PRF Beadandó
Fejlesztési napló

A projekt alapját a gyakorlati videók és a forráskódok alapján raktam össze. A projektet az Atlas felhőben hostoltam, bármilyen IP címről elérhető. Az adatbázisban két tábla van, az egyik a users tábla a másik a product tábla. A projektet a Github-ra töltöttem fel, oda is két mappát, az egyikben maga a projekt van, a másikba pedig a frontendhez tartozó kódokat töltöttem fel.

A projekt a következőképpen épül fel:
Van egy bejelentkezési és egy regisztrációs felület, ezek után a felhasználó a home page-re kerül ahol listában fel vannak sorolva az adatbázisban található termékek. A sima(nincs admin jogosultsága) felhasználó a listaelemekre kattintva megnézheti a termékeket. Admin esetében ez a felület annyiban változik, hogy a termék törölhetővé válik. A sima felhasználó a termékek megtekintése mellett, még kijelentkezni tud. Admin esetében elérhetővé válik két másik oldal, az egyik az admin oldal ahol más felhasználók jogosultságait tudja megváltoztatni. Illetve még egy oldal ahol új termékeket vehet fel az adatbáziba. Regisztráció után egy sima felhasználó jön létre, viszont a **felhasználónév: admin, jelszó: admin123** párossal be lehet lépni, mint admin jogosultságú felhasználó. Az admin nevű felhasználó jogosultsága nem változtatható meg.

A futtatáshoz :

- beadando nevű mappa megnyitása VS Code-ban
- npm install parancs kiadása
- index.js fájlban át kell írni a const dbUrl nevű változó értékét a beadási felületen lévőre

```
const dbUrl = ""; // ide kell az URL
```

- node index vagy nodemon index parancs kiadása
- böngészőben localhost:3000 címen elérhetővé válik a projekt

A továbbiakban pedig a követelmények alapján mutatom meg a kódot.

Backend (45p):

1. **A backend statikusan hostolja a frontendet** (5p)

index.js - 77.sor

```
/* Az express.static() metódusban meg kell adnunk azt a mappát,
amelyből a statikus fájlokat kiszolgáljuk. */
app
  .use(express.static(path.join(__dirname, "public")))
  .set("views", path.join(__dirname, "views"))
  .set("view engine", "ejs")
  .get("/", (req, res) => res.render("pages/index"));
```

2. **Az alkalmazás kapcsolódik egy mongodb instance-hoz** (2,5p)

inde.js - 16.sor

```
const mongoose = require("mongoose");
const app = express();
const port = process.env.PORT || 3000;
const dbUrl = ""; // ide kell az URL

mongoose.connect(dbUrl);

mongoose.connection.on("connected", () => {
  console.log("db csatlakoztatva");
  require("./bootstrapper")();
});

mongoose.connection.on("error", (err) => {
  console.log("Hiba tortent", err);
});
```

3. **Az alkalmazás képes bootstrappelni, vagyis MongoDB-t alap userekkel feltölteni** (5p)

bootsrapper.js, index.js - 20.sor

```
const mongoose = require("mongoose");
const User = require("./user.model");
const userModel = mongoose.model("user");
// Létrehoz egy admin felhasználót
async function ensureAdminExists() {
  try {
    const admin = await userModel.findOne({ admin: true });
    if (admin) {
            console.log("Az admin felhasználó már megtalálható az
adatbázisban!");
    } else {
      const newAdmin = new userModel({
```

```
        username: "admin",
        password: "admin123",
        admin: true,
        email: "admin@admin.hu",
      });
      await newAdmin.save();
      console.log("Az admin felhasználó sikeresen létrehozva!");
    }
  } catch (error) {
    console.error(
      "Hiba történt az admin ellenőrzése vagy létrehozása során: ",
      error
    );
  }
}
// Létrehoz két sima felhasználót
async function ensureUserExists() {
  try {
    const user = await userModel.findOne({ username: "user" });
    if (user) {
      console.log("A felhasználó már megtalálható az adatbázisban!");
    } else {
      const newUser = new userModel({
        username: "user",
        password: "1234",
        email: "user@freemail.hu",
        admin: false,
      });
      await newUser.save();
      console.log("A felhasználó sikeresen létrehozva!");
    }
  } catch (error) {
    console.error(
       "Hiba történt az felhasználó ellenőrzése vagy létrehozása során:
",
      error
    );
  }
}

module.exports = ensureAdminExists;
module.exports = ensureUserExists;
```

A bootstrapper-t csak azután hívom csak meg, miután sikerült csatlakozni az adatbázishoz.

