

**D O K U M E N T Á C I Ó**

# Vizsgaremek

**SZOFTVERFEJLESZTŐ ÉS -TESZTELŐ TECHNIKUS SZAKMA**

Tartalom

[Vizsgaremek 1](#_Toc187956691)

[Bevezetés 3](#_Toc187956692)

[Fejlesztői dokumentáció 3](#_Toc187956693)

[Felhasznált technológiák 3](#_Toc187956694)

[Fejlesztőkörnyezet 4](#_Toc187956695)

[Kialakított adatszerkezet 4](#_Toc187956696)

[Algoritmusok / kódok bemutatása 12](#_Toc187956697)

[Miért használunk mappereket? 23](#_Toc187956698)

[Fejlesztési lehetőségek 28](#_Toc187956699)

[Felhasználói dokumentáció 30](#_Toc187956700)

[Kezdő képernyő (bejelentkezés) 31](#_Toc187956701)

[Regisztrációs felület 32](#_Toc187956702)

[Főoldal és navigáció 33](#_Toc187956703)

[Saját profil 34](#_Toc187956704)

[Felhasználók 35](#_Toc187956705)

[Szemtorna 35](#_Toc187956706)

[Top 10-es lista 37](#_Toc187956707)

[Tesztelés 37](#_Toc187956708)

[Automatizált 37](#_Toc187956709)

[Manuális 38](#_Toc187956710)

[Összefoglalás 39](#_Toc187956711)

## Bevezetés

A modern életmód, amelyet a képernyők előtti hosszas időtöltés és a szemek túlzott igénybevétele jellemez, egyre több embert ösztönöz arra, hogy megoldásokat keressenek szemük egészségének megőrzésére. A szemtornát gyakoroltató program megalkotásának alapgondolata abból a felismerésből fakad, hogy a szem izmainak rendszeres mozgatása és erősítése hozzájárulhat a látás egészségének fenntartásához és javításához. A szemtornagyakorlatok nemcsak a szemizmokat erősítik, hanem elősegíthetik a szem fáradtságának csökkentését és a látásminőség javítását is.

A témaválasztás ötlete a csapat egyik tagjától, Vicától származik, aki jelenleg Leo Angard "Szerezd vissza éleslátásodat" című könyvét olvassa, amely a szemizmok tornáztatásáról, a lehetséges gyakorlatokról és azok hatásairól szól. Ez a könyv részletesen ismerteti, hogy milyen módszerekkel lehet a szem egészségét támogatni és milyen előnyöket nyújthat a rendszeres gyakorlás. A könyv inspirálta a csapatot arra, hogy egy olyan applikációt hozzanak létre, amely segíti az embereket abban, hogy ezeket a gyakorlatokat könnyen elsajátítsák és napi rutinjuk részévé tegyék.

Az applikáció koncepciójának egyik különlegessége, hogy kompetitív elemeket is tartalmaz, amelyek arra ösztönzik a felhasználókat, hogy minél több időt töltsenek a gyakorlással. A rendszer az elvégzett gyakorlatok mennyisége és hossza alapján pontokat ad a felhasználóknak, amelyekkel különböző matricákat gyűjthetnek. Ezek a matricák nemcsak motivációként szolgálnak, hanem lehetőséget biztosítanak arra is, hogy a felhasználók összehasonlítsák teljesítményüket egy online ranglistán. Az ilyen játékos elemek segítenek abban, hogy a felhasználók elkötelezettebbé váljanak, és szívesebben végezzék el a gyakorlatokat.

A program célja, hogy a szemtornázás ne csak egyszerű és könnyen hozzáférhető legyen, hanem szórakoztató és motiváló is, ezzel elősegítve a rendszerességet. Az applikáció egyaránt nyújt elméleti információkat a gyakorlatok hátteréről és gyakorlati útmutatást a helyes kivitelezéshez, így komplex támogatást nyújt a felhasználók számára.

## Fejlesztői dokumentáció

### Felhasznált technológiák

Az applikáció fejlesztése során az alábbi technológiákat használtuk:

* **Java 17**: A legújabb LTS (Long Term Support) verzió a nyelv fejlett funkcióinak kihasználására a stabil és folyamatos biztonsági frissítések mellett.
* **Spring Boot**: Gyors fejlesztést és könnyű konfigurációs lehetőségeket biztosít a backend fejlesztő számára.
* **PostgreSQL**: Megbízható és skálázható adatbázis-kezelő rendszer.
* **Docker**: Egy soros scripttel inicializálható egy új adatbázis szerver a használatával, ebben a projektben csak emiatt kapott szerepet. Egy konténerben fut a Postgres adatbázis szerver.
* **Angular**: Modern frontend framework a reszponzív felhasználói felület megvalósítására.
* **Angular Materials**: Felhasználói élmény javítása modern vizuális elemekkel.
* **MapStruct**: Entity->Dto és Dto->Entity konverziók automatizált használatához.
* **Spring Security**: Az alkalmazás védelme hitelesítési és autorizációs funkciókkal.
* **JPA Hibernate**: Adatbázis-műveletek egyszerűsítésére.
* **Liquibase**: Adatbázis inicializálásához és folyamatos verziókezeléséhez.

### Fejlesztőkörnyezet

#### Hardver ismertetése és választás indoklása

A fejlesztéshez a következő hardvereket használtuk:

* **Laptop**: Intel i5 vagy i7 processzorral, 16-32 GB RAM-mal és SSD tárolóval. Ezeket az eszközöket a gyakorlati helyszín biztosította számunkra.

#### Szoftverek ismertetése és választás indoklása

* **Operációs rendszer**: Windows, amely kompatibilitása és a céges infrastruktúra támogatottsága miatt került kiválasztásra.
* **IntelliJ IDEA**: A fejlesztés során ezt az integrált fejlesztőkörnyezetet használtuk, mivel hatékony eszközöket biztosít mind Java, mind Angular alkalmazások fejlesztéséhez. Integráltan kezeli mind a server, mind az adatbázis, mind a docker konténer illetve a fejlesztés során használt minden egyéb eszköz kezelő felületét.

### Kialakított adatszerkezet

#### Adatbázis

Az alkalmazás háttéradatbázisa a népszerű PostgreSQL rendszerre épül, amely széles körben elterjedt az üzleti alkalmazások és webfejlesztés területén. Az adatbázis létrehozásához a modern konténerizációs technológia, a Docker volt segítségünkre, amely megkönnyíti a fejlesztési és üzemeltetési folyamatokat. Az adatbázis létrehozását és konfigurálását az alábbiakban bemutatott docker-compose.yml fájl segítségével valósítottuk meg, amely tartalmazza a szükséges beállításokat és a szolgáltatások automatizált indítását.

#### Adatmodell UML diagram

1. Teljes adatmodell UML diagram

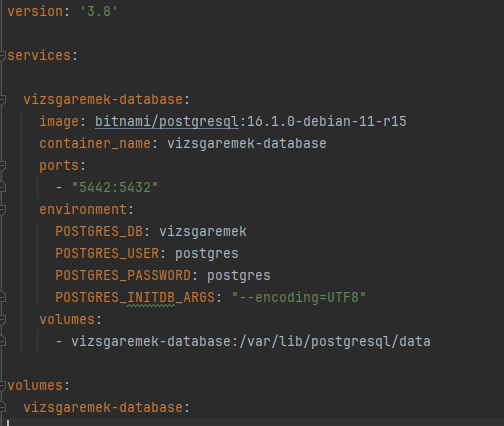
Megjegyzés:

A Liquibase automatikusan létrehozza a **DATABASECHANGELOG** és **DATABASECHANGELOGLOCK** táblákat, amelyeket az adatbázis sémaváltozások verziókövetésére használ. A **DATABASECHANGELOG** tábla tartalmazza az összes futtatott változtatás részleteit, például a változásazonosítót (*id*), a szerző nevét (*author*), a fájl nevét, és a futtatás dátumát. Ez lehetővé teszi, hogy a Liquibase nyomon kövesse, melyik változtatásokat alkalmazták már az adatbázison, és elkerülje az ismételt végrehajtást. A **DATABASECHANGELOGLOCK** tábla biztosítja, hogy egyszerre csak egy Liquibase példány hajtson végre módosításokat az adatbázison. Ezt egy zárolási mechanizmussal valósítja meg: a tábla egyetlen sorban rögzíti, hogy az adatbázis változtatásai éppen zárolva vannak-e, ezzel megelőzve a párhuzamos futásból eredő ütközéseket. Ez a két tábla kulcsfontosságú az adatbázis biztonságos és következetes frissítéséhez.

A további táblák részletesen bemutatásra kerülnek a [A Liquibase által kezelt táblák és azok működése](#_A_Liquibase_által) szakaszban.

#### Docker konténer működése

A Docker egy konténerizációs platform, amely lehetővé teszi az alkalmazások és azok futtatásához szükséges összetevők becsomagolását egy egységbe. Az adatbázis konténerizálásával biztosítottuk az alkalmazás fejlesztési és tesztelési környezetének konzisztenciáját.

A Docker Compose eszköz segítségével definiáltuk az adatbázis-szolgáltatást a következő beállításokkal:

1.ábra docker-compose.yml

#### Működés

1. **Image alapú konténer létrehozás**: A bitnami/postgresql:16.1.0-debian-11-r15 Docker image használatával biztosítottuk a PostgreSQL adatbázis legújabb verzióját.
2. **Környezeti változók**: A POSTGRES\_DB, POSTGRES\_USER és POSTGRES\_PASSWORD változók meghatározzák az adatbázis nevét, a felhasználót és a jelszót. Az POSTGRES\_INITDB\_ARGS argumentum biztosítja a létrehozott adatbázis, szövegek mentésénél alkalmazott UTF-8 kódolását.
3. **Volumenek kezelése**: A vizsgaremek-database nevű volume használatával biztosítottuk, hogy az adatbázis állandó adatai a konténer újraindítása után is elérhetők maradjanak.
4. **Hálózati beállítások**: Az ábrán az 5442 külső portot összekötöttük a konténer 5432 belső portjával, amely a PostgreSQL alapértelmezett portja erre azért volt szükség, mert ezen a gépen egy másik PostgreSQL server már használta azt a portot, de alapértelmezettként ez a beállítás mind a két oldalán 5432.

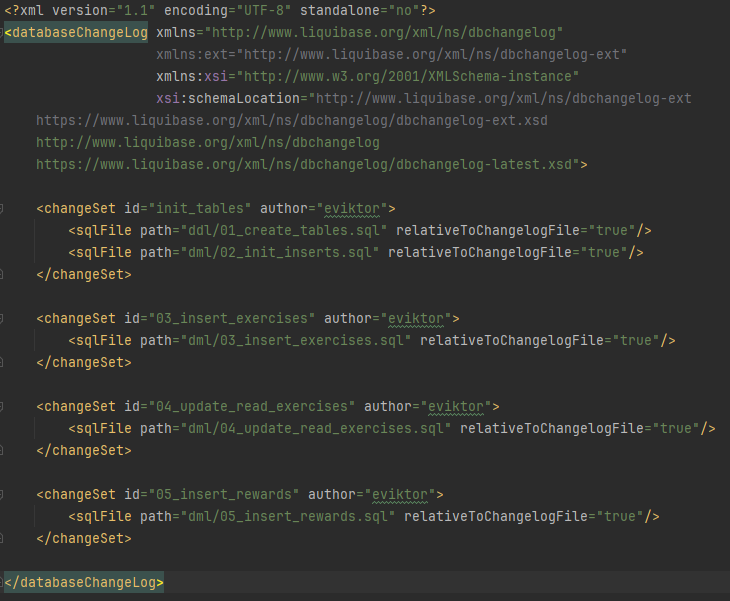
A táblák létrehozása és az inicializáló adatok betöltése az alábbi fájlok segítségével történt és a projekten belül két külön package-ba kerültek az sql scriptek viselkedésüknek megfelelően.:

* **DDL (Data Definition Language):**
  + 01\_create\_tables.sql: Tartalmazza az összes tábla létrehozására vonatkozó utasításokat.
* **DML (Data Manipulation Language):**
  + 02\_init\_inserts.sql: Az alapértelmezett adatok betöltéséhez szükséges utasításokat tartalmazza.
  + 03\_insert\_exercises.sql: A különféle szemtornagyakorlatok adatainak betöltésére szolgáló SQL utasítások.
  + 04\_update\_read\_exercises.sql: További szemtornagyakorlatok adatainak betöltésére szolgáló SQL utasítások.
  + 05\_insert\_rewards.sql: A felhasználó által elérhető plecsnik adatainak betöltésére szolgáló SQL utasítások.

