
20
21  public ngOnInit() {
22      if(localStorage.getItem('user')) {
23          localStorage.removeItem('user');
24          this.loginService.logout().subscribe(msg => {
25              console.log(msg);
26          }, error => {
27              console.log(error);
28          }
29      );
30  }
31  }
32
33  protected login() {
34      if (this.username.trim() === '' || this.password.trim() === '') {
35          this.errorMessage = 'Minden mezőt tölts ki!';
36          return;
37      }
38
39      this.loginService.login(this.username, this.password).subscribe(msg => {
40          console.log(msg);
41          localStorage.setItem('user', this.username);
42
43          this.router.navigate(['/home']);
44      }, error => {
45          console.log(error);
46          this.errorMessage = 'Hibás felhasználónév vagy jelszó';
47      });
48  }
49  }
```


A backenddel való kommunikáció elemei ki vannak szervezve service-ekbe:

Mint ahogy azt már írtam, minden egyes API endpoint hívás ki van szervezve egységesen service-ekbe.

User:

```
11 private readonly apiUrl = 'http://localhost:3000/user';
12
13 private readonly apiUrlAdminUserAdd = 'http://localhost:3000/useradd';
14
15
16 constructor(private http: HttpClient) { }
17
18 public getUsers(): Observable<User[]> {
19     return this.http.get<User[]>(this.apiUrl);
20 }
21
22 public createUser(user: User): Observable<User> {
23     return this.http.post<User>(this.apiUrlAdminUserAdd, user);
24 }
25
26 // PUT API to update a player
27 public updateUser(username: string, user: User) {
28     //return this.http.put(`/api/players/${playername}`, player);
29     return this.http.put(`${this.apiUrl}/${username}`, user);
30 }
31
32 public deleteUser(username: string): Observable<User> {
33     return this.http.delete<User>(`${this.apiUrl}/${username}`);
34 }
35
36 public getUser(username: string): Observable<User> {
37     //return this.http.get<User>(`/api/users/${username}`);
38     return this.http.get<User>(`${this.apiUrl}/${username}`);
39 }
```

Team:

```
11 private url = 'http://localhost:3000/team';
12
13 constructor(private http: HttpClient) { }
14
15 getTeams(): Observable<Team[]> {
16     return this.http.get<Team[]>(this.url);
17 }
18
19 createTeam(team: Team): Observable<any> {
20     return this.http.post<any>(this.url, team);
21 }
22
23 // PUT API to update a player
24 updateTeam(teamname: string, team: Team) {
25     //return this.http.put(`/api/players/${playername}`, player);
26     return this.http.put(`${this.url}/${teamname}`, team);
27 }
28
29 deleteTeam(teamname: string): Observable<any> {
30     return this.http.delete<Team>(`${this.url}/${teamname}`);
31 }
32
33 getTeam(teamname: string): Observable<Team> {
34     return this.http.get<Team>(`${this.url}/${teamname}`);
35 }
36
37 }
```

Player:

```
15
16  getPlayers(teamname: string): Observable<Player[]> {
17      return this.http.get<Player[]>(`${this.url}/player/${teamname}`);
18  }
19
20  // POST API to add a player
21  addPlayer(player: Player) {
22      //return this.http.post('/api/players', player);
23      return this.http.post(`${this.url}/players/`,player);
24  }
25  }
26
27  // PUT API to update a player
28  updatePlayer(playername: string, player: Player) {
29      //return this.http.put(`/api/players/${playername}`, player);
30      return this.http.put(`${this.url}/players/${playername}`,player);
31  }
32
33  // GET API to list all players
34  getAllPlayers(): Observable<Player[]> {
35      return this.http.get<Player[]>(`${this.url}/players`);
36  }
37
38  // GET API to list players with a certain teamname
39  getPlayersByTeam(teamname: string) {
40      return this.http.get(`/api/players/${teamname}`);
41  }
42
43  // DELETE API to remove a player
44  public deletePlayer(playername: string): Observable<Player> {
45      return this.http.delete<Player>(`${this.url}/players/${playername}`);
46  }
47
48  getPlayer(playername: string): Observable<Player> {
49      return this.http.get<Player>(`${this.url}/playerdetails/${playername}`);
50  }
51  }
```

Register:

```
14
15  register(usernames: string, email: string, password: string){
16      return this.http.post(environment.url + 'user', { username: usernames, email: email, password: password },
17          {responseType: 'text'});
18  }
19  }
```

Login:

```
12
13  login(usernames: string, password: string) {
14      return this.http.post(environment.url + 'login', { username: usernames, password: password },
15          {responseType: 'text'});
16  }
17  }
18
19  loginout() {
20      return this.http.post(environment.url + 'logout', {}, {withCredentials: true, responseType: 'text'}, )
21  }
22  }
```

Van authguard, amely védi a login, register utáni route-okat és az admin felületét:

A webapp rendelkezik authguardal is ami megakadályozza, hogy a user úgy léphessen a pagek-re, hogy nincsen bejelentkezve.

Az auth.guard.ts-ben van megvalósítva ez:

```
9
10 constructor(private router: Router) {
11 }
12 canActivate(
13   route: ActivatedRouteSnapshot,
14   state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
15
16   if(localStorage.getItem('user'))
17   {
18     return true;
19   } else {
20     this.router.navigate(['/login']);
21     return false;
22   }
23 }
24
25 }
26
```

Egy route-hoz pedig így kell hozzáadni:

```
//Teams and players
{path: 'home', component: HomeComponent, canActivate: [AuthGuardGuard]},
{path: 'players/:id', component: PlayersComponent, canActivate: [AuthGuardGuard]},
//Admin list
```

4. Képernyőképek a webapról:


A webapp nem rendelkezik a legjobb css-el de mivel nem is ez volt a kurzus lényege ezért egyszerűbbre fogtam ezt.

- Login:

Jelentkezz be

Felhasználónév

Jelszó

 Bejelentkezés  Regisztráció



- Register:

Regisztrálj be

Felhasználónév

Email


Jelszó

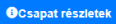
 Regisztráció  Bejelentkezés

- Home page (csapatok)


Csapatok


Barcelona

 Játékosok

 Csapat részletek

Manchester United

 Játékosok


 Csapat részletek

Manchester City


PSG

- Player (adott csapat játékosai)

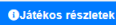
A Barcelona játékosai

 Back

Ansu Fati

 Játékos részletek


Ronald Araujo

 Játékos részletek

Pedri

Gavi


- Játékos részletek


 Főoldal

Csapat módosítás


Játékos módosítás


Felhasználó módosítás








Ansu Fati részletei


 Játékos neve: Ansu Fati

 Érték: 50000 millió dollár

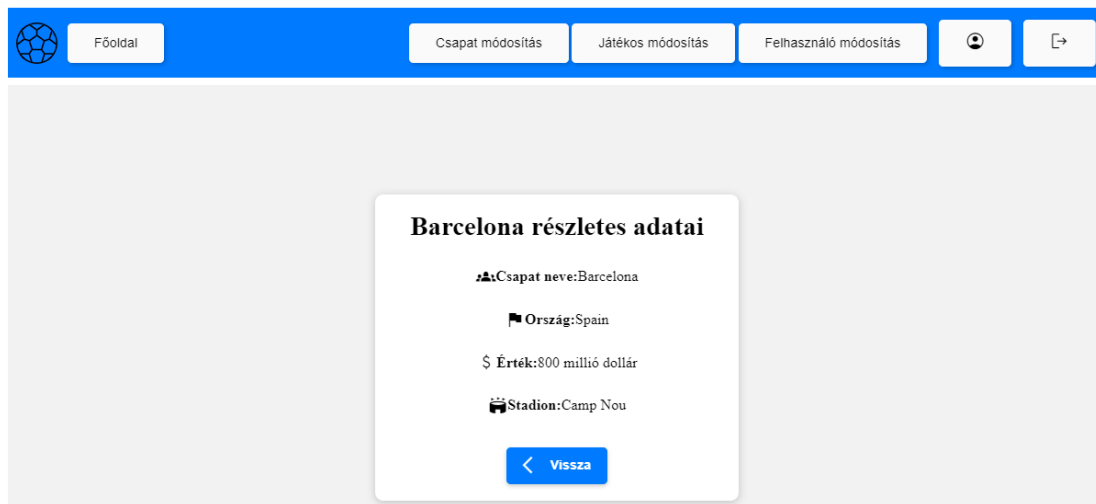
 Gólok: 200

 Nemezetiség: Spain

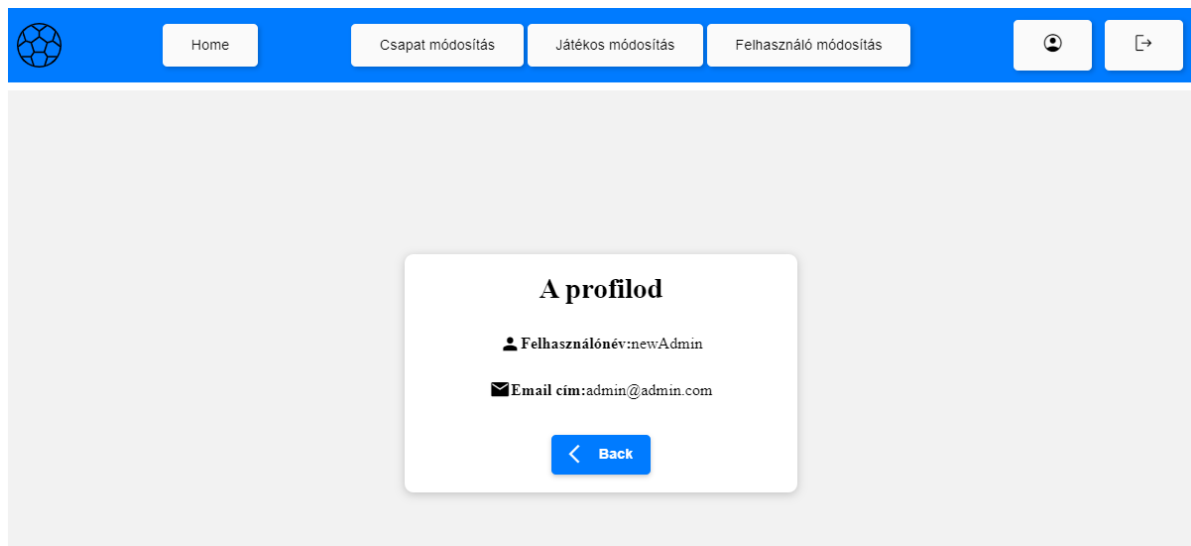
 Csapat: Barcelona

 Back

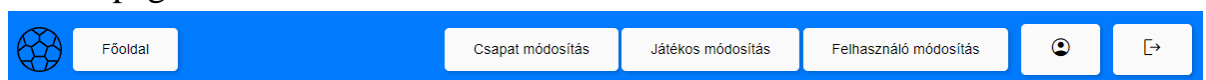
- Csapat részletek



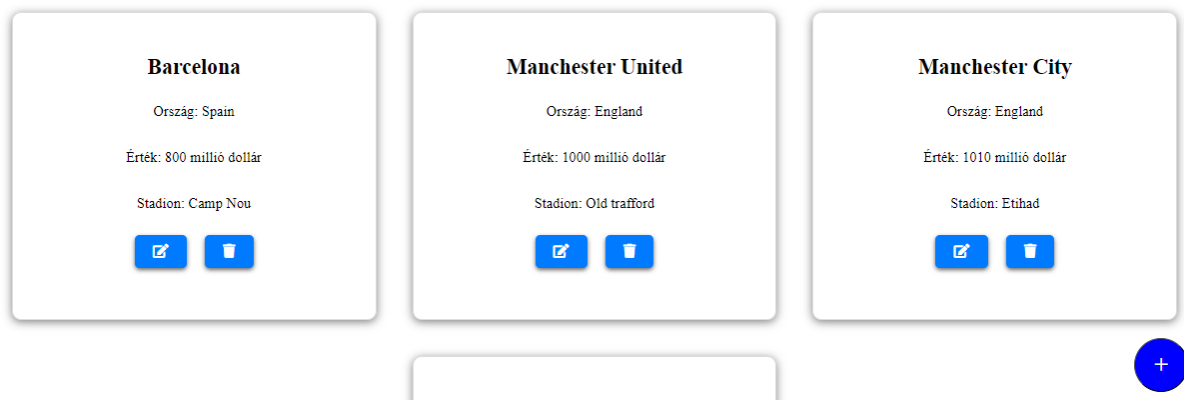
- Profile



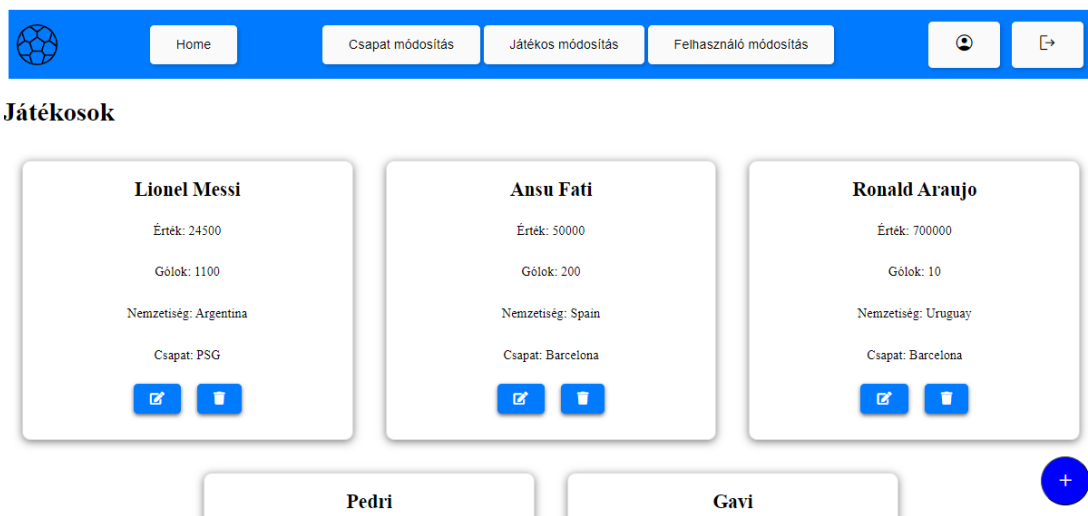
- Admin page – Team list



Csapatok

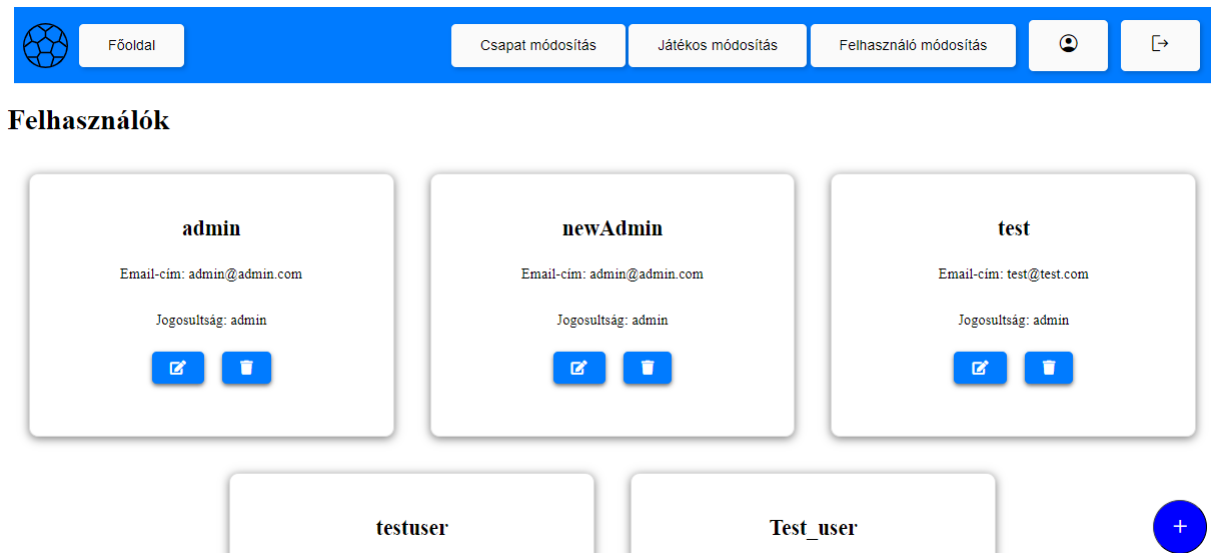


- Admin page – Player list



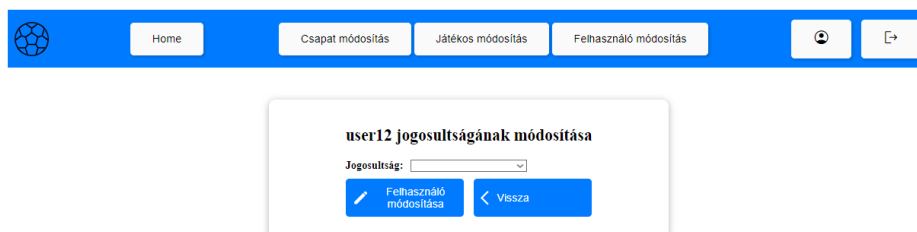
The interface shows a blue header bar with a soccer ball icon, a 'Home' button, and three buttons: 'Csapat módosítás', 'Játékos módosítás', and 'Felhasználó módosítás'. On the right are user and list icons. Below the header, the title 'Játékosok' is displayed. The main area contains three player cards: Lionel Messi (Érték: 24500, Gólok: 1100, Nemzetiség: Argentina, Csapat: PSG), Ansu Fati (Érték: 50000, Gólok: 200, Nemzetiség: Spain, Csapat: Barcelona), and Ronald Araujo (Érték: 700000, Gólok: 10, Nemzetiség: Uruguay, Csapat: Barcelona). Each card has edit and delete icons. Below these are partial cards for 'Pedri' and 'Gavi', and a blue circular button with a '+' sign.

- Admin page – User list (with cool usernames)



The interface shows a blue header bar with a soccer ball icon, a 'Főoldal' button, and three buttons: 'Csapat módosítás', 'Játékos módosítás', and 'Felhasználó módosítás'. On the right are user and list icons. Below the header, the title 'Felhasználók' is displayed. The main area contains three user cards: 'admin' (Email-cím: admin@admin.com, Jogosultság: admin), 'newAdmin' (Email-cím: admin@admin.com, Jogosultság: admin), and 'test' (Email-cím: test@test.com, Jogosultság: admin). Each card has edit and delete icons. Below these are partial cards for 'testuser' and 'Test_user', and a blue circular button with a '+' sign.

- User edit



The interface shows a blue header bar with a soccer ball icon, a 'Home' button, and three buttons: 'Csapat módosítás', 'Játékos módosítás', and 'Felhasználó módosítás'. On the right are user and list icons. Below the header, a modal window titled 'user12 jogosultságának módosítása' is displayed. It contains a 'Jogosultság:' dropdown menu and two buttons: 'Felhasználó módosítása' and 'Vissza'.

- Player edit

Ansu Fati módosítása

Érték:

Gólok:

Ország:

Csapat név:

 Játékos módosítása

 Vissza


- Team edit


Barcelona módosítása

Ország:

Csapat értéke:

Stadion:

 Csapat módosítása

 Vissza

- Team add


Csapat hozzáadása


Csapatnév:

Ország:

Csapat értéke:

Stadion:

 Csapat hozzáadása

 Vissza

- Player add

Főoldal

Csapat módosítás

Játékos módosítás

Felhasználó módosítás

Játékos hozzáadása

Játékos név:

Érték:

0

Goalok száma:

0

Nemzetiség:

Csapat név:

Játékos hozzáadása

Vissza

- User add

Home

Csapat módosítás

Játékos módosítás

Felhasználó módosítás

Felhasználó hozzáadása

Felhasználónév:

Jelszó:

Email:

Jogosultsági szint:

Jogosultsági szint

Felhasználó hozzáadása

Vissza

- Delete

Főoldal

Csapat módosítás

Játékos módosítás

Felhasználó módosítás

Játékosok

Lionel Messi

Érték: 24500

Gólok: 1100

Nemzetiség: Argentina

Csapat: PSG

Ansu Fati

Törlés megerősítés

Biztos törölni akarsz Ansu Fati-t ?

Törölés Vissza

Csapat: Barcelona

Ronald Araujo

Érték: 700000

Gólok: 10

Nemzetiség: Uruguay

Csapat: Barcelona

Pedri

Gavi

Rashford