4. A szerver megvalósít legalább két modellt, melyek sémája egyértelműen definiált (5p)

user.model.js

```
var userSchema = new mongoose.Schema(
  {
     username: { type: String, unique: true, requried: true, lowercase:
true },
    password: { type: String, required: true },
    email: { type: String, required: true },
    admin: { type: Boolean, default: false },
  },
  { collection: "users" }
);
```

product.model.js

```
var productSchema = new mongoose.Schema(
  {
     id: { type: String, unique: true, requried: true, lowercase: true
},
    name: { type: String, unique: true, required: true },
    description: { type: String, required: true },
  },
  { collection: "product" }
);
```

5. Adott legalább két olyan adatbázis hook, amelyek a modellek mentése vagy lekérése közben futnak le (5p)

```
userSchema.pre("save", function (next) {
  const user = this;
  if (user.isModified("password")) {
    bcrypt.genSalt(10, function (err, salt) {
      if (err) {
        console.log("Hiba a salt generalas soran");
        return next(err);
      }
      bcrypt.hash(user.password, salt, function (err, hash) {
        if (err) {
          console.log("Hiba a hasheles soran");
          return next(err);
        }
        user.password = hash;
        return next();
      });
    });
```

```
  } else {
    return next();
  }
});

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

6. A szerver megvalósít egy lokális autentikációs stratégiát (7,5p)

index.js

```
passport.use(
  "local",
  new localStrategy(function (username, password, done) {
    userModel.findOne({ username: username }, function (err, user) {
      // Felhasználó keresése az adatbázisban a felhasználónév alapján
      if (err) return done("Hiba lekeres soran", null);
      // Hiba kezelése az adatbázis lekérdezésekor
      if (!user) return done("Nincs ilyen felhasználónév", null);
      // Ellenőrzés, hogy létezik-e a felhasználó
      user.comparePasswords(password, function (error, isMatch) {
        // Felhasználó által megadott jelszó ellenőrzése
        if (error) return done(error, false);
              // Hiba kezelése, ha nem sikerült összehasonlítani a
jelszavakat
        if (!isMatch) return done("Hibas jelszo", false);
        // Sikeres belépés esetén felhasználó visszaadása
        return done(null, user);
      });
    });
  })
);
```

7. A szerver kezeli a login sessiont (7,5p)

routes.js

```
router.route("/login").post((req, res, next) => {
  if ((req.body.username, req.body.password)) {
    passport.authenticate("local", function (error, user) {
      if (error) return res.status(500).send(error);
      req.login(user, (error) => {
        if (error) return res.status(500).send(error);
        return res
```

```
          .status(200)
          .send({ msg: "Bejelentkezes sikeres", isAdmin: user.admin });
      });
    })(req, res, next);
  } else {
      return res.status(400).send("Hibas keres, username es password
kell");
  }
});

router.route("/logout").post((req, res, next) => {
  if (req.isAuthenticated()) {
    req.logout((err) => {
      if (err) {
        console.log("Hiba a kijelentkezés során");
        return res.status(500).send(err);
      }
      return res.status(200).send("Kijelentkezes sikeres");
    });
  } else {
    return res.status(403).send("Nem is volt bejelentkezve");
  }
});
```

8. **A szerver rendelkezik a két kezelt modell CRUD interfészeivel, illetve egy login, logout, register route-tal** (7,5p)

routes.js - elég hosszú azt most nem másolom be ide

Frontend (45p) *frontend nevű mappában megtalálhatóak ezek a kódok*

1. A frontend kommunikál a backenddel (6p)

src/utils/login.service.ts + src/utils/products.service.ts

Mindkettőből bemásolok ide 1-1 hívást.