#### A Liquibase használata és az adatbázis szerkezet

Az adatbázis-változások kezeléséhez a **Liquibase** eszközt használtuk. A Liquibase egy adatbázis verziókezelő eszköz, amely lehetővé teszi az adatbázis sémájának és adatainak változtatásait, valamint azok nyomon követését egy XML fájl segítségével, amely az adatbázis változásokat (changeSet) tartalmazza. Az adatbázis sémája és adatai könnyen frissíthetők, verziók kezelhetők anélkül, hogy közvetlen SQL parancsokat kellene futtatni.

#### **Liquibase és a** dbchangelog.xml **fájl**

A dbchangelog.xml fájl tartalmazza az adatbázis változtatásokat, amelyek a rendszer frissítésekor végrehajtásra kerülnek. Az XML fájlban található changeSet elemek egy-egy adatbázis változást képviselnek, amelyek tartalmazhatnak SQL fájlokat, adatbeviteleket vagy más adatbázis műveleteket.

2. ábra dbchangelog

2. A teljes Liquibase konfiguráció és a script fájlok elérhetők a projekt forráskódjában.

A fenti példában négy changeSet található, az egyik a táblák létrehozásáért (create\_tables.sql), a többi pedig az adatok inicializálásáért (init\_inserts.sql, insert\_exercises.sql, 03\_insert\_exercises.sql, 04\_update\_read\_exercises.sql, 05\_insert\_rewards.sql) felel.

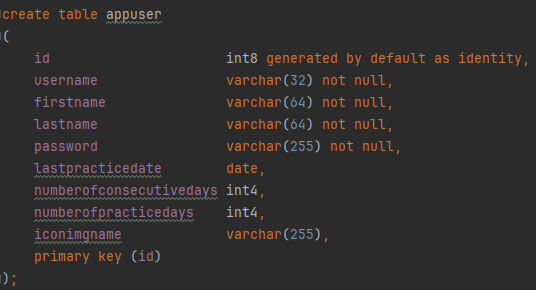
#### A Liquibase által kezelt táblák és azok működése

A fenti SQL scriptek alapján a következő táblák jönnek létre és töltődnek fel adatokkal:

##### APPUSER **tábla**

Ez a tábla az alkalmazás felhasználóinak adatait tárolja. Mezői:

* id: Azonosító (elsődleges kulcs).
* username: A felhasználó neve, kötelező mező.
* firstname, lastname: A felhasználó kereszt- és vezetékneve.
* password: A jelszó titkosítva tárolva.
* lastpracticedate: Az utolsó gyakorlat dátuma.
* numberofconsecutivedays: A következetes napok száma.
* numberofpracticedays: Az összes gyakorolt napok száma.
* iconimgname: Az ikon képfájl neve.

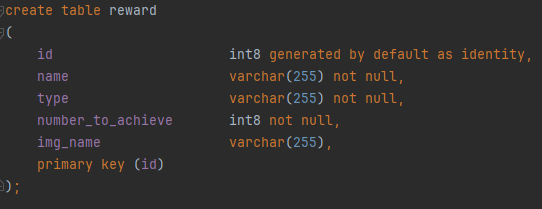
Az appuser táblához tartozó SQL script:

Ez a tábla lehetővé teszi az alapvető felhasználói adatok tárolását, mint például a felhasználónevet, a jelszót és a gyakorlatokkal kapcsolatos statisztikákat.

##### REWARD tábla

A reward tábla a felhasználók számára elérhető jutalmakat tárolja. Mezői:

* id: Azonosító (elsődleges kulcs).
* name: A jutalom neve.
* type: A jutalom típusa.
* number\_to\_achieve: A jutalom eléréséhez szükséges érték.
* img\_name: A jutalomhoz tartozó képfájl neve.

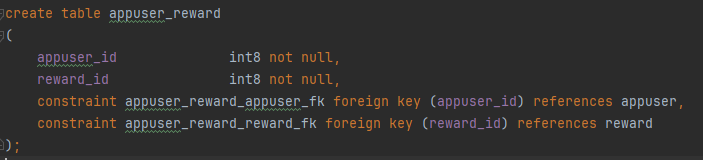
Az reward táblához tartozó SQL script

Ez a tábla tartalmazza a különféle jutalmak adatait, például a nevüket, típusukat és a szükséges feltételeket a megszerzésükhöz.

##### APPUSER\_REWARD tábla

Az appuser\_reward tábla a felhasználók és a jutalmak közötti kapcsolatot tárolja. Az oszlopai a következőket tartalmazzák:

* appuser\_id: A felhasználó ID-ja, amely hivatkozik az appuser táblára.
* reward\_id: A jutalom ID-ja, amely hivatkozik a reward táblára.

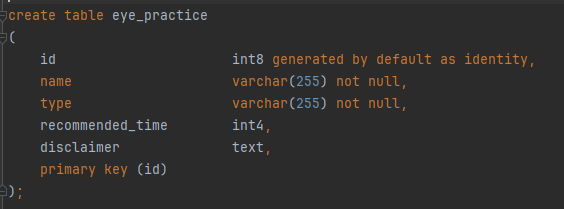
Az appuser\_reward táblához tartozó SQL script

Ez a tábla segít nyomon követni, hogy mely felhasználók mely jutalmakat kapták meg.

##### EYE\_PRACTICE tábla

Az eye\_practice tábla a különféle szemtorna gyakorlatokat tartalmazza. Mezői:

* id: Azonosító (elsődleges kulcs).
* name: A gyakorlat neve.
* type: A gyakorlat típusa (pl. torna, olvasás).
* recommended\_time: Az ajánlott időtartam.
* disclaimer: Figyelmeztetés vagy leírás a gyakorlat végzésével kapcsolatban.

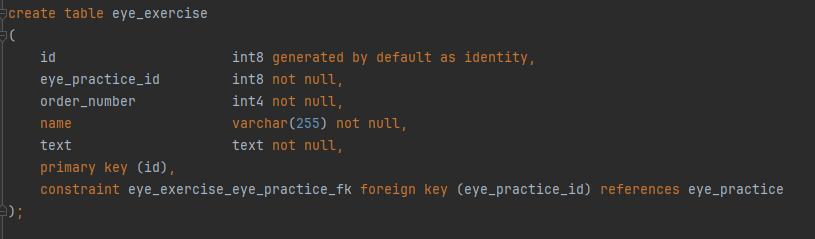
SQL scriptje:

Ez a tábla tárolja a különböző szemtorna gyakorlatok típusait és az azokhoz tartozó információkat.

##### EYE\_EXERCISES tábla

Az eye\_exercise tábla a szemgyakorlatokat tartalmazza, amelyeket a felhasználók elvégezhetnek. Mezői:

* id: Azonosító (elsődleges kulcs).
* eye\_practice\_id: Hivatkozás az eye\_practice táblára.
* order\_number: A gyakorlat sorszáma az adott csoporton belül.
* name: A gyakorlat neve.
* text: A gyakorlat leírása.

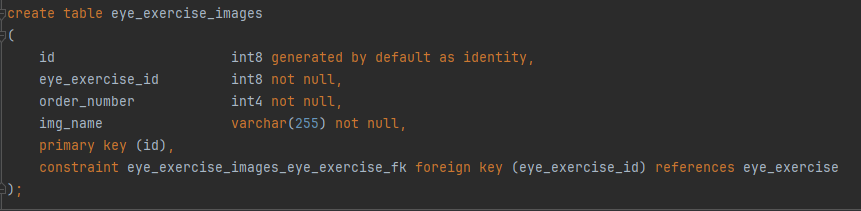
SQL scriptje:

Ez a tábla tárolja a konkrét szemgyakorlatokat, amelyeket a felhasználók végezhetnek el a szemtorna során.

##### EYE\_EXERCISE\_IMAGES tábla

Az eye\_exercise\_images tábla a szemgyakorlatokhoz tartozó képeket tárolja. Mezői:

* id: Azonosító (elsődleges kulcs).
* eye\_exercise\_id: Hivatkozás az eye\_exercise táblára.
* order\_number: A képek sorrendje a gyakorlat leírásában.
* img\_name: A képfájl neve.

SQL scriptje:

Ez a tábla tárolja a szemgyakorlatokhoz tartozó képeket, amelyek segítenek a gyakorlatok vizuális bemutatásában.

A fenti táblák és kapcsolataik létrehozása és feltöltése az SQL scriptek és a Liquibase segítségével történik. Az SQL fájlokban található parancsok végrehajtása biztosítja, hogy a rendszer megfelelő adatbázis struktúrával rendelkezzen a felhasználók és a szemtorna gyakorlatok kezelésére.

### Algoritmusok / kódok bemutatása

#### Backend

Egy tipikus Spring Boot alkalmazás rétegezett architektúrával és jól elkülönített komponensekkel rendelkezik, amelyek egyszerűen konfigurálhatók és skálázhatók. A keretrendszer rugalmasságának és automatikus konfigurációjának köszönhetően gyorsan és hatékonyan fejleszthetők modern, vállalati szintű alkalmazások

A Spring Boot egy népszerű Java keretrendszer, amely leegyszerűsíti az alkalmazásfejlesztést azáltal, hogy számos konfigurációs feladatot automatikusan elvégez. Egy tipikus Spring Boot alkalmazás több rétegből és komponensből áll, amelyek együttműködve biztosítják az alkalmazás működését. Mivel ezt a technológiát választottuk a projekt megvalósításánál így az alábbiakban részletezzük hogyan épül fel a projekt, és bemutatjuk a főbb elemeket és azok működését.

##### **Automatikus konfiguráció és Starter csomagok**

* A Spring Boot előre definiált "starter" csomagokat kínál, amelyek gyorsítják a fejlesztést.
  + Példák:
    - spring-boot-starter-web: REST API-k fejlesztéséhez.
    - spring-boot-starter-data-jpa: JPA használatához adatbázisokkal.
    - spring-boot-starter-security: Alapvető biztonsági funkciókhoz.

##### **Beépített szerver (Embedded Server)**

* A Spring Boot beépített szerverrel érkezik (pl. Tomcat, Jetty), így nincs szükség külön telepített alkalmazásszerverre.
* Az alapértelmezett szerver testreszabható az application.properties fájlban vagy konfigurációs osztályokkal.

**Példa:**

server.port=8080

##### **Hogyan működnek az annotációk?**

* A Spring Boot erőteljes annotációkra épül, amelyek deklaratív módon segítik a fejlesztést.
* Példák:
  + @Autowired: Bean-injektálásra használható.
  + @Component: Általános komponensek regisztrálása a Spring konténerbe.
  + @Transactional: Adatbázisműveletek tranzakciókezelésére.

##### **Fő komponensek és rétegek**

###### a) **Art2D2VizsgaremekApplication Class (Főindító osztály)**

* Ez az osztály tartalmazza a main metódust, amely a Spring Boot alkalmazás belépési pontja. A főosztály annotációja a @SpringBootApplication, amely három másik annotációt kombinál:
  + @Configuration: Az osztály konfigurációs osztályként működik.
  + @EnableAutoConfiguration: Engedélyezi az automatikus konfigurációt.
  + @ComponentScan: Automatikusan megkeresi és regisztrálja a komponenseket a megfelelő csomagokban.

###### b) **Controller réteg**

* A RESTful webszolgáltatásokért vagy a felhasználói kérések kezeléséért felelős réteg.
* Fő annotációk:
  + @RestController: Azt jelzi, hogy az osztály HTTP-kéréseket kezel.
  + @RequestMapping: Meghatározza az URI-k és a metódusok közötti kapcsolatot.
* A kódban implementált Controllerek:
  + AppUserController

Injektált Bean-jei:

@Autowired  
private EyePracticeService practiceService;

Metódusai:

@GetMapping("/userdata/{id}")  
public ResponseEntity<UserDto> getUserData(@PathVariable("id") Long userId) {  
 UserDto userDto = appUserService.getUserData(userId);  
 return new ResponseEntity<>(userDto, HttpStatus.*OK*);  
}  
  
@GetMapping("/top10")  
public ResponseEntity<List<UserDto>> getTop10() {  
 List<UserDto> userDto = appUserService.getTop10();  
 return new ResponseEntity<>(userDto, HttpStatus.*OK*);  
}  
  
@GetMapping  
public ResponseEntity<List<UserDto>> getAllUsers() {  
 List<UserDto> users = appUserService.getAllUsers();  
 return new ResponseEntity<>(users, HttpStatus.*OK*);  
}  
  
@PostMapping("/start-exercise")  
public void startExerciseByUser(@RequestBody StartExerciseDto startExerciseDto) {  
 appUserService.startExerciseByUser(startExerciseDto);  
}  
  
@GetMapping("/rewards/{id}")  
public ResponseEntity<Set<RewardDto>> getAllRewardsByUser(@PathVariable("id") Long userId) {  
 Set<RewardDto> rewards = appUserService.getAllRewardByUser(userId);  
 return new ResponseEntity<>(rewards, HttpStatus.*OK*);  
}

* + AuthController

Injektált Bean-jei:

@Autowired  
private AuthService authService;

Metódusai:

@PostMapping("/authenticate")  
public ResponseEntity<UserDto> authenticateUser(@RequestBody LoginDto loginDto) {  
 UserDto userDto = authService.authenticateUser(loginDto);  
 return new ResponseEntity<>(userDto, HttpStatus.*OK*);  
}  
  
@PostMapping("/register")  
public ResponseEntity<UserDto> registerUser(@RequestBody RegistrationDto registrationDto) {  
 UserDto userDto = authService.registerUser(registrationDto);  
 return new ResponseEntity<>(userDto, HttpStatus.*OK*);  
}

* + ExerciseController

Injektált Bean-jei:

private EyeExerciseService eyeExerciseService;

Metódusa:

@GetMapping("/list/{eye-practice-id}")  
public ResponseEntity<List<EyeExerciseDto>> getAllEyeExerciseByEyePracticeId(@PathVariable("eye-practice-id") Long eyePracticeId) {  
 List<EyeExerciseDto> list = eyeExerciseService.getAllEyeExerciseByEyePracticeId(eyePracticeId);  
 return new ResponseEntity<>(list, HttpStatus.*OK*);  
}

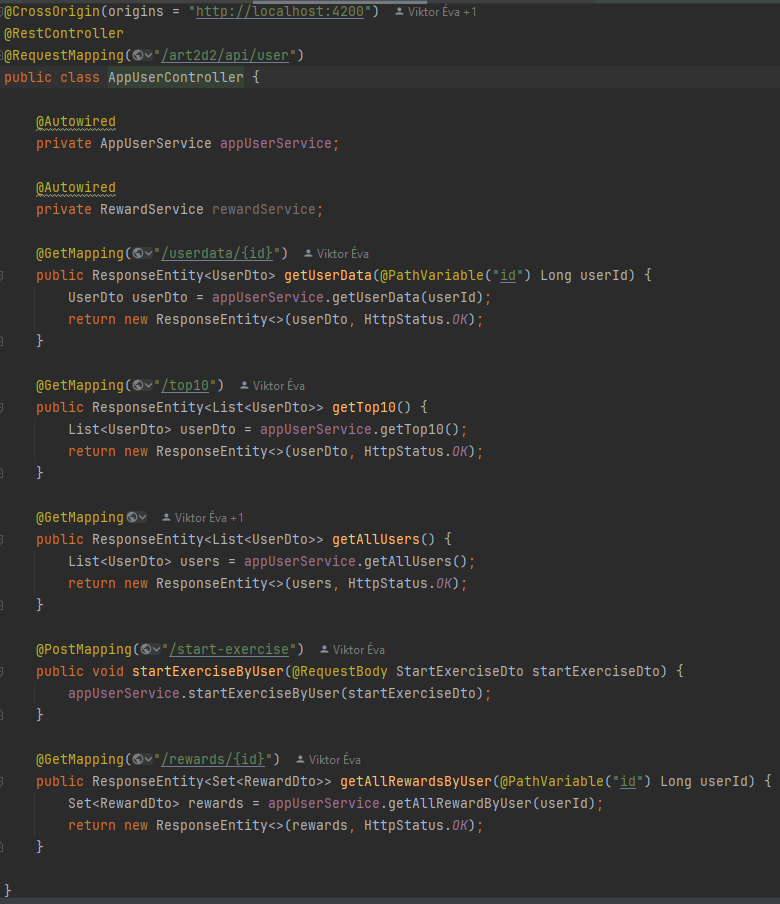
* + PracticeController

Injektált Bean-jei:

@Autowired  
private EyePracticeService practiceService;

Metódusa:

@GetMapping("/{type}")  
public ResponseEntity<EyePracticeDto> getPracticeByType(@PathVariable("type") String type) {  
 EyePracticeDto practiceDto = practiceService.getPracticeByType(type);  
 return new ResponseEntity<>(practiceDto, HttpStatus.*OK*);  
}

**Példa:**

###### c) **Service réteg**

* A business logikát valósítja meg.
* Az osztályok @Service annotációval vannak megjelölve, ami jelzi a Spring számára, hogy üzleti logikát tartalmazó osztályként kezelje.

Implementált szerviz osztályok:

* AppUserService

Injektált Bean-jei:

@Autowired  
private AppUserRepository userRepository;  
  
@Autowired  
AppUserMapper appUserMapper;  
  
@Autowired  
RewardService rewardService;  
  
@Autowired  
RewardMapper rewardMapper;

* AuthService

Injektált Bean-jei:

@Autowired  
private AuthenticationProvider authenticationProvider;  
  
@Autowired  
private PasswordEncoder passwordEncoder;  
  
@Autowired  
private AppUserRepository userRepository;  
  
@Autowired  
AppUserMapper appUserMapper;

* EyeExcerciseService

Injektált Bean-jei:

@Autowired  
private EyeExerciseRepository eyeExerciseRepository;  
  
@Autowired  
EyeExerciseMapper eyeExerciseMapper;  
  
@Autowired  
private AppUserService appUserService;

* EyePracticeService

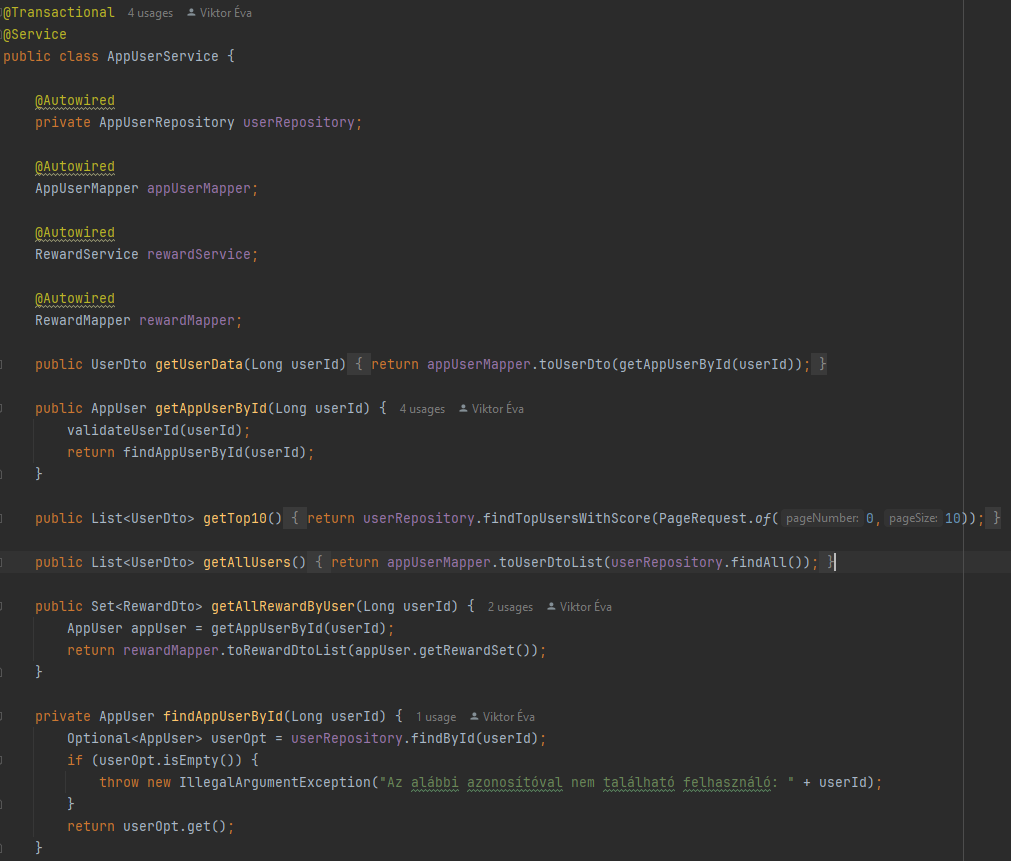
Injektált Bean-jei:

@Autowired  
private EyePracticeRepository eyePracticeRepository;  
  
@Autowired  
EyePracticeMapper eyePracticeMapper;

* RewardService

Injektált Bean-jei:

@Autowired  
private RewardRepository rewardRepository;  
  
@Autowired  
private AppUserRepository userRepository;

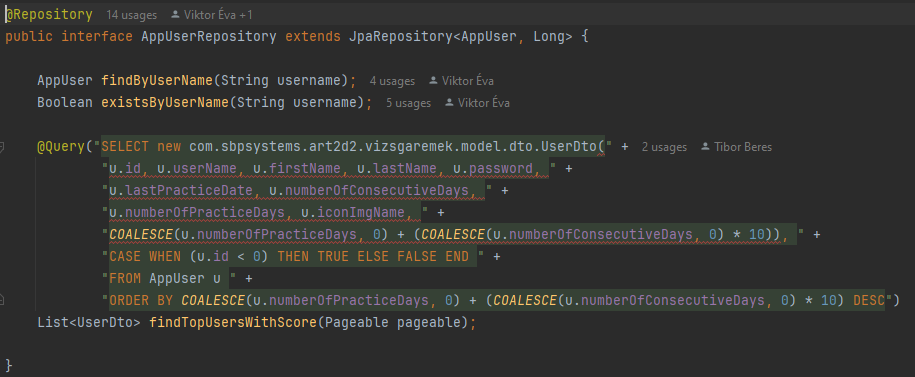
**Példa:**

###### d) **Repository réteg**

* Az adatbázis műveleteket végzi (CRUD-műveletek).
* Az osztályokat vagy interfészeket @Repository annotációval látják el.
* A Spring Data JPA általában automatikusan implementálja az egyszerű CRUD-funkciókat.

Implementált Repository osztályok:

* AppUserRepository
* EyeExcerciseRepository
* EyePracticeRepository
* RewardRepository

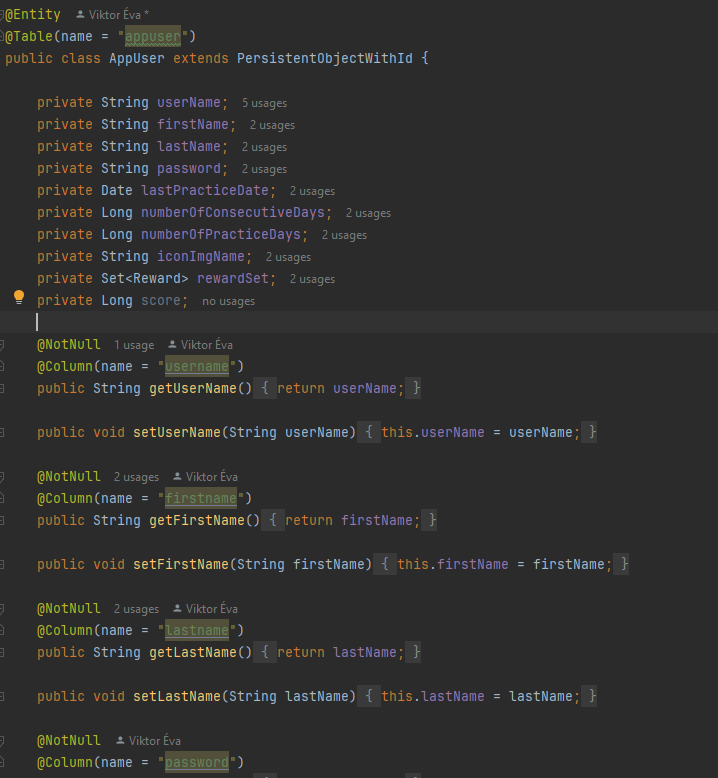
**Példa:**

###### e) **Model réteg**

* Az alkalmazás adatmodelljeit (entitásait) tartalmazza.
* Az osztályokat gyakran @Entity annotációval látják el, ami jelzi, hogy az adott osztály egy adatbázistáblához tartozik.

Implementált entitások:

* AppUser
* EyeExcercise
* EyeExcerciseImages
* EyePractice
* PersistentObjectWithId
* Reward

**Példa:**

##### **Konfiguráció és testreszabás**

###### a) **application.properties**

* Az alkalmazás fő konfigurációs fájlja, amelyben különböző tulajdonságokat határozhatunk meg, például:
  + Szerverport:

server.port=8080

* + Adatbázis-beállítások:

spring.datasource.url=jdbc:postgresql://localhost:5442/vizsgaremekspring.datasource.username=postgres

spring.datasource.password=postgres

* + Naplózási szintek:

logging.level.org.springframework=DEBUG

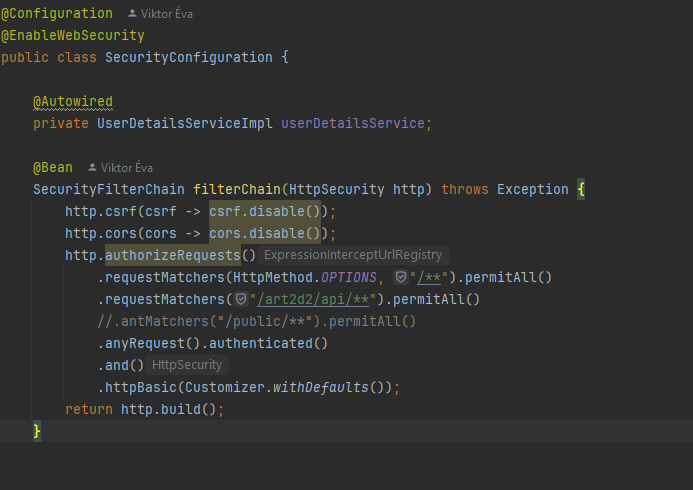
* + Liquibase

spring.liquibase.change-log=classpath:db/changeLog.xml

spring.liquibase.enabled=true

###### b) **Egyedi konfiguráció osztályokkal**

* Komplexebb konfigurációkhoz használhatunk @Configuration annotációval ellátott osztályokat.

**Példa:**

###### c) **Profile-alapú konfiguráció**

* A Spring lehetővé teszi különböző konfigurációk használatát környezetek szerint (pl. dev, test, prod).
* Az application-dev.properties és application-prod.properties fájlok segítségével különböző környezetekhez igazíthatjuk a beállításokat.
* Jelen projektünk mérete és célja miatt erre most nem volt szükség

**Aktiválás:**

spring.profiles.active=dev

##### DTO-k és Mapperek

###### **Miért nem utaztatjuk az entitásokat a hálózaton?**

**Kötöttség a belső adatmodellhez:**

* + Az entitások a belső adatbázis-modellt tükrözik. Ha ezeket közvetlenül küldjük el, a belső struktúra felfedése miatt a rendszer sebezhetővé válhat.
  + A belső adatmodell változása (pl. új mezők hozzáadása vagy táblák újrastrukturálása) automatikusan API-változásokat eredményezne, ami kliensoldali problémákat okozhat.

**Biztonsági kockázatok:**

* + Az entitások gyakran tartalmaznak érzékeny vagy belső használatra szánt adatokat, amelyeket nem lenne szabad kitenni a hálózaton (pl. jelszóhash-ek, auditinformációk, belső azonosítók).
  + Egy nem megfelelően ellenőrzött entitás küldése megkönnyítheti a támadásokat (pl. SQL injection kockázat).

**Lazy Loading problémák:**

* + Az entitások általában tartalmaznak kapcsolatokat (pl. @OneToMany, @ManyToOne), amelyek lazy loading-et használnak. Ha egy ilyen kapcsolatot hálózaton keresztül próbálunk elérni, ez LazyInitializationException-t okozhat, ha az entitás már nincs az EntityManager kontextusában.

**Nagy adatméret:**

* + Az entitások sok felesleges adatot tartalmazhatnak (pl. metaadatokat, kapcsolatok teljes hierarchiáját), ami felesleges sávszélességet fogyaszthat.

**Integrációs nehézségek:**

* + Az entitások közvetlen használata kliensoldalon (pl. frontend) technológiai függőségeket okozhat, mivel azok gyakran JPA-specifikus annotációkat tartalmaznak, amelyek nem kompatibilisek más környezetekkel.

###### **Miért használunk DTO-kat?**

**Adat minimalizálása:**

* + A DTO-k csak a szükséges adatokat tartalmazzák, így csökkentik a hálózaton átadott információk mennyiségét.
  + Például egy felhasználó entitása tartalmazhat jelszót és auditinformációkat, míg a DTO csak a felhasználónevet és az e-mail címet tartalmazza.

**Rugalmasság:**

* + A DTO-k segítségével a külső API-t függetleníthetjük a belső adatmodell változásaitól.
  + Például ha az entitás módosul, a DTO-kat és az API-kat nem feltétlenül kell megváltoztatni.

**Jobb kliensélmény:**

* + A DTO-k kifejezetten az API fogyasztói igényeihez igazíthatók, így könnyebbé válik a kliensoldali fejlesztés.

**Adatformázás:**

* + A DTO-k lehetőséget adnak az adatok előfeldolgozására (pl. dátumformázás, számítások elvégzése) még a szerveren, ezzel tehermentesítve a klienst.

##### Enumok

A nyeremények típusait és a gyakorlatok typusait Enum osztályokkal valósítottuk meg, melyek a következők:

* EyepracticeType

Értékei lehetnek: *TORNA*("Torna"),  
*OLVASAS*("Olvasás");

* RewardType

Értékei lehetnek: *PRACTICEDAYS\_REWARD*,  
*CONSECUTIVEDAYS\_REWARD*;

### **Miért használunk mappereket?**

**Automatikus átalakítás:**

* + A mapperek (pl. MapStruct vagy ModelMapper) automatikusan átalakítják az entitásokat DTO-kká és vissza, ezzel csökkentve a kézi kódolásra fordított időt.

**Kód tisztasága:**

* + A mapperek használatával elkerülhető az, hogy a konverziós logika szétszóródjon a kódban, ami tisztább és karbantarthatóbb rendszert eredményez.

**Teljesítményoptimalizálás:**

* + A statikus mapperek (pl. MapStruct) fordítási időben generált kódot használnak, ami sokkal gyorsabb, mint a runtime reflektív mapperek (pl. ModelMapper).

**Összetett konverziók kezelése:**

* + A mapperek támogatják az összetett átalakításokat, például a kapcsolódó entitások DTO-kká alakítását, feltételek alapján történő mezőkitöltést stb.

###### Implementált DTO-k:

* EyeExcerciseDto

Mezői:

private Long id;  
private Long eyePracticeId;  
private Integer orderNumber;  
private String name;  
private String text;  
private List<EyeExerciseImagesDto> images;

* EyeExcerciseImageDto

Mezői:

private Long eyeExerciseId;  
private Integer orderNumber;  
private String imgName;

* EyePracticeDto

Mezői:

private Long id;  
private String name;  
private EyePracticeType type;  
private Integer recommendedTime;  
private String disclaimer;

* LoginDto

Mezői:

private String userName;  
private String password;

* RegistrationDto

Mezői:

private String userName;  
private String firstName;  
private String lastName;  
private String password;

* RewardDto

Mezői:

private Long id;  
private String name;  
private RewardType type;  
private String imgName;

* StartExcerciseDto

Mezői:

private Long userId;

* UserDto

Mezői:

private Long id;  
private String userName;  
private String firstName;  
private String lastName;  
private String password;  
private Date lastPracticeDate;  
private Long numberOfConsecutiveDays;  
private Long numberOfPracticeDays;  
private String iconImgName;  
private Long score;  
private boolean isAdmin;

###### Implementált Mapperek:

* AppUserMapper
* EyeExcerciseMapper
* EyePracticeMapper
* RewardMapper

#### Frontend

##### Az Angular alkalmazások tipikus elemei, működésük és konfigurációjuk

Az angular keretrendszer rugalmasságának és moduláris megközelítésének köszönhetően hatékonyan támogatja az összetett frontend alkalmazások fejlesztését. A megfelelő konfiguráció és struktúra kialakítása biztosítja az átláthatóságot, a karbantarthatóságot és a skálázhatóságot

Az Angular egy népszerű TypeScript-alapú frontend keretrendszer, amely támogatja a komponensalapú fejlesztést és a moduláris architektúrát. Az alábbiakban áttekintjük az Angular alkalmazások tipikus elemeit, azok működését és a konfiguráció lehetőségeit.

###### **Modulok (Modules)**

Az Angular alkalmazások a @NgModule annotációjával ellátott modulokból épülnek fel, amelyek egyesítik a komponenseket, direktívákat, pipe-okat és szolgáltatásokat.

* **Root modul**: Az alkalmazás belépési pontja (app.module.ts). Ez konfigurálja az alkalmazást, és tartalmazza a root komponens definiálását.
* **Feature modulok**: A nagyobb alkalmazásokat kisebb, jól elkülönített részekre osztják. Ilyenek például eye-practise.module.ts, top-10.module.ts stb.
* **Shared modul**: Az alkalmazásban többször használt komponenseket, pipe-okat és direktívákat tartalmazza.Pl.: shared.module.ts

**Konfiguráció:** Az imports tömbben adhatjuk meg a külső és belső modulok függőségeit.

@NgModule({  
 imports: [CommonModule, FormsModule, MaterialModule, ReactiveFormsModule, RouterModule],  
 declarations: [...components],  
 exports: [...components, MaterialModule, CommonModule],  
})  
export class SharedModule {  
   
}

###### **Komponensek (Components)**

A komponensek az Angular alkalmazások alapvető építőelemei, amelyeket a @Component dekorátorral hozunk létre. Egy komponens tartalmazza:

* **HTML sablon (template)**: Az adatok megjelenítésére szolgál.
* **CSS/SCSS stílusok**: Az adott komponens kinézetének definiálására.
* **TypeScript logika**: Az interakciókat és adatkezelést valósítja meg.

**Konfiguráció:** A @Component annotációval adhatjuk meg a sablon és a stílus fájlokat:

@Component({  
 selector: 'pst-base-edit',  
 templateUrl: './base-edit.component.html',  
 styleUrl: './base-edit.component.scss'  
})  
export class BaseEditComponent implements AfterViewInit, OnInit {implementation…}

###### **Szolgáltatások (Services)**

A szolgáltatások az üzleti logika és az adatok központi kezelésére szolgálnak. A @Injectable annotációval ellátott elemek, ezek teszik lehetővé a dependency injectiont (DI).

* Az HttpClient szolgáltatással például könnyen végezhetünk API hívásokat.
* A szolgáltatások singletonként működnek az adott DI tartományon belül.

**Konfiguráció:**

* Automatikusan rendelkezésre állnak az providedIn: 'root' konfigurációval.
* Kézzel is deklarálhatjuk őket az AppModule providers tömbjében.
* @Injectable({ providedIn: 'root' })  
  export class UserService {  
   constructor(private http: HttpClient) { }  
    
    
   getUserData(userId: number) {  
   return this.http.get<AppUser>(`${***environment***.apiUrl}/user/userdata/${userId}`);  
   }  
    
   getTop10() {  
   return this.http.get<AuthUser[]>(`${***environment***.apiUrl}/user/top10`);  
   }  
    
   getAll() {  
   return this.http.get<AuthUser[]>(`${***environment***.apiUrl}/user`);  
   }  
    
   postStartExercise(userId: number) {  
   return this.http.post<any>(`${***environment***.apiUrl}/user/start-exercise`, { userId });  
   }  
    
   getAllRewardsByUser(userId: number) {  
   return this.http.get<Reward[]>(`${***environment***.apiUrl}/user/rewards/${userId}`);  
   }  
    
  }

###### **Direktívák (Directives)**

A direktívák az Angular sablonok viselkedésének módosítására szolgálnak. Két fő típusuk van:

* **Strukturális direktívák**: A DOM struktúráját módosítják (például \*ngIf, \*ngFor).
* **Attribútum direktívák**: Egy meglévő DOM elem megjelenését vagy viselkedését módosítják (például ngClass, ngStyle).
* **Jelenleg nem tartalmaz a projekt direktívákat**.

###### **Pipe-ok (Pipes)**

A pipe-ok az adatok formázására szolgálnak a sablonban. Az Angular beépített pipe-okat biztosít, mint például | date, | currency, de egyedi pipe-okat is létrehozhatunk.

###### **Routing és Navigáció**

Az Angular Router modul lehetővé teszi az egyoldalas alkalmazások (SPA) különböző nézetei közötti navigációt.

* A RouterModule konfigurációja az útvonalakat definiálja.
* Paraméterezett útvonalak (/details/:id) lehetővé teszik dinamikus tartalom betöltését.

**Példa:**

const routes: Routes = [  
 { path: '', component: EyePractiseContainerComponent },  
 {  
 path: 'simple-eye-practises',  
 data: {title: 'Szemtorna'},  
 component: PlainEyeExercisesComponent,  
 },  
 {  
 path: 'read-small-letters',  
 data: {title: 'Kisbetűk olvasása'},  
 component: ReadingMinusculeLettersComponent,  
 }  
];  
  
@NgModule({  
 imports: [RouterModule.*forChild*(routes)],  
 exports: [RouterModule],  
})  
export class EyePractiseRoutingModule {}

###### **Adatkezelés**

Az Angular alkalmazásokban az adatok kétirányú kötésével (two-way data binding) a sablon és a komponens logika között egyszerűen kezelhetjük az adatok változását.

* Az adatokat egyirányúan ([property]) vagy kétirányúan kötjük ([(ngModel)]).
* Az @Input() és @Output() annotációk lehetővé teszik az adatok komponensek közötti továbbítását.

###### **Külső könyvtárak integrációja**

Az Angular CLI egyszerűvé teszi külső könyvtárak telepítését és használatát (például Angular Material, ngx-translate, stb.).

ng add @angular/material

###### **Környezetfüggő konfigurációk**

Az Angular alkalmazás rootjában tároljuk a különböző környezetekre (pl. fejlesztés, éles) vonatkozó konfigurációkat. pl.: angular.json

**Példa:**

"configurations": {  
 "production": {  
 "buildTarget": "ART2D2VizsgaremekGui:build:production"  
 },  
 "development": {  
 "buildTarget": "ART2D2VizsgaremekGui:build:development"  
 }  
 },  
 "defaultConfiguration": "development"  
},

### Fejlesztési lehetőségek

Mivel ez az alkalmazás „csak” a vizsga kedvéért, annak követelményeinek teljesítése érdekében jött létre így rengeteg területen nem csak lehet, hanem szükséges is a fejlesztése, ha úgy tekintünk rá, mint egy alkalmazásra, ami éles környezetben is futtatásra kerül. Ezek közül a teljesség igénye nélkül csak párat emelnénk ki pár szóban.

#### Biztonság

Jelenleg egy „HTTP BASIC” authentikáció került megvalósításra, ami a mai modern internetes világban már nem állja meg a helyét, így itt mindenképp továbbfejlesztés szükséges. Erre több lehetőség is létezik.

Néhány példa:

* + Felhasználónév-jelszó alapú hitelesítés
  + HTTP\_BASIC (Jelenleg megvalósítva)
  + OAuth2 és OpenID Connect
  + JWT (Json Web Token)
  + LDAP
  + Digest
  + SAML
  + Kerberos vagy NTLM

Jogosultságkezelés pedig nem került még implementálásra, pedig az elengedhetetlen ahhoz, hogy bizonyos tartalmakat csak bizonyos jogosultsággal rendelkező személyek láthassanak, kezelhessenek. Ez alapján ez a terület is fejlesztésre szorul.

Spring Boot és különösen a Spring Security széles körű támogatást nyújt jogosultságkezelésre. Az alábbiakban néhány lehetséges megoldást mutatunk be:

* + **Role alapú jogosultság kezelés**

A felhasználókhoz szerepköröket (pl. ROLE\_ADMIN, ROLE\_USER) rendelhetsz.

Az URL-ekhez, metódusokhoz vagy erőforrásokhoz jogosultságot adhatsz a szerepkörök alapján.

* + **Authority alapú jogosultság kezelés**

Finomabb jogosultságkezelés, ahol nem csak szerepkörök, hanem konkrét jogosultságok alapján korlátozhatók az erőforrások (pl. READ\_PRIVILEGE, WRITE\_PRIVILEGE).

* + **URL alapú jogosultságkezelés**

URL szinten szabályozható, hogy mely endpointokat érhetnek el a felhasználók.

* + **Metódus alapú jogosultságkezelés**

A Spring Security annotációival szabályozhatók a hozzáférések a metódus szintjén.

Gyakori annotációk:

@PreAuthorize: Metódus előtt ellenőrzi a jogosultságot.

@PostAuthorize: Metódus után ellenőrzi az eredményt.

#### Felhasználói felület

Jelenleg a regisztráció és a bejelentkezés folyamatoknál már ad visszajelzést a rendszer, ha valamit nem jól töltünk ki, de sokszor nem egyértelmű, hogy az adott mezőnél miért nem jó az érték. Ezeket a fajta hibakezeléseket és üzeneteket tovább lehetne fejleszteni a felhasználói élmény növelése érdekében.

#### Funkcionalitás

Regisztrációkor most automatikusan mentjük a felhasználót az adatbázisba. Ez a metódus több helyen is fejleszthető. Alapvető validálásként e-mail-es visszaigazolást lehetne küldeni. A regisztráció során capcha is ellenőrizhetné, hogy valós személy szeretne-e regisztrálni. Napjainkban az is elvárható lenne, hogy egy nagyobb tech cég fiókjával integráltan regisztráljunk felhasználót. pl.: gmail fiók, facebook fiók stb… Gyakorló feladatok tekintetében most két gyakorlatilag statikus oldal található a webalkalmazásban. További fejlesztésekkel ezek lehetnének interaktívak is, animációkkal, gyakorlat segítő felületekkel és funkciókkal. A legtovább fejleszthető és variálható része a projektnek a Pontrendszer és játék élmény a gyakorlás mellett. A jelenlegi nagyon egyszerű pontszámítás a feladatok interaktívabbá tételével párhuzamosan egyre összetettebbé válhatna. Pl. online real time multiplayer, amiben a gyakorlatokat verseny szerüen végezhetjük többedmagunkkal.

## Felhasználói dokumentáció

Az elkészült projekt egy Webes alkalmazás, ami multiplatform, azaz bármilyen internet képes eszközzel használható, mivel csak egy böngészőre van szükség a használatához. A felület responsive dizájnal készült, így szinte minden képernyőmérettel használható az alkalmazás. A képernyők automatikusan igazodnak a megjelenítő felület méretéhez. Alapértelmezésként a <http://localhost/4200> címen érhető el, de éles környezetben a vásárolt domain név beírásával navigálhatunk a kezdőlapra. Pl.: <http://szemtorna.hu> Ekkor a kezdőképernyőre jutunk.

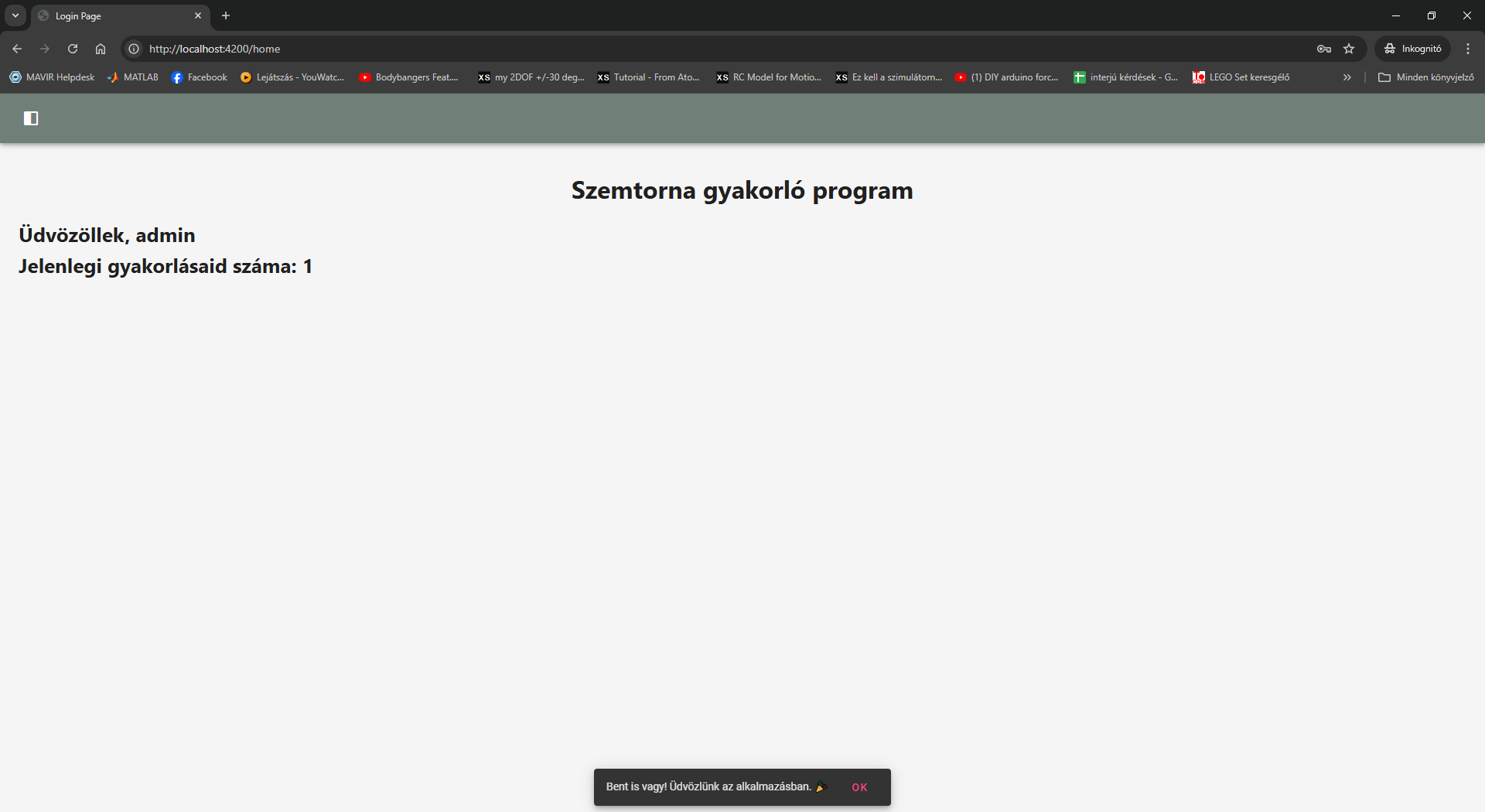
### Kezdő képernyő (bejelentkezés)

Ezen a képernyőn két lehetősége van a felhasználónak. Amennyiben rendelkezik már regisztrált felhasználóval, akkor kitölti a felhasználónév és a jelszó mezőt, majd a „Bejelentkezés” gombra kattintva belép a webalkalmazásba. Ha nem rendelkezik még felhasználói fiókkal, akkor a „Regisztráció” gombra kattintva átnavigál a regisztrációs felületre az alkalmazás.

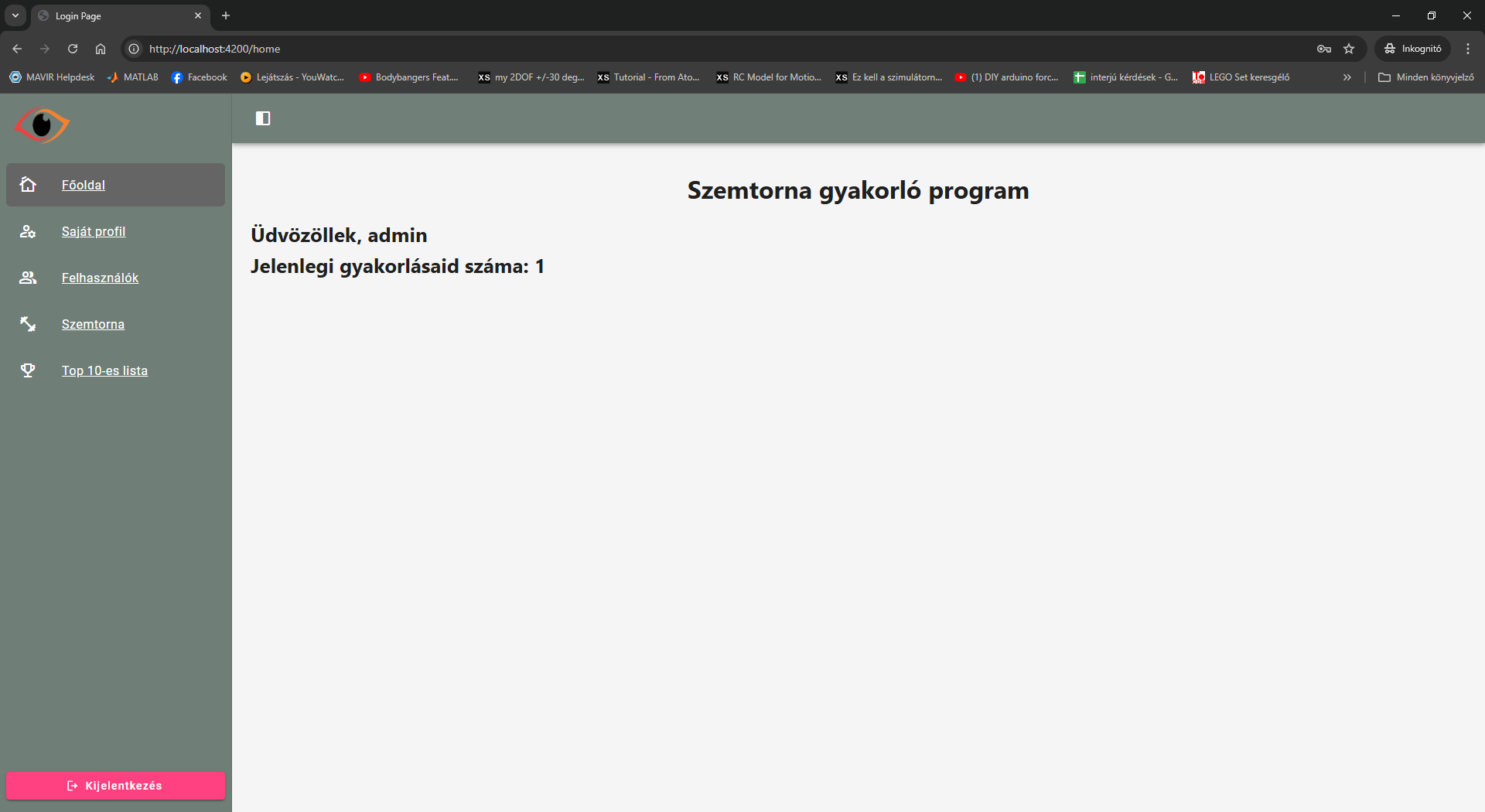
### Regisztrációs felület

Ezen a felületen a megfelelő mezők kitöltése után a „Regisztráció” gomb megnyomásával regisztrálhatjuk a felhasználói fiókunkat a rendszerbe. Amennyiben valamelyik adat nem megfelelő, vagy hiányzik, azt a rendszer visszajelzi a felhasználó számára.

### Főoldal és navigáció

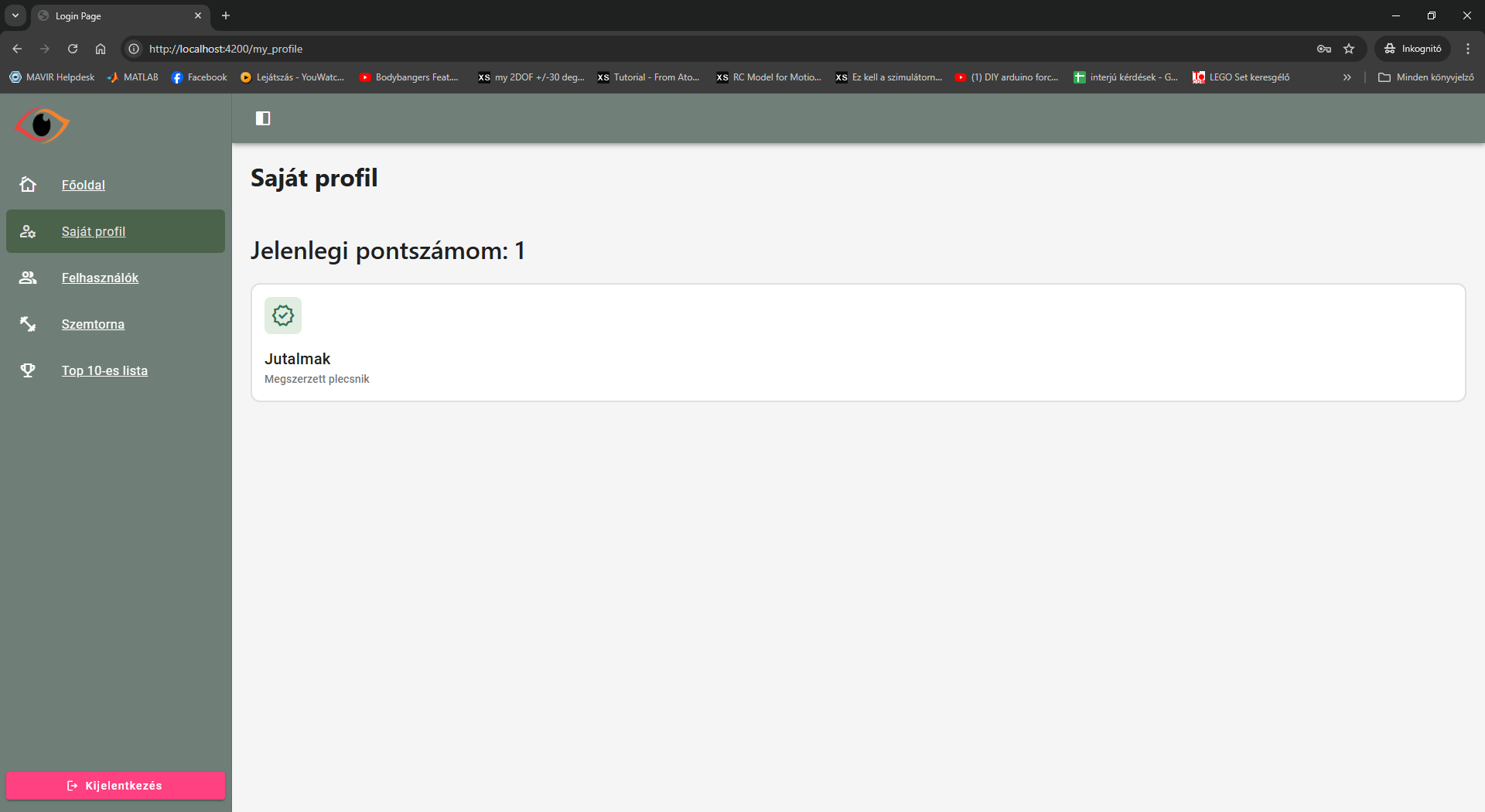


Sikeres bejelentkezés esetén üdvözöl minket az alkalmazás egy felugró ablakban ami automatikusan eltűnik, de az ok gomb megnyomásával eltüntethető. Ezen kívül statikusan is köszönt minket és kiírja az eddigi gyakorlásaink számát. Az ablak bal felső sarkában lévő ikonra kattintva előhozható és elrejthető a navigációs sáv.

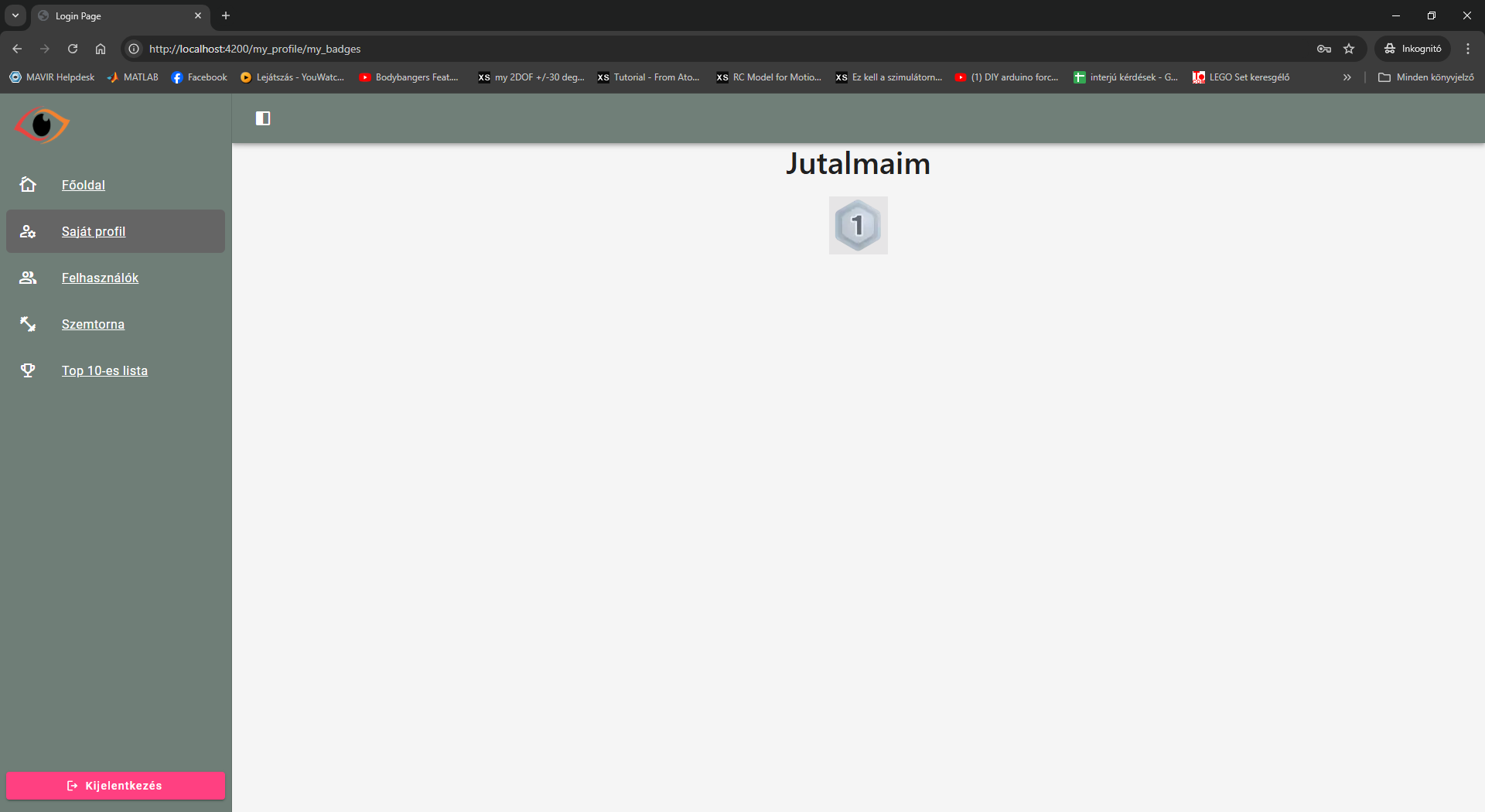


A megjelenő menüsávban tudunk navigálni a webalkalmazás oldalai között. Ezen kívül itt van lehetőségünk a kijelentkezésre is az alkalmazásból. Kijelentkezés esetén újra a bejelentkező felületen találjuk magunkat. A menüsávból az alábbi oldalak érhetők el:

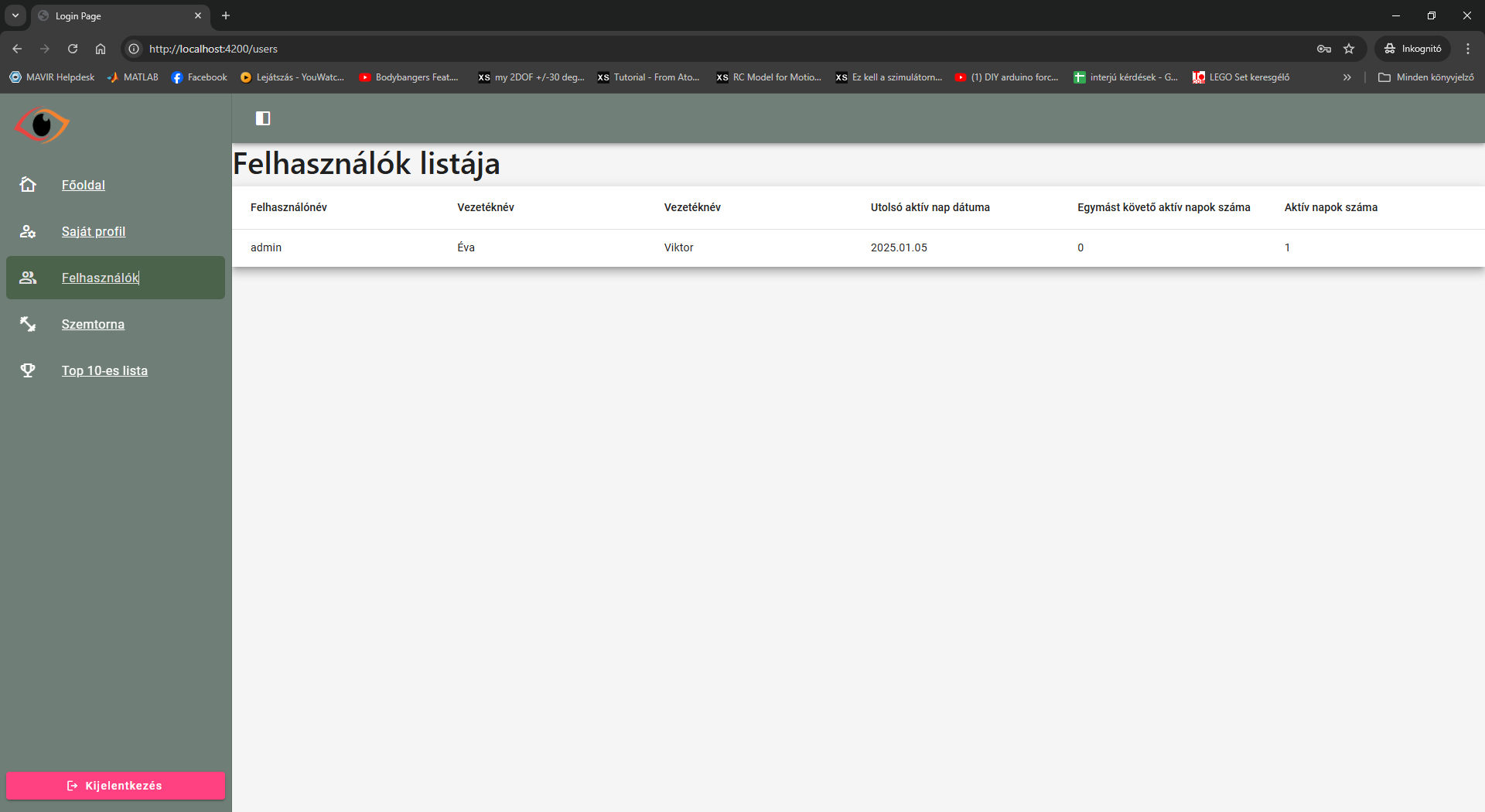
### Saját profil



Ezen a felületen látható az aktuálisan elért pontszám, és lehetőség van átnavigálni a megszerzett jutalmak felületére. A jutalmak csempére kattintva átkerülünk a jutalmaink nézetre.

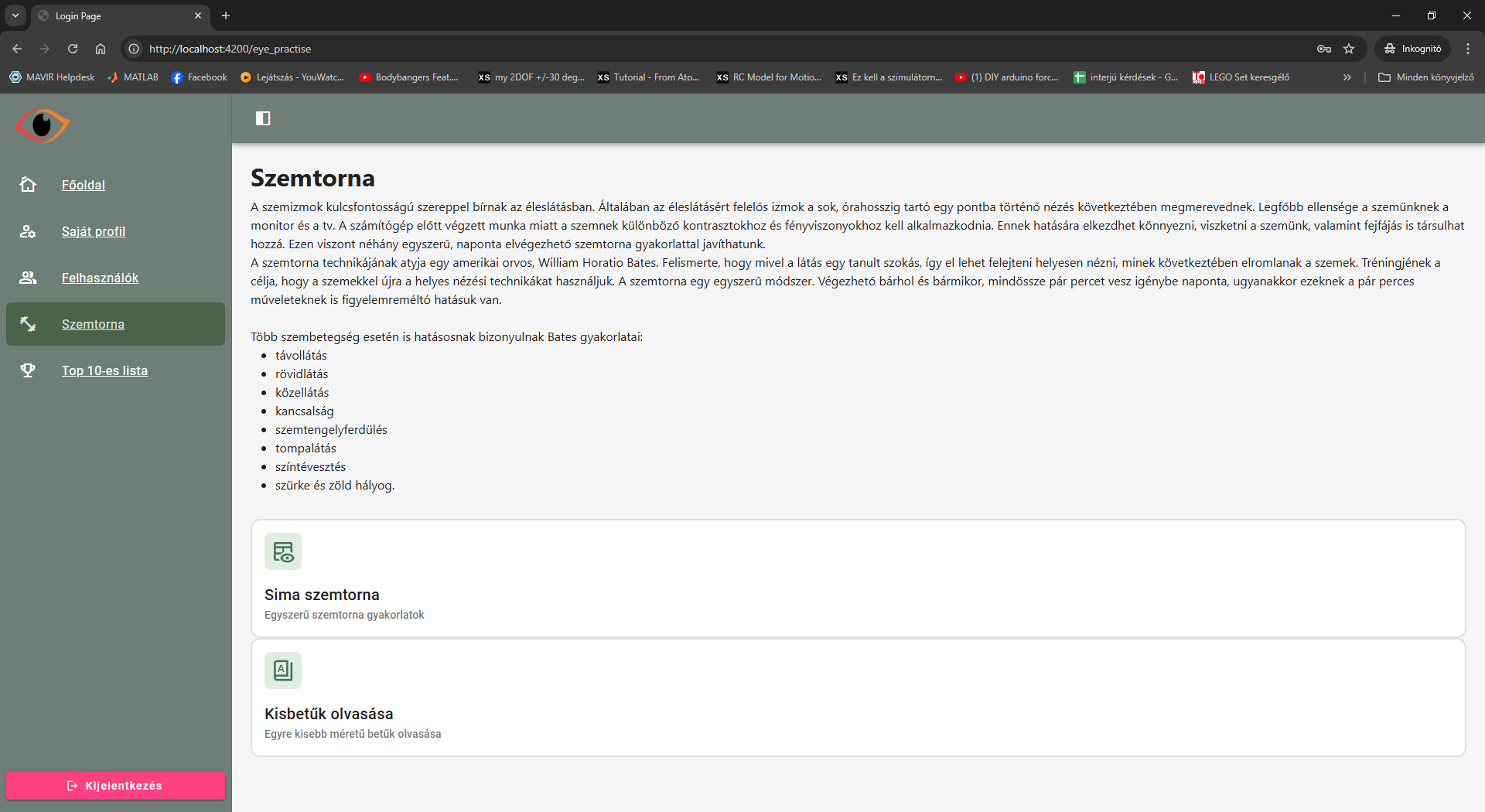


### Felhasználók

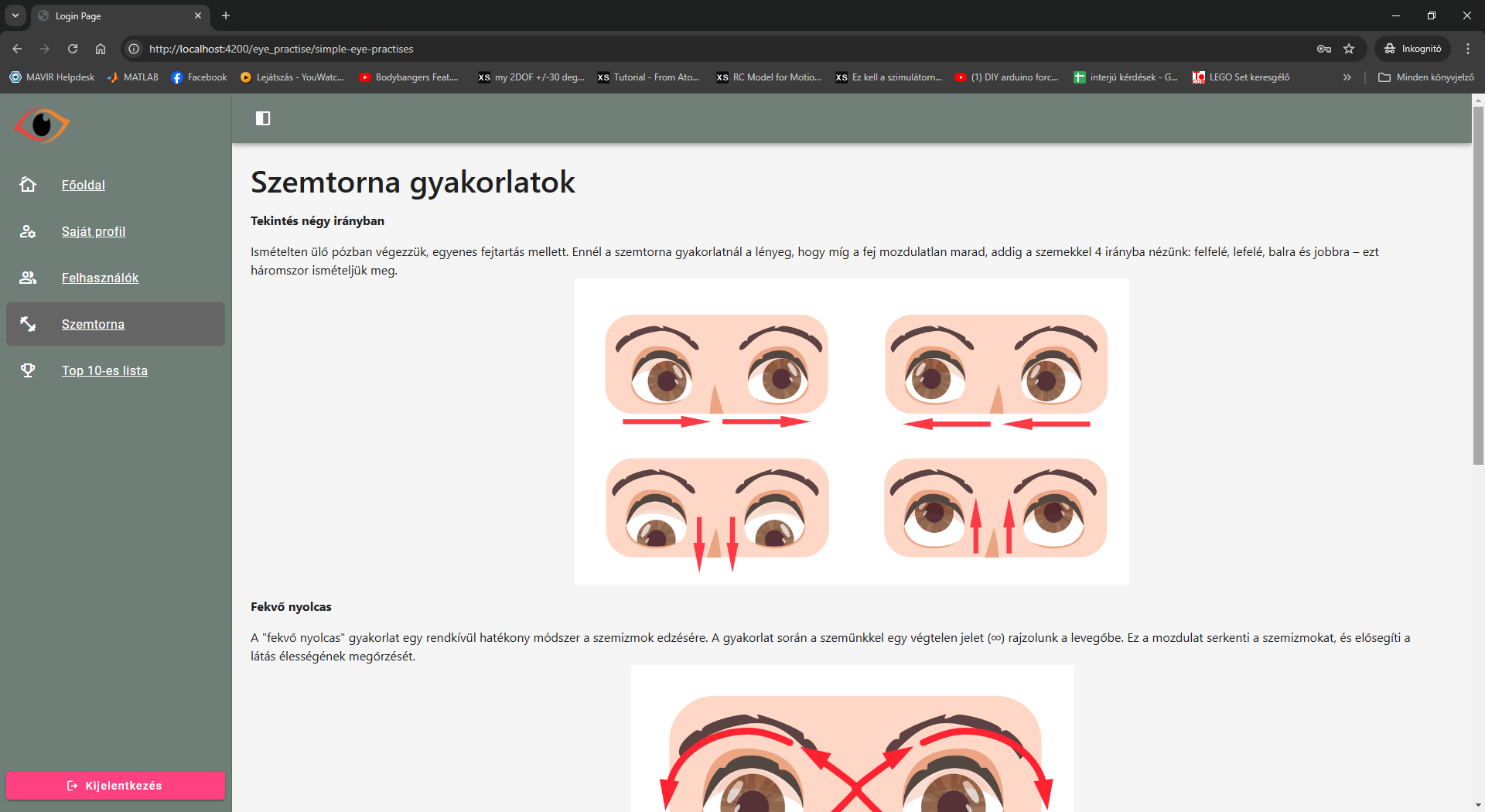


Ez a felület alapvetően az adminisztrátor számára készült. Itt van lehetőségünk áttekinteni a felhasználók adatait. Ez a felület adna lehetőséget a felhasználók kezelésére is, de ez még nem került implementálásra.

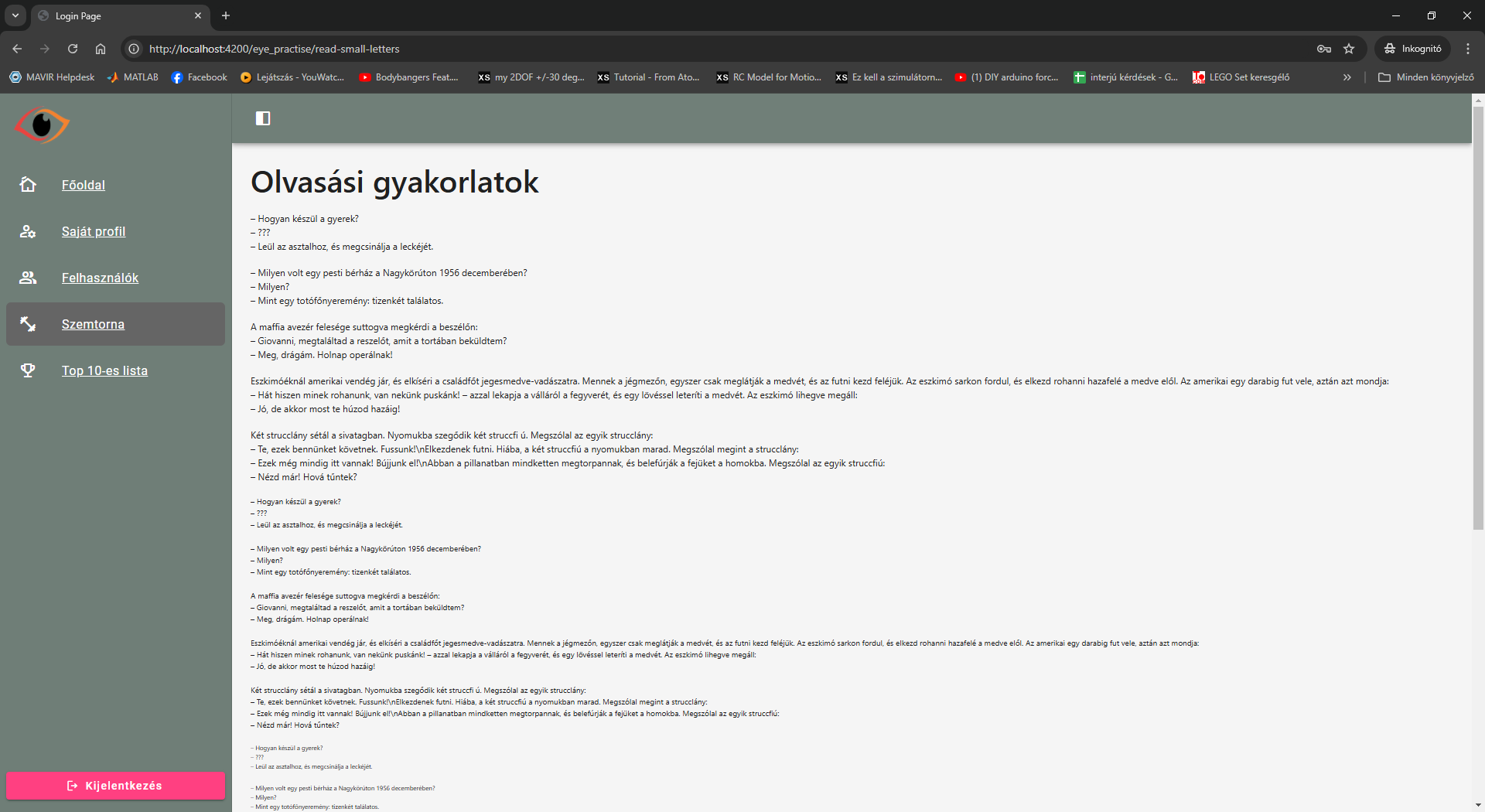
### Szemtorna



Ezen a felületen olvasható egy rövid összefoglalás a szemtornáról és itt van lehetőségünk választani, hogy milyen típusú szemtornát szeretnénk gyakorolni.

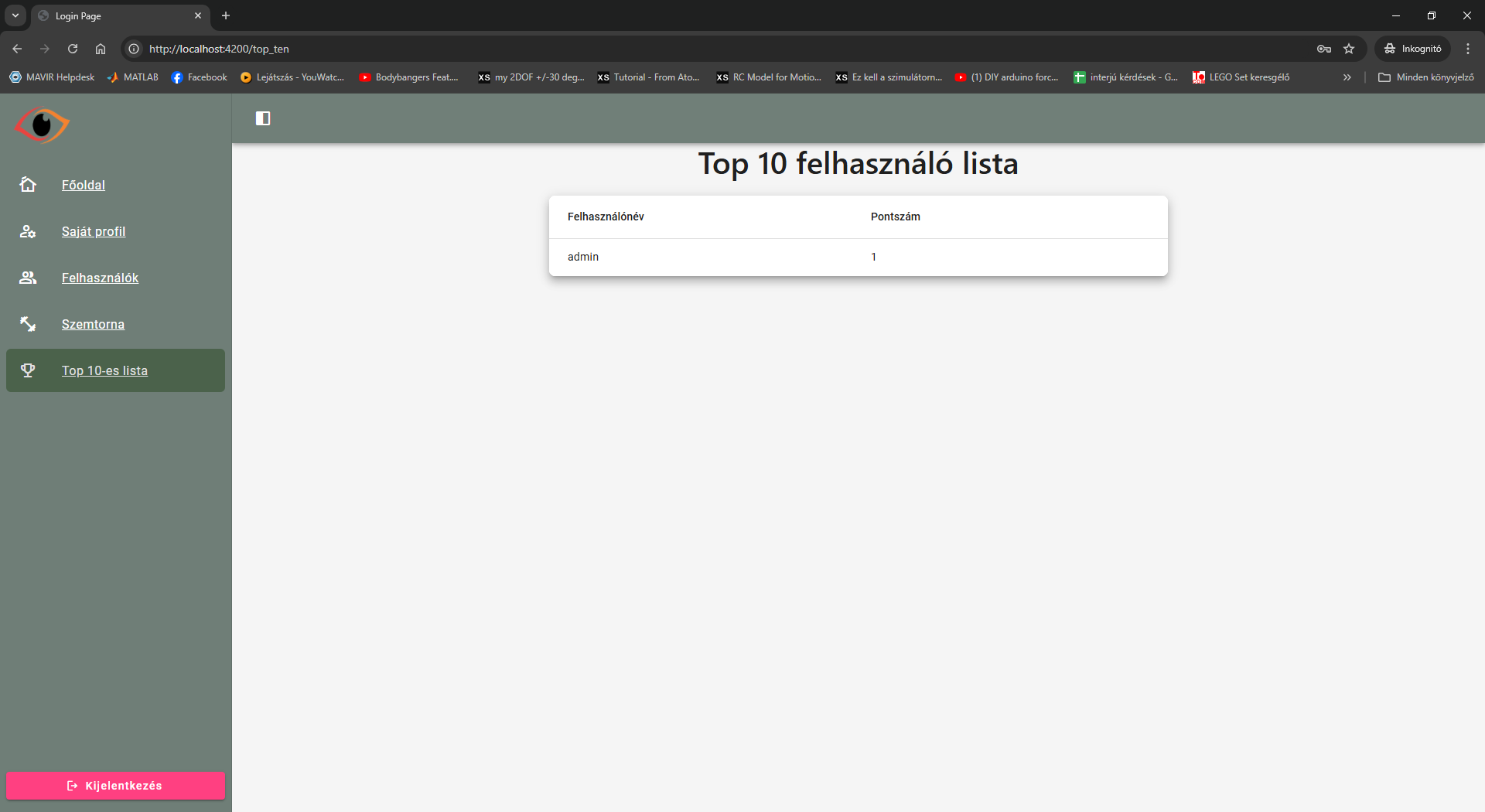


Sima szemtorna gyakorlatokat, vagy apró betűk olvasását.



A gyakorlatok megnyitásáért pontot ad a rendszer. Ezzel teszi kompetitívvé a gyakorlást.

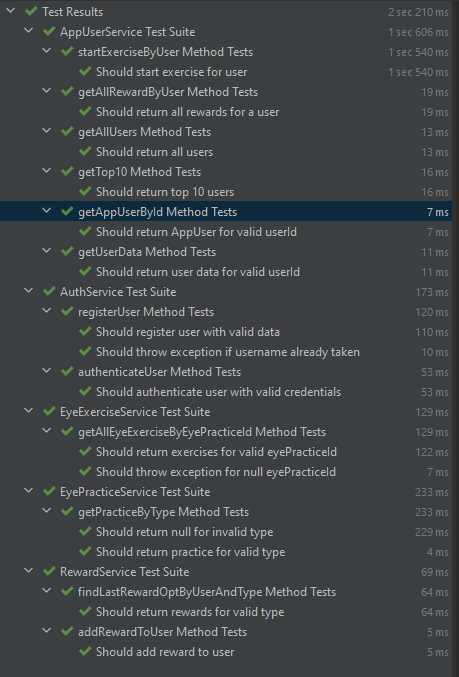
### Top 10-es lista



Ezen a felületen látható az alkalmazást használó felhasználók közül a legtöbb ponttal rendelkező személyek listája. Az első 10 személy kerül felsorolásra csak, pontszámmal együtt.

## Tesztelés

### Automatizált

A backend oldalon a Service osztályok metódusaira készültek JUnit automatizált egységtesztek, melyek sikeresen lefutnak az alábbi esetekkel.:

Ezek a tesztek megtalálhatók és futtathatók a forráskódban.

### Manuális

Frontend oldalon, illetve integrációs tekintetben manuális tesztelést végeztünk a szoftveren az elkészült funkciók kipróbálásával. Mivel a felhasználói leírás képein egyértelműen követhető, hogy a felületek alapvetően működnek, ezért ebben a részben csak néhány kiemelt esetet mutatnánk be.

#### Kiemelt szélsőséges esetek



Üres felhasználónév és jelszó esetén mind két mező pirossal kiemelve.



Túl rövid jelszó megadása esetén figyelmeztető üzenet jelenik meg.

## Összefoglalás

Az elkészült projekt egy egyszerű, de komplett alkalmazás, amely a vizsga követelményeinek teljesítésére és a csapat szakmai fejlődésére helyezte a hangsúlyt. A csapat sikeresen megvalósította azokat a célkitűzéseket, amelyek az első fejezetben megfogalmazásra kerültek, és egy funkcionális, gyakorlatban használható alkalmazást hozott létre. A rendszer alapvetően megfelel az elvárt működési és minőségi kritériumoknak, de sokkal inkább egy megbízható alapot nyújt több területen történő továbbfejlesztéshez. Révén, hogy ez egy vizsga mű, nem volt cél a csapat számára, hogy olyan erős authentikációs és authorizációs réteget fejlesszen, ami egy éles környezetben is megállná a helyét, sem az, hogy a funkciók 100% kidolgozásra kerüljenek. Ez a projekt inkább arra alkalmas, hogy a használt technológiák alapjait prezentálja, ugyanakkor ezt ne csak papíron, hanem használható interaktív formában, kipróbálható módon tegye. Ez az, amit a csapat szeretett volna elérni, ami meg is valósult.

Tisztában vagyunk vele, hogy sok helyen még javításra, továbbfejlesztésre szorul az alkalmazás, ennek egy részét a fejlesztési lehetőségek fejezetben ki is fejtettük.

A fejlesztés során használt technológiák, mint például a Spring Boot backendhez és az Angular frontendhez, biztosították az alkalmazás stabilitását, bővíthetőségét és modern felhasználói élményt nyújtottak. Az alkalmazott verzionálási megoldások, mint a Liquibase az adatbázis-kezelés terén, lehetővé tették a séma változtatásainak precíz nyomon követését. Az elkészült adatmodell és az ehhez kapcsolódó funkcionalitások sikeresen támogatják a projekt által kitűzött célokat.

A projekt fejlesztése során a legnagyobb kihívást az integrációs feladatok jelentették, különösen a backend és a frontend közötti zökkenőmentes adatátvitel biztosítása. Emellett a biztonsági mechanizmusok megvalósítása és a felhasználóbarát interfész kialakítása is komoly tanulási lehetőségeket nyújtott a csapat számára.

Összességében a projekt nemcsak az oktatási követelmények teljesítéséhez járult hozzá, hanem jelentős szakmai tapasztalattal is gazdagított minden csapattagot. Az elkészült alkalmazás alapján lehetőség nyílik a további fejlesztésekre, mint például a funkcionalitások bővítése, a felhasználói élmény tovább javítása, vagy a biztonsági protokollok erősítése, ami hosszabb távon egy piacképes termék alapja lehet.