```
makeAdmin(username: string) {
    return this.http.put(
      environment.serverUrl + '/user/makeAdmin',
      { username: username },
      { responseType: 'text' }
    );
  }
```

```
create(id: string, name: string, description: string) {
    return this.http.post(
      environment.serverUrl + '/product',
      { id: id, name: name, description: description },
      { responseType: 'text' }
    );
  }
```

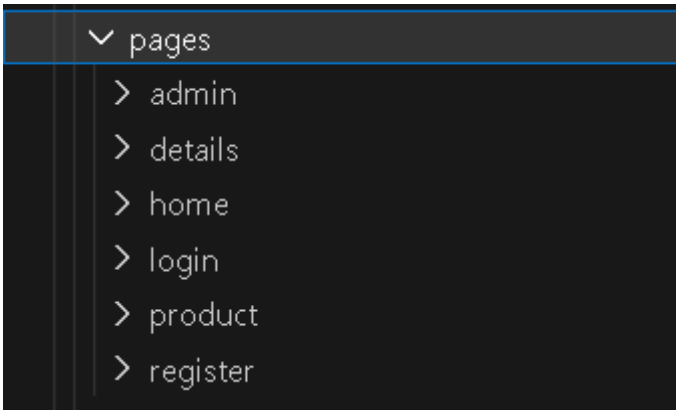
2. A frontend komponensei route-okkal érhetőek el, a navigáció megfelelően működik (6p)

app-routing.module.ts

```
const routes: Routes = [
  { path: '', redirectTo: 'login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'home', component: HomeComponent, canActivate: [AuthGuard] },
  {
    path: 'admin',
    component: AdminComponent,
    canActivate: [AuthGuard, RoleGuard],
  },
  {
    path: 'details/:id',
    component: DetailsComponent,
    canActivate: [AuthGuard],
  },
  {
    path: 'product',
    component: ProductComponent,
    canActivate: [AuthGuard, RoleGuard],
  },
  { path: '**', component: ErrorComponent },
```

```
];
```

3. **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** (12p)



4. **A frontend a bejelentkezéshez a backend megfelelő végpontjait szólítja meg** (6p)

login.servide.ts

```
login(username: string, password: string) {
    return this.http.post(
      environment.serverUrl + '/login',
      { username: username, password: password },
      { withCredentials: true, responseType: 'json' }
    );
  }

  logout() {
    return this.http.post(
      environment.serverUrl + '/logout',
      {},
      { withCredentials: true, responseType: 'text' }
    );
  }
```

5. **A backenddel való kommunikáció elemei ki vannak szervezve service-ekbe** (6p)

src/utils/login.service.ts + src/utils/products.service.ts

6. **Van authguard, amely védi a login, register utáni route-okat és az admin felületét** (9p)

src/guards/role.guard.ts + src/guards/auth.guard.ts

```
canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ):
    | Observable<boolean | UrlTree>
    | Promise<boolean | UrlTree>
    | boolean
```

```
    | UrlTree {
    if (localStorage.getItem('user')) {
      return true;
    } else {
      this.router.navigate(['/login']);
      return false;
    }
  }
```

```
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ):
    | Observable<boolean | UrlTree>
    | Promise<boolean | UrlTree>
    | boolean
    | UrlTree {
      if (localStorage.getItem('isAdmin') === 'true') {
        return true;
      } else {
        this.router.navigate(['/home']);
        return false;
      }
  }
```

Dokumentáció

Az adatbázis hívásoknál van console.log(), error és a catch ágak is le vannak kezelve (ugye ezeket böngészőből a F12 megnyomásával nézhetőek meg).

Illetve az adatbázis csatlakozásánál és a bootstrap lefutása után is kapunk logokat: