Pollák Csengő

“A jövő zenéje!”

Csapattagok

Jani Patrik (Csapatvezető)

Tamáskovits Gyula Ákos

Barna Máté Levente

Tartalomjegyzék

Bevezetés

Téma és Témaválasztás

Vizsgadolgozatunk témájának egy olyan projektet szerettünk volna választani, amely kapcsolódik az oktatáshoz és kézzelfogható változást hozhat az iskolai mindennapokba. Az ötlet a csapat egyik tagjától származik, aki felvetette, hogy egy webalkalmazással tehetnénk élénkebbé az iskolai légkört. Ebből kiindulva született meg a Pollák Csengő ötlete, amelynek célja a hagyományos csengőhang lecserélése egy dinamikusabb, változatosabb megoldásra, hogy feldobja a szüneteket és megtörje az évek alatt megszokott egyhangúságot.

A szoftver célja

A Pollák Csengő egy úttörő megoldás, amely kifejezetten arra lett tervezve, hogy javítsa az oktatási környezetet, és kényelmesebb, vonzóbb légkört biztosítson a diákok számára. Ez az innovatív alkalmazás egy sokoldalú webplatformot kínál, amely mobil eszközökön és számítógépeken egyaránt elérhető, biztosítva ezzel a széleskörű használhatóságot. A platform alapfunkcionalitása lehetővé teszi a diákok számára, hogy szórakoztató és interaktív módon vegyenek részt a választásban, amelynek során a kedvenc dalaikra szavazhatnak, és így meghatározzák a "Hónap Csengőhangját". Ez a funkció dinamikus csengőhangok széles választékát vezeti be, amelyek az iskolai év során rotálódnak, friss és élvezetes csengetési rendet biztosítva a diákok számára nap mint nap. A Pollák Csengő nemcsak a vidám és derűs légkör megteremtésében segít az iskolában, hanem könnyed és innovatív módon járul hozzá a diákok elköteleződéséhez, pozitívan befolyásolva azok általános jóllétét és mindennapi iskolai életét. Zökkenőmentes integrációjával és felhasználóbarát felületével ez az alkalmazás erősíti a közösségi érzést, miközben örömet és újdonságot csempész az oktatási környezetbe.

Kinek és miért

A Pollák Csengő alkalmazás elsősorban iskolák számára lett kifejlesztve, különösen azoknak, amelyek célja, hogy javítsák a diákok iskolai élményét, elősegítve a pozitív és közösség-orientált légkört. A rendszer lehetőséget biztosít a diákok számára, hogy aktívan részt vegyenek iskolájuk napi környezetének alakításában, kedvenc dalaikra szavazva, ami szórakoztató és interaktív elemet ad az iskola mindennapjaiba.

Kinek, miért?

- **Iskolák:** Olyan iskoláknak, amelyek célja, hogy élénk és vonzó környezetet alakítsanak ki, ahol a diákok jobban összekapcsolódnak és aktívan részt vesznek az iskola közösségi életében.
- **Diákok:** A diákoknak, hogy hangot adjanak véleményüknek az iskolai évük hangulatának meghatározásában, így élvezetesebbé téve mindennapi rutinjukat, és elősegítve a tulajdonosi és részvételi érzést az iskola kultúrájában.
- **Tanárok és személyzet:** A tanároknak és az iskolai személyzetnek, hogy dinamikusabb és felemelőbb légkört teremtsenek az iskolában, hozzájárulva ezzel a pozitív és támogató környezet kialakításához, amely kedvező a tanulásra és a társas interakciókra is.

Fejlesztői dokumentáció

Konkurenciák

A Pollák Csengő alkalmazás egyik vetélytársa a Bella-Iskolacsengő, amely egy hasonló szolgáltatást nyújt az iskolák számára. A Bella-Iskolacsengő egy online platform, amely lehetővé teszi az iskolák számára, hogy saját csengőhangot válasszanak, és a diákok számára lehetőséget biztosítanak a szavazásra. Az alkalmazás egyedi és személyre szabott csengőhangokat kínál, amelyeket a diákok kedvenc dalai alapján választhatnak ki. A Bella-Iskolacsengő egy modern és interaktív megoldás, amely segíti az iskolákat a diákok bevonásában és a közösségi élmény erősítésében.

A Bella-Iskolacsengő alkalmazás egy előnnyel rendelkezik, az egyedi csengőhangok és a személyre szabott választások lehetőségével. Az alkalmazás felhasználóbarát felülettel rendelkezik, amely egyszerű és intuitív használatot biztosít a diákok és az iskolai személyzet számára.

A Pollák Csengő alkalmazás és a Bella-Iskolacsengő közötti különbség az, hogy a Pollák Csengő egy dinamikus és interaktív platformot kínál, amely lehetővé teszi a diákok számára, hogy szavazzanak kedvenc dalaikra, és meghatározzák a "Hónap Csengőhangját". Az alkalmazás egyedi és változatos csengőhangokat kínál, amelyek napi szinten friss és élvezetes hallási élményt nyújtanak a diákoknak.

Egy másik vetélytárs a Gipen Iskolacsengő mely kevesebb funkcionalitást nyújt mint a Pollák Csengő. Az Iskolacsengő egy modern és minimalista megoldás, amely lehetővé teszi az iskolák számára, hogy saját csengőhangot válasszanak, és a diákok számára lehetőséget biztosítanak a szavazásra. Az alkalmazás egyszerű és letisztult felülettel rendelkezik, amely gyors és hatékony használatot biztosít a diákok és az iskolai személyzet számára.

Más vetélytárs a térben hasonló funkcionalitással, mint alkalmazásunk, jelenleg a piacon nem elérhető. Ez miatt a Pollák Csengő egyedi és innovatív megoldásnak számít, amely különleges és élvezetes hallási élményt nyújt a diákoknak.

Fejlesztői környezet

A **Pollák Csengő** alkalmazás fejlesztése során különböző modern és hatékony fejlesztőeszközöket használunk, amelyek támogatják a kódírást, az adatbáziskezelést, a konténerizációt. Ezek az eszközök nemcsak megkönnyítik a fejlesztési folyamatot, hanem biztosítják a stabil és hatékony működést is.

Ebben a részlegben részletesen bemutatjuk azokat a kulcsfontosságú eszközöket és technológiákat, amelyeket a fejlesztői környezetünkben használunk, beleértve:

- **WebStorm** mely a JetBrains által fejlesztett, mély integrációjáról ismert a JavaScript keretrendszerekkel, intelligens kódkiegészítéssel és robusztus hibakeresési képességekkel. Kiválóan alkalmas összetett projektekhez, különösen azokhoz, amelyek TypeScript-et és modern JavaScript keretrendszereket, mint például a React, Angular és Vue használnak. Beépített eszközei a verziókezeléshez, teszteléshez és refaktoráláshoz átfogó IDE-vé teszik. Versenytársaihoz, mint például a Sublime Text, képest a WebStorm fejlettebb funkciókat és jobb integrációt kínál a fejlesztési munkafolyamatokkal, bár magasabb költséggel jár.
- **Visual Studio Code** (VS Code) egy ingyenes, nyílt forráskódú kódszerkesztő, amely sokoldalúsága és kiterjedt bővítmény-ökoszisztémája miatt nagy népszerűségre tett szert. Számos programozási nyelvet és keretrendszert támogat, így sok fejlesztő kedvenc választása. A VS Code könnyű természete, valamint olyan erőteljes funkciók, mint az IntelliSense, beépített Git támogatás és beépített terminál, rendkívül hatékony fejlesztési környezetet biztosítanak. Az Atomhoz, egy másik népszerű szerkesztőhöz képest a VS Code gyorsabb teljesítményt és aktívabb közösséget kínál, ami gazdagabb bővítménykészletet és gyakori frissítéseket eredményez.
- **Docker** és **Docker-compose** a konténerizációhoz és a konzisztens fejlesztési környezetekhez: A Docker lehetővé teszi az alkalmazások konténerizálását, ami biztosítja a fejlesztési és futtatási környezetek konzisztenciáját. A Docker-compose segítségével egyszerűen definiálhatjuk és kezelhetjük a többkonténeres alkalmazásokat. Versenytársai, mint például a Vagrant, nem nyújtanak olyan könnyű és gyors konténerizációs megoldást, mint a Docker.

- **Pgadmin** az adatbázis kezeléséhez: A Pgadmin egy erőteljes és felhasználóbarát eszköz a PostgreSQL adatbázisok kezeléséhez. Lehetővé teszi az adatbázisok egyszerű adminisztrációját és karbantartását. Versenytársai, mint például a DBeaver, szintén jó alternatívák, de a Pgadmin kifejezetten a PostgreSQL-re optimalizált, ami előnyt jelent a specifikus funkciók és teljesítmény szempontjából.
- **nginx** a frontend kiszolgálásához: Az nginx egy nagy teljesítményű és könnyen konfigurálható webszerver, amely kiválóan alkalmas a statikus fájlok kiszolgálására és a terheléselosztásra. Versenytársai, mint például az Apache, szintén népszerűek, de az nginx alacsonyabb erőforrásigénye és nagyobb teljesítménye miatt előnyösebb választás a modern webalkalmazások számára.
- **Postgres 16.6** a backend adatbázishoz: A PostgreSQL egy nyílt forráskódú, robusztus és skálázható adatbázis-kezelő rendszer, amely számos fejlett funkcióval rendelkezik, mint például a tranzakciókezelés és a replikáció. Versenytársai, mint például a MySQL, szintén népszerűek, de a PostgreSQL fejlettebb funkciói és rugalmassága miatt gyakran előnyösebb választás a komplex alkalmazások számára.

Felhasznált technológiák

A **Pollák Csengő** alkalmazás modern és hatékony technológiákra épül, amelyek biztosítják a stabil működést, a gyors fejlesztési folyamatokat és a felhasználóbarát élményt. Az alkalmazás fejlesztése során a legújabb frontend és backend megoldásokat alkalmaztuk, amelyek lehetővé teszik a skálázható és könnyen karbantartható kód írását. Az alábbiakban megismerheti azokat a technológiákat és eszközöket, amelyek kulcsszerepet játszottak a fejlesztési folyamatban:

- **TypeScript**: A TypeScript egy statikusan típusos JavaScript kiterjesztés, amely opcionális statikus típusokat ad a nyelvhez. Segít a fejlesztőknek korán észlelni a hibákat a típusellenőrzés révén, és javítja a kód karbantarthatóságát és olvashatóságát. A JavaScripthez képest a TypeScript jobb eszköztámogatást és skálázhatóságot kínál nagy kódbázisokhoz. Modern keretrendszerekkel, mint az Angular és a React való integrációja miatt előnyös választás mind a frontend, mind a backend fejlesztéshez.

- **HTML**: A HTML (HyperText Markup Language) a weboldalak létrehozásának szabványos jelölőnyelve. Strukturálja a webes tartalmat, és elengedhetetlen bármilyen webalkalmazás építéséhez. Más jelölőnyelvekhez képest a HTML-t minden böngésző támogatja, és hatalmas eszköz- és erőforrás-ökoszisztémával rendelkezik, ami nélkülözhetetlenné teszi a frontend fejlesztéshez.
- **CSS**: A CSS (Cascading Style Sheets) a weboldalak stílusának és elrendezésének meghatározására szolgál. Lehetővé teszi a fejlesztők számára, hogy elválasszák a tartalmat a dizájntól, megkönnyítve ezzel a weboldal megjelenésének és érzésének karbantartását és frissítését. Az inline stílusokhoz vagy JavaScript-alapú stílusmegoldásokhoz képest a CSS hatékonyabb és skálázhatóbb módot kínál a stílusok kezelésére nagy projektek esetén.
- **Sass**: A Sass (Syntactically Awesome Style Sheets) egy előfeldolgozó szkriptnyelv, amelyet CSS-re fordítanak vagy értelmeznek. Kiterjeszti a CSS-t olyan funkciókkal, mint a változók, beágyazott szabályok és mixinek, így a stíluslap karbantarthatóbb és újrahasználhatóbb lesz. A sima CSS-hez képest a Sass erősebb és rugalmasabb szintaxist kínál, amely jelentősen felgyorsíthatja a fejlesztési folyamatot.
- **SCSS**: Az SCSS (Sassy CSS) a Sass egy szintaxisa, amely teljesen kompatibilis a CSS-sel. Kombinálja a Sass erejét a CSS ismerős szintaxisával, lehetővé téve a fejlesztők számára, hogy a Sass összes funkcióját használják, miközben CSS-szerű szintaxisban írnak. Más CSS előfeldolgozókhoz képest az SCSS zökkenőmentesebb átmenetet kínál a CSS-t már ismerő fejlesztők számára.
- **SQL**: Az SQL (Structured Query Language) a relációs adatbázisok kezelésének és manipulálásának szabványos nyelve. Lehetővé teszi a fejlesztők számára, hogy különféle műveleteket hajtsanak végre, például lekérdezéseket, frissítéseket és adatkezelést. A NoSQL adatbázisokhoz képest az SQL adatbázisok erős konzisztenciát és összetett lekérdezések támogatását biztosítják, így alkalmasak megbízható tranzakciókat és adatintegritást igénylő alkalmazásokhoz.
- **NestJS** a backend keretrendszerhez: A NestJS egy progresszív Node.js keretrendszer, amely a TypeScript nyelvet használja, és moduláris architektúrát kínál. Lehetővé teszi a skálázható és könnyen karbantartható backend alkalmazások fejlesztését. Versenytársai, mint például az Express.js, szintén népszerűek, de a NestJS strukturáltabb megközelítése és beépített támogatása a modern fejlesztési mintákhoz (mint például a Dependency Injection) miatt előnyösebb választás lehet.

- **Jest**: A Jest egy JavaScript tesztelési keretrendszer, amelyet a Facebook tart fenn, és amely a JavaScript kódbázisok helyességének biztosítására szolgál. Kiváló fejlesztői élményt nyújt olyan funkciókkal, mint a pillanatkép-tesztelés, egy erőteljes mock könyvtár és beépített kódlefedettségi jelentések. Más tesztelési keretrendszerekhez, például a Mocha vagy a Jasmine-hez képest a Jest integráltabb és felhasználóbarátabb élményt kínál, különösen a React alkalmazások esetében.
- **Vue**: A Vue.js egy progresszív JavaScript keretrendszer felhasználói felületek építéséhez. Úgy tervezték, hogy fokozatosan bevezethető legyen, ami azt jelenti, hogy annyit vagy keveset használhat a Vue-ból, amennyire szüksége van. A Vue reaktivitási rendszere és komponens alapú architektúrája megkönnyíti az összetett alkalmazások építését. Az olyan keretrendszerekhez képest, mint az Angular vagy a React, a Vue egyszerűbb és rugalmasabb API-t kínál, amely könnyebben megtanulható és integrálható a meglévő projektekbe.
- **Pinia**: A Pinia egy állapotkezelő könyvtár a Vue.js számára, amelyet a Vuex helyettesítésére terveztek. Egyszerűbb és intuitívabb API-t kínál, megkönnyítve és hatékonyabbá téve az állapotkezelést. A Vuex-hez képest a Pinia jobb TypeScript támogatást és modulárisabb architektúrát biztosít, ami tisztább és karbantarthatóbb kódot eredményezhet.
- **Vuetify**: A Vuetify egy Material Design komponens keretrendszer a Vue.js számára. Széles körű előre tervezett komponenseket biztosít, amelyek követik a Google Material Design irányelveit, megkönnyítve a vizuálisan vonzó és konzisztens felhasználói felületek létrehozását. Más UI keretrendszerekhez, például a Bootstrap vagy a Bulma-hoz képest a Vuetify mélyebb integrációt kínál a Vue-val és átfogóbb komponenskészletet.
- **Playwright**: A Playwright egy végponttól végpontig terjedő tesztelési keretrendszer, amelyet a Microsoft fejlesztett ki. Lehetővé teszi a fejlesztők számára, hogy olyan teszteket írjanak, amelyek szimulálják a felhasználói interakciókat a webalkalmazásokkal, biztosítva, hogy az alkalmazás a vártnak megfelelően viselkedjen. A Seleniumhoz képest a Playwright jobb teljesítményt, megbízhatóbb automatizálást és támogatást kínál a modern webes funkciókhoz, mint például a WebSockets és a szolgáltatási munkavállalók.
- **Vite**: A Vite egy új generációs frontend build eszköz, amely gyors és optimalizált fejlesztési élményeket kínál. Natív ES modulokat és modern böngészőfunkciókat

használ, hogy azonnali hot module replacement-et és villámgyors build-eket biztosítson. A hagyományos bundlerekhez, például a Webpack-hoz képest a Vite jelentősen gyorsabb fejlesztési és build időket kínál, így ideális a modern webfejlesztéshez.

- **Prisma**: A Prisma egy nyílt forráskódú ORM (Object-Relational Mapping) eszköz a Node.js és a TypeScript számára. Egyszerűsíti az adatbázis-hozzáférést és -kezelést egy típusbiztos lekérdező építővel és egy intuitív adatmodellezési nyelvvel. A hagyományos ORM-ekhez, például a Sequelize-hez képest a Prisma jobb TypeScript támogatást, automatikus migrációkat és modernebb API-t kínál, megkönnyítve ezzel az adatbázisok típusbiztos módon történő kezelését.
- **MDI icons**: A Material Design Icons (MDI) egy átfogó ikonkönyvtár, amely a Google Material Design irányelveit követve készült. Széles körű, kiváló minőségű ikonokat biztosít, amelyek könnyen integrálhatók a webalkalmazásokba. Más ikonkönyvtárakhoz, például a Font Awesome-hoz képest az MDI konzisztens és vizuálisan vonzóbb ikonokat kínál, amelyek megfelelnek a modern tervezési elveknek.
- **Handlebars:** A Handlebars egy egyszerű és kiterjeszthető sablonmotor JavaScript-hez. Lehetővé teszi dinamikus HTML sablonok létrehozását logikai utasítások és iterációk segítségével. A sablonokat könnyen olvasható és karbantartható formában írhatjuk meg, mivel a HTML-t és a sablonlogikát elválasztja egymástól. A Handlebars támogatja a részleges sablonokat és a segédfüggvényeket, amelyek lehetővé teszik a sablonok újrafelhasználhatóságát és bővíthetőségét. A Handlebars előnyei közé tartozik a gyors teljesítmény és a széleskörű kompatibilitás különböző környezetekben, beleértve a böngészőket és a Node.js-t is.

Docker Compose fájl

Ez a docker-compose konfigurációs fájl felelős a **Pollák Csengő** alkalmazás megfelelő működéséhez szükséges szolgáltatások, hálózatok és tárolóközegek meghatározásáért és kezeléséért. A dokumentumban definiált beállítások biztosítják a különböző komponensek zökkenőmentes együttműködését, lehetővé téve az alkalmazás gördülékeny telepítését és futtatását konténerizált környezetben. Az alábbiakban részletesen részletes ismertetés olvasható a fájl egyes sorairól, beleértve a konfigurációk jelentését és azok működését a rendszer egészének kontextusában.

Szolgáltatások

1. **csengo-v2-postgres:**
   - **`container_name:`** `csengo-v2-postgres`: Beállítja a konténer nevét `csengo-v2-postgres`-ra.
   - **`image:`** `postgres:16.6`: A PostgreSQL 16.6 verzióját használja.
   - **`restart:`** `always`: Biztosítja, hogy a konténer mindig újrainduljon, ha leáll.
   - **`environment`**: Környezeti változókat állít be a PostgreSQL konténerhez:
     - **`POSTGRES_USER:`** `csengo`: Beállítja a PostgreSQL felhasználót `csengo`-ra.
     - **`POSTGRES_PASSWORD:`** `csengo`: Beállítja a PostgreSQL jelszót `csengo`-ra.
     - **`POSTGRES_DB:`** `csengo`: Beállítja a PostgreSQL adatbázis nevét `csengo`-ra.
   - **`volumes:`**
     - **`csengo-v2_db`**`:/var/lib/postgresql/data`: Csatolja a `csengo-v2_db` kötetet a PostgreSQL adatainak megőrzéséhez.
   - **`networks:`**
     - **`csengo-v2`**: Csatlakoztatja a konténert a `csengo-v2` hálózathoz.

2. **csengo-ts-server-v2:**
   - **`container_name:`** `csengo-ts-server-v2`: Beállítja a konténer nevét `csengo-ts-server-v2`-ra.
   - **`build`**: A konténer felépítéséhez használt Dockerfile:
     - **`context:`** `./csengo-ts-server-v2`: A build kontextus a `csengo-ts-server-v2` könyvtár.
     - **`dockerfile:`** `Dockerfile`: A Dockerfile elérési útja.
   - **`image:`** `csengo-ts-server-v2`: A konténer képének neve.
   - **`restart:`** `always`: Biztosítja, hogy a konténer mindig újrainduljon, ha leáll.
   - **`volumes:`**
     - **`csengo-v2_sounds`**`:/data`: Csatolja a `csengo-v2_sounds` kötetet a konténer `/data` könyvtárához.
   - **`ports:`**
     - **`"3300:3300"`**: A 3300-as portot a gazdagépen a 3300-as portra térképezi a konténerben.
   - **`depends_on`**: Meghatározza a szolgáltatás függőségeit:
     - `csengo-v2-postgres`
   - **`networks:`** `- csengo-v2`: Csatlakoztatja a konténert a `csengo-v2` hálózathoz.

- **environment**: Környezeti változókat állít be a TypeScript szerverhez:
  - **PORT:** 3300: Beállítja a szerver portját 3300-ra.
  - **DATABASE_URL:** "postgresql://csengo:csengo@csengo-v2-postgres:5432/csengo?schema=public": Beállítja az adatbázis kapcsolat URL-jét.
  - **UPLOAD_PATH:** "./data/audio": Beállítja a hangfájlok feltöltési útvonalát.
  - **TOKEN_SECRET:** "token-secret": Beállítja a token titkot az autentikációhoz.
  - **JWT_SECRET:** "jwt-secret": Beállítja a JWT titkot az autentikációhoz.
  - **CORS_ORIGIN:** "*": Beállítja a CORS eredetet.
  - **CORS_DOMAIN:** "localhost": Beállítja a CORS domaint.
  - **DEV:** true: Beállítja a fejlesztési módot igazra.
  - **WS_API_KEY:** "ws-api-key": Beállítja a WebSocket API kulcsot.

3. **csengo-ts-client-v2:**
- **container_name:** csengo-ts-client-v2: Beállítja a konténer nevét csengo-ts-client-v2-ra.
- **build**: A konténer felépítéséhez használt Dockerfile:
  - **context:** ./csengo-ts-client-v2: A build kontextus a csengo-ts-client-v2 könyvtár.
  - **dockerfile:** Dockerfile: A Dockerfile elérési útja.
  - **target:** prod: A build célja a prod.
  - **args**: Build argumentumok:
    - VITE_API_URL=http://localhost:3300
    - VITE_COOKIE_DOMAIN=localhost
- **image:** csengo-ts-client-v2: A konténer képének neve.
- **ports:**
  - **"8080:80"**: A 8080-as portot a gazdagépen a 80-as portra térképezi a konténerben.
- **networks:**
  - csengo-v2: Csatlakoztatja a konténert a csengo-v2 hálózathoz.

4. **csengo-v2-pgadmin**:
- **container_name:** csengo-v2-pgadmin: Beállítja a konténer nevét csengo-v2-pgadmin-ra.
- **image:** dpage/pgadmin4: A pgAdmin 4 képét használja.
- **restart:** always: Biztosítja, hogy a konténer mindig újrainduljon, ha leáll.
- **ports:**
  - **"8081:80":** A 8081-es portot a gazdagépen a 80-as portra térképezi a konténerben.
- **environment**: Környezeti változókat állít be a pgAdmin-hoz:
  - **PGADMIN_DEFAULT_EMAIL:** csengo@csengo.dev: Beállítja az alapértelmezett email címet a pgAdmin-hoz.
  - **PGADMIN_DEFAULT_USERNAME:** admin: Beállítja az alapértelmezett felhasználónevet a pgAdmin-hoz.
  - **PGADMIN_DEFAULT_PASSWORD:** admin: Beállítja az alapértelmezett jelszót a pgAdmin-hoz.
  - **PGADMIN_CONFIG_WTF_CSRF_ENABLED:** "False": Letiltja a CSRF védelmet.

- `volumes:`
  - **`csengo-v2_pgadmin:`** `/var/lib/pgadmin`: Csatolja a `csengo-v2_pgadmin` kötetet a pgAdmin adatainak megőrzéséhez.
- `networks:`
  - **`csengo-v2`**: Csatlakoztatja a konténert a `csengo-v2` hálózathoz.

Hálózatok

- **`csengo-v2`**: Meghatároz egy egyedi hálózatot `csengo-v2` néven, amelyen a szolgáltatások kommunikálhatnak egymással.

Kötetek

- **`csengo-v2_db`**: Meghatároz egy kötetet `csengo-v2_db` néven a PostgreSQL adatok megőrzéséhez.
- **`csengo-v2_sounds`**: Meghatároz egy kötetet `csengo-v2_sounds` néven a hangadatok megőrzéséhez.
- **`csengo-v2_pgadmin`**: Meghatároz egy kötetet `csengo-v2_pgadmin` néven a pgAdmin adatok megőrzéséhez.

Adatbázis kezelő

Az adatbázis kezelő az alábbi alábbi módókon érhető el a futtatás környezetétől függően:

- **Docker nélküli futtatás esetén:** A telepitésre került PgAdmin alkalmazásban.
- **Dockerrel történő futtatás esetén:** A „`http://localhost:8081`” címen.

Kezelés Docker nélkül

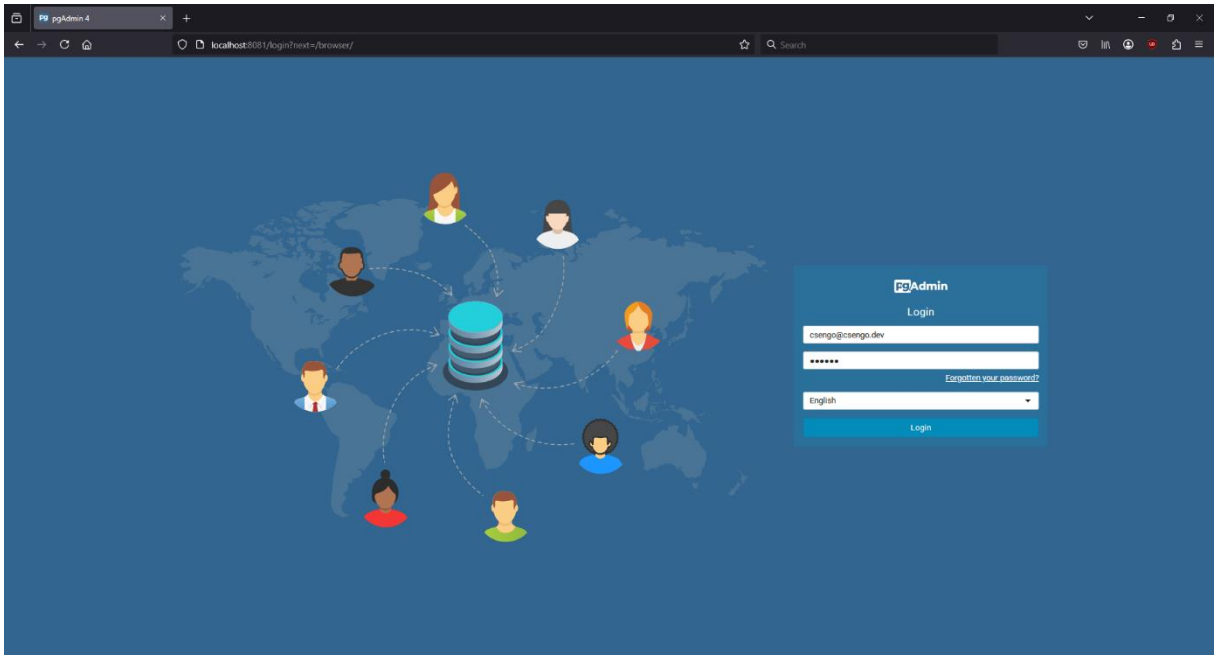
Amennyiben nem docker segítségével indította az alkalmazást, a lokálisan telepített PgAdmin alkalmazás segítsévégel tudja kezelni az adatbázist.

Ennek a kinézete és a használatának módja is megegyezik a webfelületes kezelőpanellel, éltérést csak a lokálisan használt adatbázisok okozhatnak. Képes illusztrációkat is találhat az alábbi „Kezelés Dockerrel” részlegben.

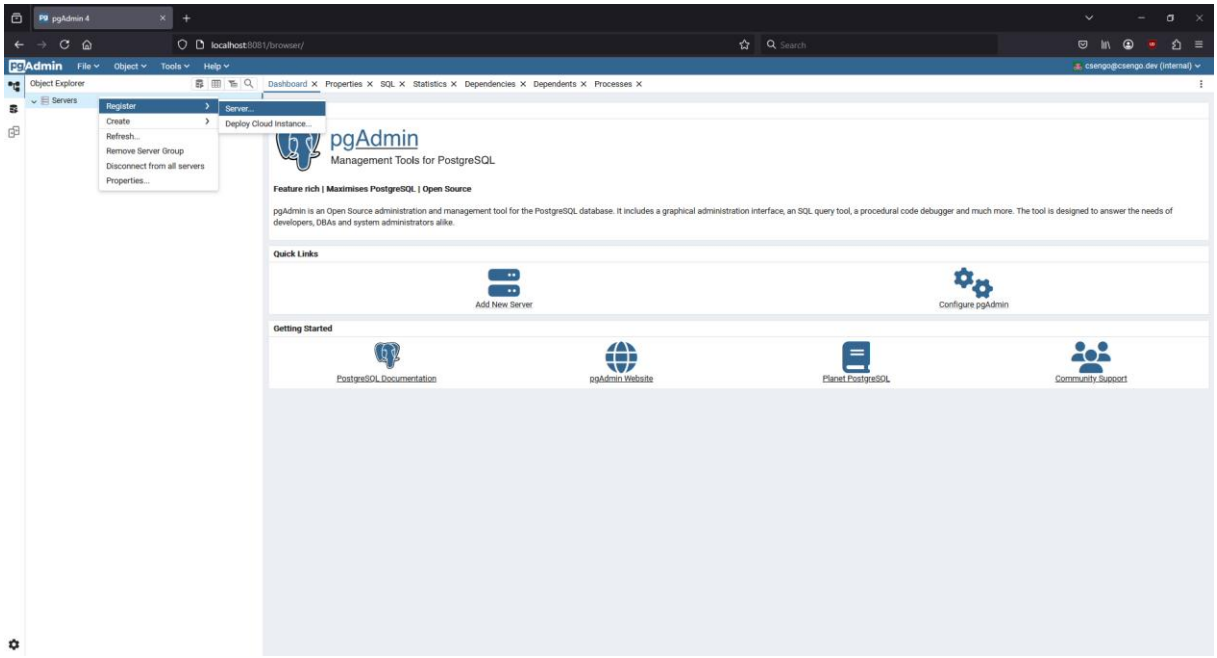
Kezelés Dockerrel

Az adatbázis kezelőhöz bejelentkezéshez használja az alábbi adatokat:

- Email: `csengo@csengo.dev`
- Felhasználónév: `admin`
- Jelszó: `admin`

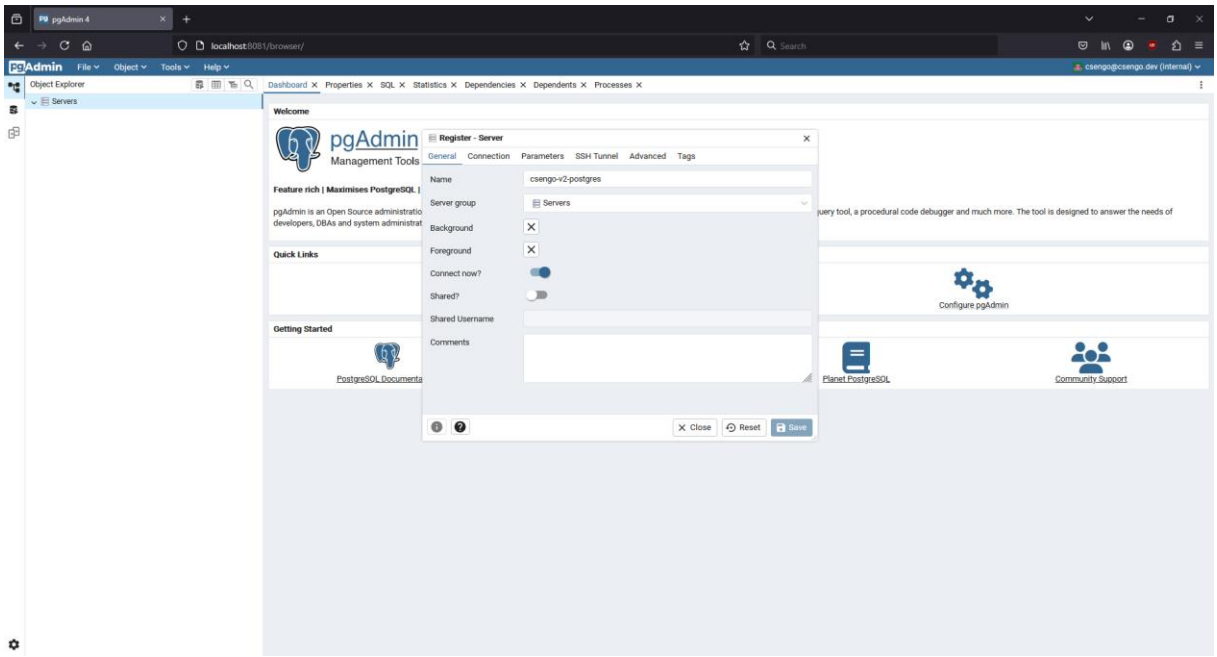


Az pgAdmin adatbázis kezelőn belül a következő módon tudunk kapcsolódni az adatbázishoz:



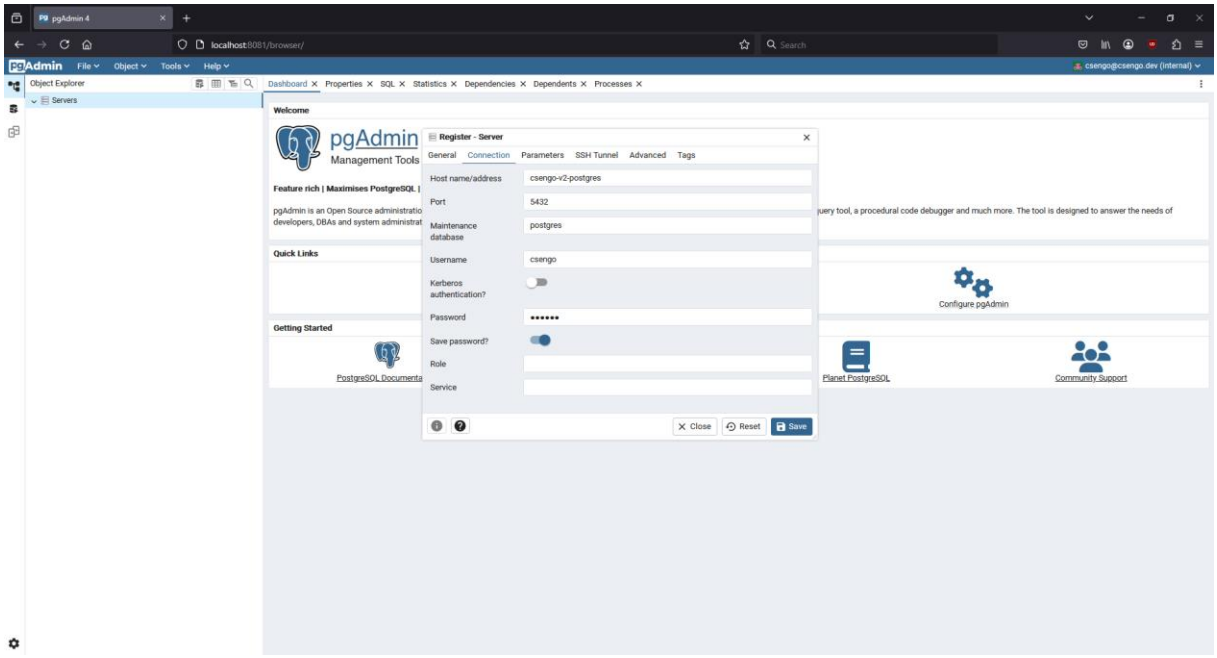
A következő adatokat kell megadni a kapcsolódás elnevezéséhez a következő módon:

- Name: `csengo-v2-postgres`

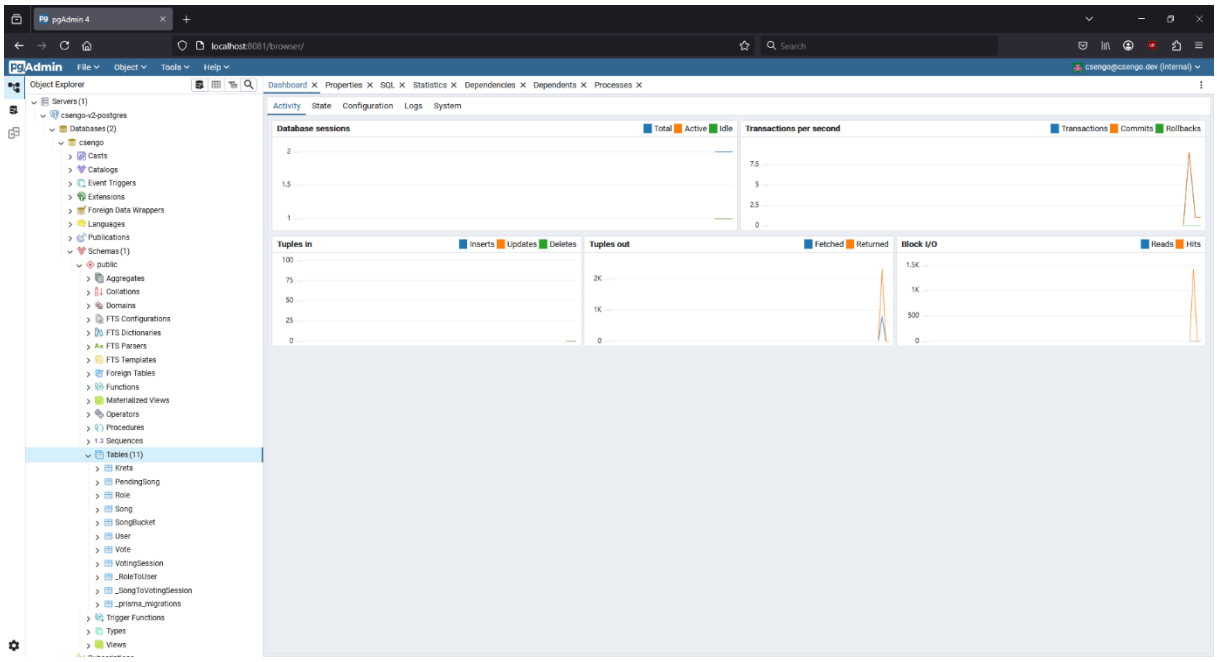


A következő adatokat kell megadni a kapcsolódáshoz a következő módon:

- Host name/address: `csengo-v2-postgres`
- Port: `5432`
- Maintenance database: `postgres`
- Username: `csengo`
- Password: `csengo`
- Save password: `True`



A következő módon lehet az adatbázisban tárolt táblákat megtekinteni:



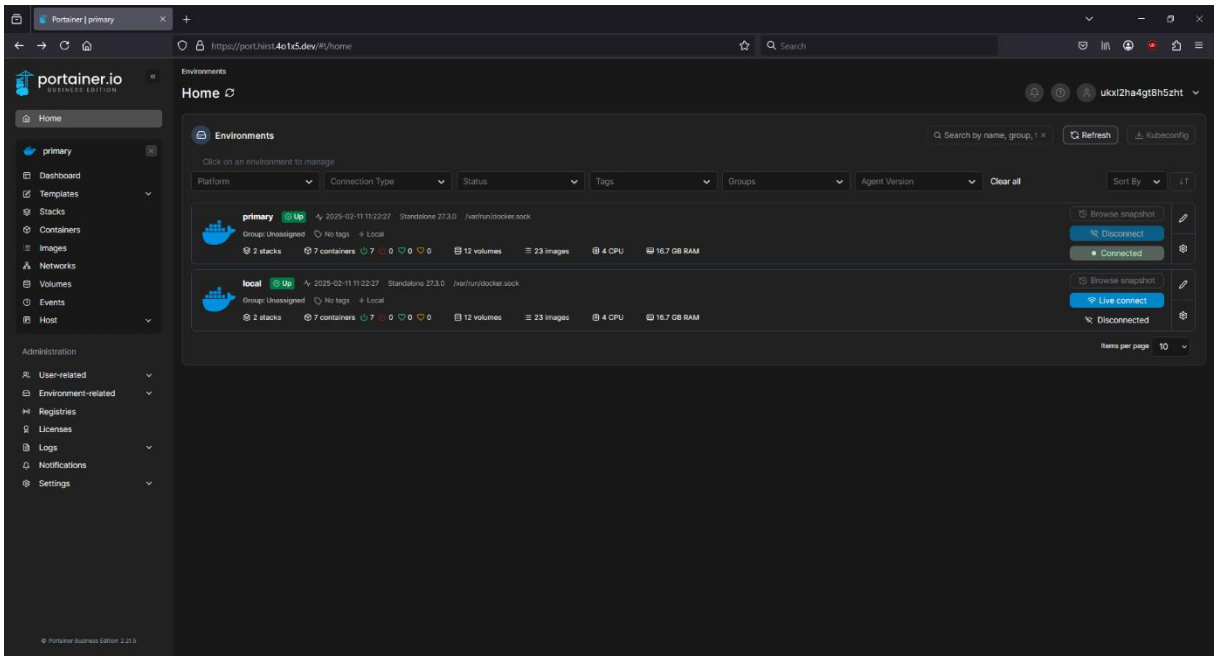
Portainer-en keresztüli konténer kezelés

Ezt a technológiát használtuk a fejlesztés során. Ebben a részlegben megismerheti a működését és a használatának a célját.

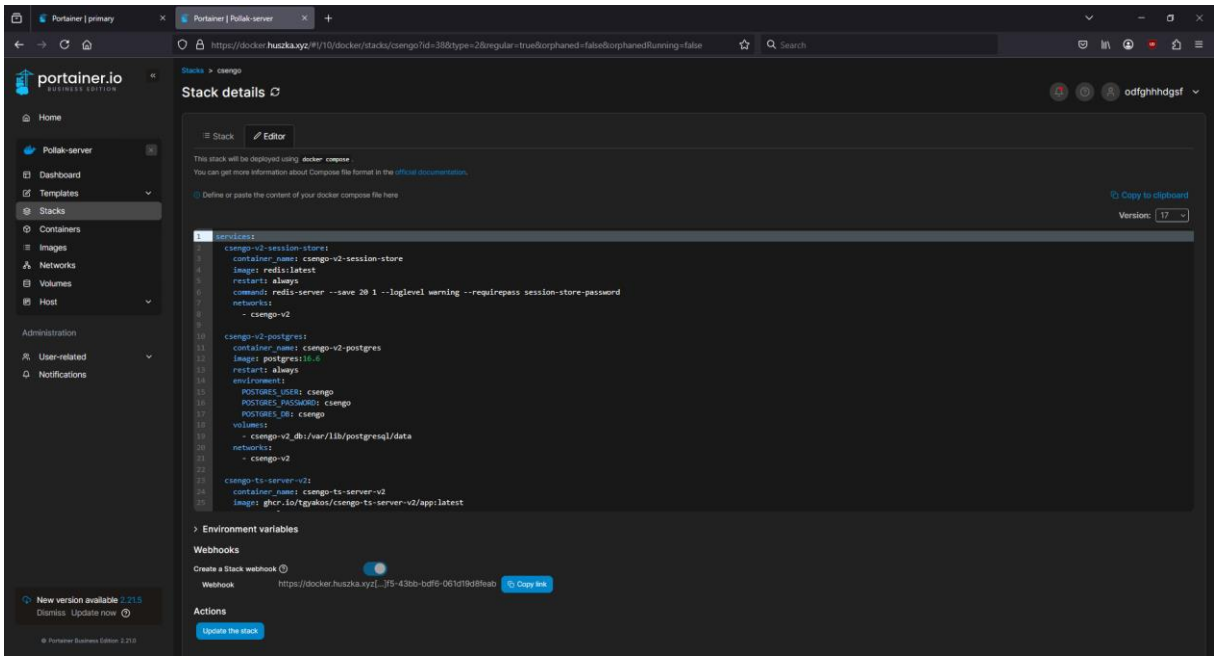
A Portainer egy könnyen használható, webes felülettel rendelkező konténerkezelő eszköz, amely lehetővé teszi a konténerek kezelését, figyelését és felügyeletét. A Portainer segítségével egyszerűen létrehozhat, indíthat, leállíthat és törölhet konténereket, valamint ellenőrizheti a konténerek állapotát és naplóit.

Ez a dokumentáció feltételezi, hogy egy Portainer példány fut a localhost:9000 címen.

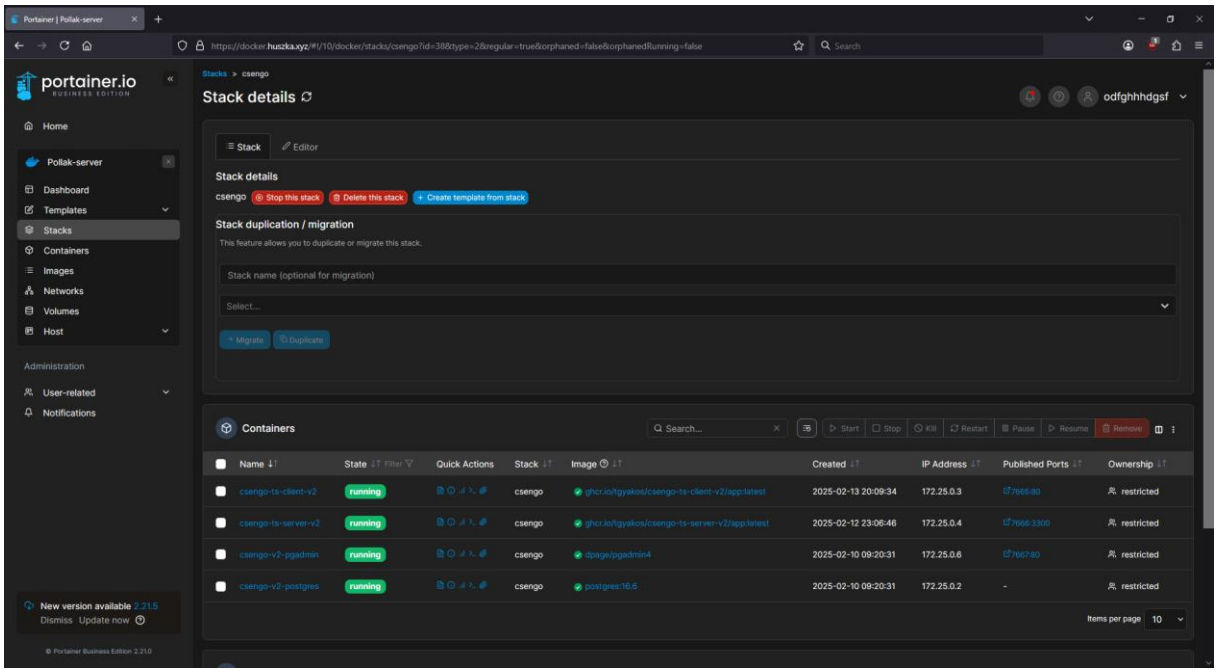
1. Nyissa meg a böngészőt, és navigáljon a http://localhost:9000 címre.
2. Jelentkezzen be a Portainerbe a felhasználónév és jelszó megadásával.
3. Az ábra alapján válassza ki a számára megfelelő Docker környezetet.



4. Készítsen elő egy új Stack-et a Stacks fül alatt, majd másolja bele a docker-compose.dev.v2.yml fájl tartalmát. Valamint adja hozzá a megfelelő fájlokat, hogy a konténerek le tudjanak épülni.



5. Sikeres stack létrehozása után a konténerek automatikusan elindulnak, és a Portainer segítségével kezelheti őket.



Futtatás fejlesztői módban

A Pollák Csengő alkalmazás fejlesztői módban való futtatásához a következő lépéseket kell követnie:

Szükséges szoftverek telepítése

- Telepítse a Docker Desktop nevű alkalmzást és amennyiben nem Windows operáriós rendszert használ, a Docker-compose nevű alkalmazást is a számítógépére, majd indítsa el a Docker Desktop alkalmazást.

Backend elindítása

1. Amennyiben szükséges változtassa meg a .env.example fájlban található környezeti változókat az ön által kívántakra.
2. Nyissa meg a terminált a `csengo-ts-server-v2` mappában.
3. Windows operációs rendszeren futtassa a következő parancsokat a szerver indításához:

```
docker-compose -f docker-compose.dev.yml up csengo-v2-postgres-dev -d
copy .env.example .env
npm install
npm run prisma:update:prod
npm run start:dev
```

4. A szerver elindítása után a következő URL-ek érhetőek el:
   - Swagger dokumentáció: http://localhost:3300/swagger
   - REST API: http://localhost:3300/api

Frontend elindítása

1. Amennyiben szükséges változtassa meg a .env.example fájlban található környezeti változókat az ön által kívántakra
2. Nyissa meg a terminált a `csengo-ts-client-v2` mappában.
3. Windows operációs rendszeren futtassa a következő parancsokat a kliens indításához:

```
copy .env.example .env
npm install
npm run dev
```

4. A kliens elindítása után a következő URL-ek érhetőek el:
   - Weboldal: http://localhost:3000

Fejlesztői admin fiók regisztrálása

Ha a backend szerver elindult, akkor a Swagger dokumentációban regisztrálhat egy új fejlesztői adminisztrátor fiókot a következő lépésekkel:

1. Nyissa meg a Swagger dokumentációt a http://localhost:3300/swagger URL-en.
2. Kattintson a **`POST /api/auth/register`**-dev útvonalra.
3. Kattintson a **`Try it out`** gombra.
4. Adja meg a következő adatokat az adat testben:
   - **`username`**: Az új fejlesztői adminisztrátor felhasználóneve.
   - **`email`**: Az új fejlesztői adminisztrátor e-mail címe.
   - **`password`**: Az új fejlesztői adminisztrátor jelszava. Legalább 6 karakter hosszú kell legyen.
   - **`om`**: Egy egyedi OM azonosító a fejlesztői adminisztrátorhoz. Egy szám amely maximum 11 karakter hosszú lehet. Minden lekérdezéshez egyedinek kell lennie.
5. Kattintson a **`Execute`** gombra.
6. Jelentkezzen be az alkalmazásba a http://localhost:3000/login úton, a regisztrált fejlesztői adminisztrátor fiókkal.

Strukturális felépítés

Frontend felépítése

A Pollák Csengő frontend alkalmazásának metodológiája a következő: minden komponens egyedi felelősségű, és egyedi funkcionalitással rendelkezik. A komponensek egymásra épülnek, és a kisebb komponensek nagyobb komponensekbe vannak beágyazva. A komponensek egymással kommunikálnak a Pinia állapotkezelő segítségével, valamint egy egyedi event busz használatával, és a komponensek közötti navigáció a Vue Router segítségével történik. Minden lekérdezés egy Pinia store-ban történik, hogy a komponensek egyszerű és konzisztens állapotot tarthassanak. A komponensek a Vuetify keretrendszer segítségével vannak stilizálva, és a Vite build eszköz segítségével kerülnek összeállításra. A frontend strukturális felépítése a következő:

- **.github**
  - *workflows*: Tartalmazza a GitHub Actions munkafolyamatokat.
    - *ci.yml*: A folyamatos integrációs munkafolyamat konfigurációja.
- **.idea**
  - *codeStyles*: Kódstílus beállítások.
    - *codeStyleConfig.xml*: Kódstílus konfiguráció.
    - *Project.xml*: Projekt szintű kódstílus beállítások.
  - *inspectionProfiles*: Ellenőrzési profilok.
    - *Project_Default.xml*: Alapértelmezett ellenőrzési profil.
  - *runConfigurations*: Futtatási konfigurációk.
    - *All_Tests.xml*: Összes teszt futtatási konfigurációja.
    - *dev.xml*: Fejlesztési futtatási konfiguráció.
  - *csengo-ts-client-v2.iml*: Projekt fájl.
  - *discord.xml*: Discord integrációs beállítások.
  - *git_toolbox_blame.xml*: Git blame eszköz beállításai.
  - *git_toolbox_prj.xml*: Git projekt eszköz beállításai.
  - *modules.xml*: Modulok konfigurációja.
  - *vcs.xml*: Verziókezelő rendszer beállításai.
  - *watcherTasks.xml*: Felügyeleti feladatok beállításai.
- **.vscode**
  - *extensions.json*: Ajánlott kiterjesztések listája.
  - *settings.json*: VSCode beállítások.
- **config**
  - *nginx.conf*: Nginx konfigurációs fájl.
- **public**
  - *favicon.ico*: Weboldal ikon fájl.
- **src**
  - *App.vue*: Az alkalmazás fő komponense.
  - *auto-imports.d.ts*: Automatikus importok típusdefiníciói.

- *components.d.ts*: Komponensek típusdefiníciói.
- *main.ts*: Az alkalmazás belépési pontja.
- *typed-router.d.ts*: Típusdefiníciók a routerhez.
- **assets**: Statikus fájlok és képek.
  - *background.jpg*: Háttérkép.
  - *pollakLogo.png*: Logó kép.
- **components**: Az alkalmazás különböző komponensei.
  - *LoginCard.vue*: Bejelentkezési kártya komponens.
  - *RegisterCard.vue*: Regisztrációs kártya komponens.
  - **admin**: Adminisztrációs felület komponensei.
    - *Download.vue*: Letöltési komponens.
    - **pending-song**: Függő zenék kezelése.
      - *PendingSong.vue*: Függő zene komponens.
      - *PendingSongList.vue*: Függő zenék listája.
    - **session**: Szavazási szekciók kezelése.
      - *CreateSessionPopup.vue*: Szavazási szekció létrehozása.
      - *Session.vue*: Szavazási szekció komponens.
      - *SessionList.vue*: Szavazási szekciók listája.
      - *UpdateSessionPopup.vue*: Szavazási szekció frissítése.
      - *ViewSessionSongsPopup.vue*: Szavazási szekció zenéinek megtekintése.
    - **song**: Zenék kezelése.
      - *CreateSongPopup.vue*: Zene létrehozása.
      - *Song.vue*: Zene komponens.
      - *SongList.vue*: Zenék listája.
      - *UpdateSongPopup.vue*: Zene frissítése.
    - **user**: Felhasználók kezelése.
      - *UpdateUserPopup.vue*: Felhasználó frissítése.
      - *User.vue*: Felhasználó komponens.
      - *UserList.vue*: Felhasználók listája.
  - **common**: Közös komponensek.
    - *Toast.vue*: Értesítési komponens.
  - **dashboard**: Felhasználói irányítópult komponensei.
    - *WelcomeSign.vue*: Üdvözlő jelzés.
    - **left-card**: Bal oldali kártya komponensei.
      - *LeftLandingCard.vue*: Bal oldali kártya fő komponense.
      - *SongUpload.vue*: Zene feltöltése.
      - *SongVote.vue*: Zene szavazás.
      - *SongVoteList.vue*: Szavazási zenék listája.
      - *UploadInput.vue*: Feltöltési bemenet.
    - **right-card**: Jobb oldali kártya komponensei.
      - *PreviousWinner.vue*: Előző nyertes.
      - *RightLandingCard.vue*: Jobb oldali kártya fő komponense.
      - *SongSelectionList.vue*: Zene kiválasztási lista.
      - **song-selection**: Zene kiválasztási komponensek.
        - *PlaySong.vue*: Zene lejátszása.
        - *SongSelection.vue*: Zene kiválasztása.
        - *VoteSong.vue*: Zene szavazása.
  - **snipper**: YouTube vágó komponensek.

        - *SnipperPanel.vue*: YouTube vágó komponens.
- **layouts**: Oldal elrendezések.
    - *landing.vue*: Kezdőoldal elrendezés.
    - *user.vue*: Felhasználói elrendezés.
    - *snipper.vue*: YouTube vágó oldal elrendezés.
- **pages**: Oldalak.
    - *forbidden.vue*: Tiltott oldal.
    - *index.vue*: Főoldal.
    - *login.vue*: Bejelentkezési oldal.
    - *register.vue*: Regisztrációs oldal.
    - *test.vue*: Teszt oldal.
    - *tv.vue*: TV oldal.
    - *[...slug].vue*: Dinamikus útvonalak kezelése.
    - *admin.vue*: Adminisztrációs oldal.
    - *snipper.vue*: YouTube vágó oldal.
- **plugins**: Bővítmények.
    - *index.ts*: Bővítmények regisztrálása.
    - *vuetify.ts*: Vuetify bővítmény konfiguráció.
- **router**: Útvonalak kezelése.
    - *index.ts*: Útvonalak konfigurációja.
- **stores**: Állapotkezelők.
    - *index.ts*: Állapotkezelők regisztrálása.
    - *tv.ts*: TV állapotkezelő.
    - *login.ts*: Bejelentkezés állapotkezelő.
    - *register.ts*: Regisztráció állapotkezelő.
    - **admin**: Adminisztrációs állapotkezelők.
        - **pending-song**: Függő zenék állapotkezelői.
            - *pendingSong.ts*: Függő zene állapotkezelő.
            - *pendingSongList.ts*: Függő zenék listájának állapotkezelője.
        - **session**: Szavazási szekciók állapotkezelői.
            - *createSessionPopup.ts*: Szavazási szekció létrehozása állapotkezelő.
            - *session.ts*: Szavazási szekció állapotkezelő.
            - *sessionList.ts*: Szavazási szekciók listájának állapotkezelője.
            - *updateSessionPopup.ts*: Szavazási szekció frissítése állapotkezelő.
        - **song**: Zenék állapotkezelői.
            - *createSongPopup.ts*: Zene létrehozása állapotkezelő.
            - *song.ts*: Zene állapotkezelő.
            - *songList.ts*: Zenék listájának állapotkezelője.
            - *updateSongPopup.ts*: Zene frissítése állapotkezelő.
        - **timetable**: Időbeosztás állapotkezelői.
            - *timeList.ts*: Időbeosztás állapotkezelő.
        - **user**: Felhasználók állapotkezelői.
            - *updateUserPopup.ts*: Felhasználó frissítése állapotkezelő.
            - *user.ts*: Felhasználó állapotkezelő.
        - *download.ts*: Letöltés állapotkezelő.
    - **dashboard**: Felhasználói irányítópult állapotkezelői.
        - *welcomeSign.ts*: Üdvözlő jelzés állapotkezelő.
        - **left-card**: Bal oldali kártya állapotkezelői.
            - *songVoteList.ts*: Szavazási zenék listájának állapotkezelője.

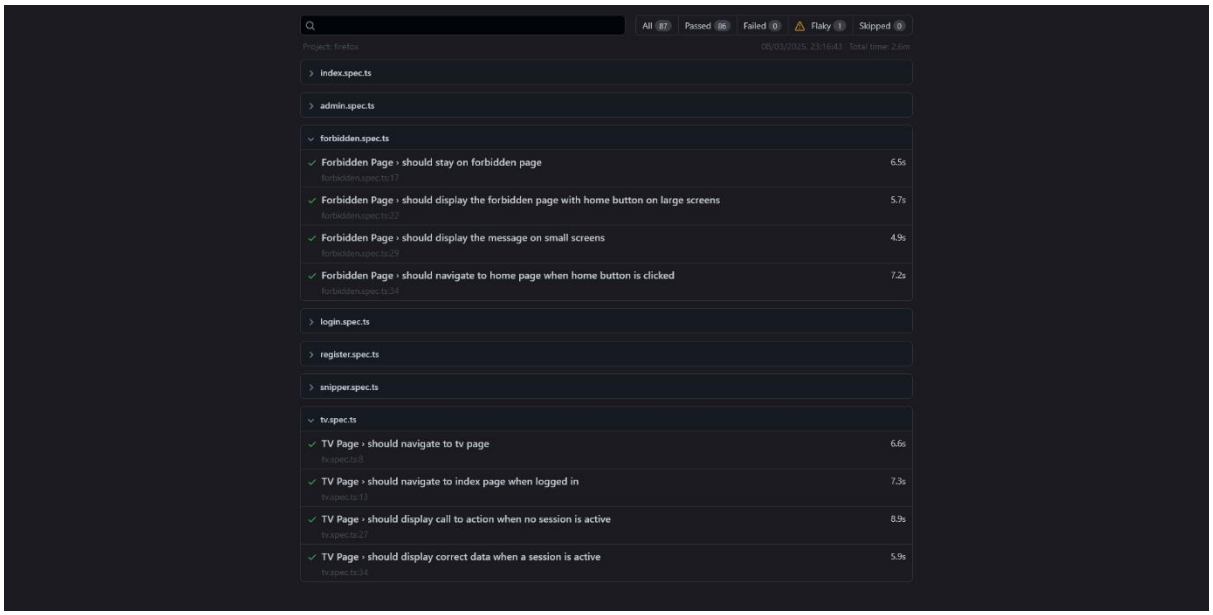
        - *uploadInput.ts*: Feltöltési bemenet állapotkezelő.
      - **right-card**: Jobb oldali kártya állapotkezelői.
        - *previousWinner.ts*: Előző nyertes állapotkezelő.
        - *songSelectionList.ts*: Zene kiválasztási lista állapotkezelő.
        - **song-selection**: Zene kiválasztási állapotkezelők.
          - *playSong.ts*: Zene lejátszása állapotkezelő.
          - *voteSong.ts*: Zene szavazása állapotkezelő.
      - **snipper**: YouTube vágó állapotkezelői.
        - *uploadSong.ts*: YouTube vágó állapotkezelő.
  - **styles**: Stílusok.
    - *login.scss*: Bejelentkezési oldal stílusai.
    - *register.scss*: Regisztrációs oldal stílusai.
    - *settings.scss*: Beállítások stílusai.
  - **types**: Típusdefiníciók.
    - *jwt.d.ts*: JWT típusdefiníciók.
    - *role.enum.d.ts*: Szerepkörök típusdefiníciói.
    - *session.ts*: Szavazási szekció típusdefiníciók.
    - *toast.enum.ts*: Értesítési típusok.
  - **utils**: Segédfüggvények.
    - *eventBus.util.ts*: Eseménybusz segédfüggvény.
    - *logger.custom.util.ts*: Egyedi naplózó segédfüggvény.
- **tests**
  - *admin.spec.ts*: Adminisztrációs tesztek.
  - *forbidden.spec.ts*: Tiltott oldal tesztje.
  - *index.spec.ts*: Főoldal tesztje.
  - *login.spec.ts*: Bejelentkezési oldal tesztje.
  - *register.spec.ts*: Regisztrációs oldal tesztje.
  - *snipper.spec.ts*: YouTube vágó oldal tesztje.
  - *tsconfig.json*: Teszt konfiguráció.
  - *tv.spec.ts*: TV oldal tesztje.
- *.browserslistrc*: A böngészők támogatásának konfigurációs fájlja.
- *.dockerignore*: A Docker által figyelmen kívül hagyandó fájlok listája.
- *.editorconfig*: Kódformázási szabályok különböző szerkesztők és IDE-k számára.
- *.env.example*: Példa környezeti változók fájlra.
- *.eslintrc-auto-import.json*: ESLint konfigurációs fájl automatikus importáláshoz.
- *.gitignore*: A Git által figyelmen kívül hagyandó fájlok listája.
- *.prettierrc*: Prettier kódformázó eszköz konfigurációs fájlja.
- *docker-compose.dev.yml*: Docker Compose konfiguráció fejlesztési környezethez.
- *Dockerfile*: Docker kép építéséhez szükséges utasításokat tartalmazó fájl.
- *env.d.ts*: TypeScript típusdefiníciók környezeti változókhoz.
- *eslint.config.js*: ESLint konfigurációs fájl.
- *index.html*: Az alkalmazás belépési pontja HTML formátumban.
- *package-lock.json*: A projekt függőségeinek pontos verzióit rögzítő fájl.
- *package.json*: A projekt függőségeit és szkripteket tartalmazó fájl.
- *playwright.config.ts*: Playwright tesztelési keretrendszer konfigurációs fájlja.
- *sonar-project.properties*: SonarQube projekt konfigurációs fájl.
- *tsconfig.app.json*: TypeScript konfigurációs fájl az alkalmazás számára.
- *tsconfig.json*: A TypeScript projekt alapértelmezett konfigurációs fájlja.
- *tsconfig.node.json*: TypeScript konfigurációs fájl Node.js projektekhez.

- *tsconfig.vitest.json*: TypeScript konfigurációs fájl Vitest tesztelési keretrendszerhez.
- *vite.config.mts*: Vite konfigurációs fájl.

Frontend Tesztelés

Az alábbi szakaszban bemutatjuk a projekt ellenőrzésére használt frontend teszteket. Részletesen ismertetjük az egyes fájlokat és a bennük található teszteseteket. A webalkalmazás minden oldalát alapos tesztelésnek vetettük alá, amely biztosítja, hogy az összes funkció megfelelően és hibamentesen működik.

Az alábbi képen látható az összes Frontend teszt lefutásának összegzése, valamint néhány teszt lefutásának eredménye:



A frontend tesztelési folyamatok a következők:

`admin.spec.ts`

Ez a tesztfájl felelős az Admin oldal helyes megjelenítéséért és zavartalan működéséért. Minden funkciót alaposan teszteltünk, hogy biztosítsuk a problémamentes használatot a jövőbeli felhasználók számára. Az alábbiakban megtalálhatók a pontos tesztek és azok céljai:

- **should display the admin page**: Ellenőrzi, hogy az admin oldal megjelenik-e.
- **should display home button**: Ellenőrzi, hogy a kezdőlap gomb megjelenik-e.
- **should display dropdown menu**: Ellenőrzi, hogy a legördülő menü megjelenik-e.
- **should display logout option when dropdown menu is pressed**: Ellenőrzi, hogy a kijelentkezés opció megjelenik-e, ha a legördülő menüt megnyomják.

- **should go to pending song when pending song is pressed**: Ellenőrzi, hogy a függőben lévő zenék oldalra navigál-e a felhasználó, ha a megfelelő gombot megnyomják.
- **should go to votes when votes is pressed**: Ellenőrzi, hogy a szavazások oldalra navigál-e a felhasználó, ha a megfelelő gombot megnyomják.
- **should go to miscellaneous when miscellaneous is pressed**: Ellenőrzi, hogy az egyebek oldalra navigál-e a felhasználó, ha a megfelelő gombot megnyomják.
- **should display no data on song page when no data is present**: Ellenőrzi, hogy a zenék oldalon nincs adat megjelenik-e, ha nincs adat.
- **should display no data on pending song page when no data is present**: Ellenőrzi, hogy a függőben lévő zenék oldalon nincs adat megjelenik-e, ha nincs adat.
- **should display no data on votes page when no data is present**: Ellenőrzi, hogy a szavazások oldalon nincs adat megjelenik-e, ha nincs adat.
- **should display correct data on song page when its present**: Ellenőrzi, hogy a zenék oldalon helyesen jelenik-e meg az adat, ha van adat.
- **should starting playing audio on song page when play button is pressed**: Ellenőrzi, hogy a zenék oldalon elindul-e a lejátszás, ha a lejátszás gombot megnyomják.
- **should stop playing audio on song page when play button is pressed twice**: Ellenőrzi, hogy a zenék oldalon leáll-e a lejátszás, ha a lejátszás gombot kétszer megnyomják.
- **should upload new song on song page when upload button is pressed**: Ellenőrzi, hogy a zenék oldalon feltölthető-e egy új zene, ha a feltöltés gombot megnyomják.
- **should rename song on song page when rename button is pressed**: Ellenőrzi, hogy a zenék oldalon átnevezhető-e egy zene, ha az átnevezés gombot megnyomják.
- **should delete song on song page when delete button is pressed**: Ellenőrzi, hogy a zenék oldalon törölhető-e egy zene, ha a törlés gombot megnyomják.
- **should go to pending song page when pending song button is pressed**: Ellenőrzi, hogy a függőben lévő zenék oldalra navigál-e a felhasználó, ha a megfelelő gombot megnyomják.
- **should display correct data on pending song page when its present**: Ellenőrzi, hogy a függőben lévő zenék oldalon helyesen jelenik-e meg az adat, ha van adat.
- **should play audio on pending song page when play button is pressed**: Ellenőrzi, hogy a függőben lévő zenék oldalon elindul-e a lejátszás, ha a lejátszás gombot megnyomják.
- **should stop playing audio on pending song page when play button is pressed twice**: Ellenőrzi, hogy a függőben lévő zenék oldalon leáll-e a lejátszás, ha a lejátszás gombot kétszer megnyomják.
- **should approve song on pending song page when approve button is pressed**: Ellenőrzi, hogy a függőben lévő zenék oldalon engedélyezhető-e egy zene, ha az engedélyezés gombot megnyomják.
- **should disapprove song on pending song page when disapprove button is pressed**: Ellenőrzi, hogy a függőben lévő zenék oldalon elutasítható-e egy zene, ha az elutasítás gombot megnyomják.
- **should go to votes page when votes button is pressed**: Ellenőrzi, hogy a szavazások oldalra navigál-e a felhasználó, ha a megfelelő gombot megnyomják.

- **should display correct data on votes page when its present**: Ellenőrzi, hogy a szavazások oldalon helyesen jelenik-e meg az adat, ha van adat.
- **should display all part taking songs when view button is pressed**: Ellenőrzi, hogy a szavazások oldalon megjelenik-e az összes résztvevő zene, ha a megtekintés gombot megnyomják.
- **should edit all data when edit button is pressed**: Ellenőrzi, hogy a szavazások oldalon szerkeszthető-e az összes adat, ha a szerkesztés gombot megnyomják.
- **should delete voting session on votes page when delete button is pressed**: Ellenőrzi, hogy a szavazások oldalon törölhető-e egy szavazási szakasz, ha a törlés gombot megnyomják.
- **should create new voting session on votes page when create button is pressed**: Ellenőrzi, hogy a szavazások oldalon létrehozható-e egy új szavazási szakasz, ha a létrehozás gombot megnyomják.
- **should go to users page when users button is pressed**: Ellenőrzi, hogy a felhasználó a felhasználók oldalra navigál-e, ha a megfelelő gombot megnyomja.
- **should display correct data on users page when its present**: Ellenőrzi, hogy a felhasználók oldalon helyesen jelenik-e meg az adat, ha van adat.
- **should edit user's password on users page when edit button is pressed and only password is filled**: Ellenőrzi, hogy a felhasználók oldalon szerkeszthető-e a felhasználó jelszava, ha csak a jelszó mezőt töltik ki és a szerkesztés gombot megnyomják.
- **should edit user's role on users page when edit button is pressed and only role is filled**: Ellenőrzi, hogy a felhasználók oldalon szerkeszthető-e a felhasználó szerepköre, ha csak a szerepkör mezőt töltik ki és a szerkesztés gombot megnyomják.
- **should edit user's role and password on users page when edit button is pressed and both are filled**: Ellenőrzi, hogy a felhasználók oldalon szerkeszthető-e a felhasználó szerepköre és jelszava, ha mindkét mezőt kitöltik és a szerkesztés gombot megnyomják.
- **should display all buttons on miscellaneous page**: Ellenőrzi, hogy az egyebek oldalon megjelenik-e az összes gomb.
- **should download songs in voting on miscellaneous page when download button is pressed**: Ellenőrzi, hogy az egyebek oldalon letölthetők-e a szavazásban lévő zenék, ha a letöltés gombot megnyomják.
- **should download winner song on miscellaneous page when download button is pressed**: Ellenőrzi, hogy az egyebek oldalon letölthető-e a nyertes zene, ha a letöltés gombot megnyomják.

`forbidden.spec.ts`

Ez a tesztfájl felelős az Tiltott oldal helyes megjelenítéséért és zavartalan működéséért. Minden funkciót alaposan teszteltünk, hogy biztosítsuk a problémamentes használatot a jövőbeli felhasználók számára. Az alábbiakban megtalálhatók a pontos tesztek és azok céljai:

- **should stay on forbidden page**: Ellenőrzi, hogy az oldal a tiltott oldalon marad-e.
- **should display the forbidden page with home button on large screens**: Ellenőrzi, hogy a tiltott oldal megjelenik-e a kezdőlap gombbal nagy képernyőkön.

- **should display the message on small screens**: Ellenőrzi, hogy a tiltott oldal üzenetet jelenít-e meg kis képernyőkön.
- **should navigate to home page when home button is clicked**: Ellenőrzi, hogy a kezdőlap gombra kattintva a felhasználó a kezdőlapra navigál-e.

`index.spec.ts`

Ez a tesztfájl felelős a Kezdőlap oldal helyes megjelenítéséért és zavartalan működéséért. Minden funkciót alaposan teszteltünk, hogy biztosítsuk a problémamentes használatot a jövőbeli felhasználók számára. Az alábbiakban megtalálhatók a pontos tesztek és azok céljai:

- **should display the home page**: Ellenőrzi, hogy a kezdőlap megjelenik-e.
- **should display the navigation bar**: Ellenőrzi, hogy a navigációs sáv megjelenik-e.
- **should navigate to about page when about link is clicked**: Ellenőrzi, hogy az "About" linkre kattintva a felhasználó az "About" oldalra navigál-e.
- **should display the footer**: Ellenőrzi, hogy a lábléc megjelenik-e.
- **should display the correct title**: Ellenőrzi, hogy a helyes cím megjelenik-e a kezdőlapon.
- **should display featured articles**: Ellenőrzi, hogy a kiemelt cikkek megjelennek-e a kezdőlapon.
- **should navigate to article page when article is clicked**: Ellenőrzi, hogy egy cikkre kattintva a felhasználó a cikk oldalára navigál-e.
- **should display the search bar**: Ellenőrzi, hogy a keresősáv megjelenik-e.
- **should show search results when a query is entered**: Ellenőrzi, hogy a keresési eredmények megjelennek-e, ha egy lekérdezést beírnak.
- **should display the login button**: Ellenőrzi, hogy a bejelentkezés gomb megjelenik-e.
- **should navigate to login page when login button is clicked**: Ellenőrzi, hogy a bejelentkezés gombra kattintva a felhasználó a bejelentkezési oldalra navigál-e.
- **should display the register button**: Ellenőrzi, hogy a regisztráció gomb megjelenik-e.
- **should navigate to register page when register button is clicked**: Ellenőrzi, hogy a regisztráció gombra kattintva a felhasználó a regisztrációs oldalra navigál-e.

`login.spec.ts`

Ez a tesztfájl felelős a Bejelentkezési oldal helyes megjelenítéséért és zavartalan működéséért. Minden funkciót alaposan teszteltünk, hogy biztosítsuk a problémamentes használatot a jövőbeli felhasználók számára. Az alábbiakban megtalálhatók a pontos tesztek és azok céljai:

- **should render login page correctly**: Ellenőrzi, hogy a bejelentkezési oldal helyesen jelenik-e meg.
- **should show validation errors if fields are empty**: Ellenőrzi, hogy a mezők üresen hagyása esetén megjelennek-e az érvényesítési hibák.
- **should handle login with correct credentials**: Ellenőrzi, hogy a bejelentkezés helyesen történik-e a megfelelő hitelesítő adatokkal.

- **should show error message on failed login**: Ellenőrzi, hogy hibás bejelentkezés esetén megjelenik-e a hibaüzenet.
- **should navigate to register page**: Ellenőrzi, hogy a regisztrációs oldalra navigál-e a felhasználó, ha a megfelelő linkre kattint.

`register.spec.ts`

Ez a tesztfájl felelős a Regisztrációs oldal helyes megjelenítéséért és zavartalan működéséért. Minden funkciót alaposan teszteltünk, hogy biztosítsuk a problémamentes használatot a jövőbeli felhasználók számára. Az alábbiakban megtalálhatók a pontos tesztek és azok céljai:

- **should display the registration form**: Ellenőrzi, hogy a regisztrációs űrlap megjelenik-e.
- **should have all input fields**: Ellenőrzi, hogy az összes bemeneti mező megjelenik-e.
- **should display labels for input fields**: Ellenőrzi, hogy a bemeneti mezők címkéi megjelennek-e.
- **should navigate to login page when link is clicked**: Ellenőrzi, hogy a bejelentkezési oldalra navigál-e a felhasználó, ha a megfelelő linkre kattint.
- **should register a new user**: Ellenőrzi, hogy egy új felhasználó regisztrálható-e.
- **should show error message on registration failure**: Ellenőrzi, hogy regisztrációs hiba esetén megjelenik-e a hibaüzenet.
- **should show error message on password mismatch**: Ellenőrzi, hogy jelszó nem egyezőség esetén megjelenik-e a hibaüzenet.

`tv.spec.ts`

Ez a tesztfájl felelős a Tv oldal helyes megjelenítéséért és zavartalan működéséért. Minden funkciót alaposan teszteltünk, hogy biztosítsuk a problémamentes használatot a jövőbeli felhasználók számára. Az alábbiakban megtalálhatók a pontos tesztek és azok céljai:

- **should navigate to tv page**: Ellenőrzi, hogy a TV oldal megjelenik-e.
- **should navigate to index page when logged in**: Ellenőrzi, hogy ha a felhasználó be van jelentkezve, akkor a kezdőoldalra navigál-e.
- **should display call to action when no session is active**: Ellenőrzi, hogy ha nincs aktív szavazási szekció, akkor megjelenik-e a cselekvési hívás.
- **should display correct data when a session is active**: Ellenőrzi, hogy ha aktív szavazási szekció van, akkor helyesen jelenik-e meg az adat.

`snipper.spec.ts`

Ez a tesztfájl felelős a Snipper oldal helyes megjelenítéséért és zavartalan működéséért. Minden funkciót alaposan teszteltünk, hogy biztosítsuk a problémamentes használatot a jövőbeli felhasználók számára. Az alábbiakban megtalálhatók a pontos tesztek és azok céljai:

- **should display snipper page**: Ellenőrzi, hogy a Snipper oldal megjelenik-e.
- **should display home button**: Ellenőrzi, hogy a kezdőlap gomb megjelenik-e.
- **should display all elements**: Ellenőrzi, hogy az összes elem megjelenik-e az oldalon.
- **should display warning message when no youtube link is given**: Ellenőrzi, hogy figyelmeztető üzenet jelenik-e meg, ha nincs megadva YouTube link.
- **should display warning message when no section is given**: Ellenőrzi, hogy figyelmeztető üzenet jelenik-e meg, ha nincs megadva szekció a zenéből.
- **should display warning message when no title is given**: Ellenőrzi, hogy figyelmeztető üzenet jelenik-e meg, ha nincs megadva cím.
- **should not allow section to be more than 15 seconds**: Ellenőrzi, hogy a szekció hossza nem haladhatja meg a 15 másodpercet.
- **should request upload when all data is given**: Ellenőrzi, hogy a feltöltési kérés elindul-e, ha minden adat meg van adva.
- **should display error message when request fails**: Ellenőrzi, hogy hibaüzenet jelenik-e meg, ha a feltöltési kérés sikertelen.

Backend felépítése

A Pollák Csengő backend alkalmazásának metodológiája a következő: a backend alkalmazás moduláris felépítésű, a különböző funkciókat külön modulokba szervezi. A modulok egymással kommunikálnak a NestJS Dependency Injection segítségével, és a különböző modulok közötti navigáció a NestJS Router segítségével történik. Minden lekérdezés egy Prisma ORM segítségével történik, hogy a modulok egyszerű és konzisztens adatbázis hozzáférést tarthassanak. A modulok a NestJS keretrendszer segítségével vannak összeállítva és futtatva.

- **.github**
  - *workflows*: Tartalmazza a GitHub Actions munkafolyamatokat.
    - *ci.yml*: A folyamatos integrációs munkafolyamat konfigurációja.
- **.idea**
  - *codeStyles*: Kódstílus beállítások.
    - *codeStyleConfig.xml*: Kódstílus konfiguráció.
    - *Project.xml*: Projekt szintű kódstílus beállítások.
  - *inspectionProfiles*: Ellenőrzési profilok.
    - *Project_Default.xml*: Alapértelmezett ellenőrzési profil.
  - *runConfigurations*: Futtatási konfigurációk.
    - *docker_compose_dev_yml_csengo_v2_postgres_dev__Compose_Deployment.xml*: Docker Compose futtatási konfiguráció.
    - *prisma_update.xml*: Prisma frissítési konfiguráció.
    - *start_dev.xml*: Fejlesztési futtatási konfiguráció.
    - *test_cov.xml*: Teszt lefedettségi konfiguráció.
    - *test_e2e.xml*: Végpontok közötti teszt konfiguráció.
  - *csengo-ts-server-v2.iml*: Projekt fájl.
  - *discord.xml*: Discord integrációs beállítások.
  - *git_toolbox_blame.xml*: Git blame eszköz beállításai.
  - *git_toolbox_prj.xml*: Git projekt eszköz beállításai.
  - *modules.xml*: Modulok konfigurációja.
  - *prettier.xml*: Prettier beállítások.
  - *vcs.xml*: Verziókezelő rendszer beállításai.
- **prisma**
  - *schema.prisma*: Prisma séma fájl.
  - **migrations**: Adatbázis migrációk.
    - *migration_lock.toml*: Migrációs zárolási fájl.
    - *20241210185133_prod*: Migrációs mappa.
      - *migration.sql*: SQL migrációs fájl.
    - *20250117190305_prod*: Migrációs mappa.
      - *migration.sql*: SQL migrációs fájl.
- **src**
  - *main.ts*: Az alkalmazás belépési pontja.
  - **config**: Konfigurációs szolgáltatások.
    - *config.local.module.ts*: Helyi konfigurációs modul.
    - *prisma.config.service.ts*: Prisma konfigurációs szolgáltatás.

- **filter**: Szűrők.
  - *delete.file.on.error.filter.ts*: Fájl törlése hiba esetén szűrő.
  - *filter.module.ts*: Szűrő modul.
- **module**: Modulok.
  - *app.module.ts*: Alkalmazás modul.
  - **app**: Alkalmazás modul komponensei.
    - *app.controller.spec.ts*: Alkalmazás vezérlő tesztje.
    - *app.controller.ts*: Alkalmazás vezérlő.
    - *app.service.spec.ts*: Alkalmazás szolgáltatás tesztje.
    - *app.service.ts*: Alkalmazás szolgáltatás.
  - **auth**: Hitelesítési modul.
    - *auth.controller.spec.ts*: Hitelesítési vezérlő tesztje.
    - *auth.controller.ts*: Hitelesítési vezérlő.
    - *auth.decorator.ts*: Hitelesítési dekorátor.
    - *auth.guard.spec.ts*: Hitelesítési őr tesztje.
    - *auth.guard.ts*: Hitelesítési őr.
    - *auth.module.ts*: Hitelesítési modul.
    - *auth.service.spec.ts*: Hitelesítési szolgáltatás tesztje.
    - *auth.service.ts*: Hitelesítési szolgáltatás.
    - **dto**: Adatátviteli objektumok.
      - *login.dto.ts*: Bejelentkezési DTO.
      - *register.dto.ts*: Regisztrációs DTO.
  - **pendingSongs**: Függő zenék modul.
    - *pending.songs.controller.spec.ts*: Függő zenék vezérlő tesztje.
    - *pending.songs.controller.ts*: Függő zenék vezérlő.
    - *pending.songs.module.ts*: Függő zenék modul.
    - *pending.songs.service.spec.ts*: Függő zenék szolgáltatás tesztje.
    - *pending.songs.service.ts*: Függő zenék szolgáltatás.
  - **role**: Szerepkör modul.
    - *role.decorator.ts*: Szerepkör dekorátor.
    - *role.enum.ts*: Szerepkör enum.
    - *role.guard.spec.ts*: Szerepkör őr tesztje.
    - *role.guard.ts*: Szerepkör őr.
  - **snipper**: Snipper modul.
    - *snipper.controller.spec.ts*: Snipper vezérlő tesztje.
    - *snipper.controller.ts*: Snipper vezérlő.
    - *snipper.module.ts*: Snipper modul.
    - *snipper.service.ts*: Snipper szolgáltatás.
    - *snipper.service.spec.ts*: Snipper szolgáltatás tesztje.
    - **dto**: Adatátviteli objektumok.
      - *add.song.dto.ts*: Snipper létrehozási DTO.
  - **songs**: Zenék modul.
    - *songs.controller.spec.ts*: Zenék vezérlő tesztje.
    - *songs.controller.ts*: Zenék vezérlő.
    - *songs.module.ts*: Zenék modul.
    - *songs.service.spec.ts*: Zenék szolgáltatás tesztje.
    - *songs.service.ts*: Zenék szolgáltatás.

- **dto**: Adatátviteli objektumok.
  - *create.song.dto.ts*: Zene létrehozási DTO.
  - *update.schedule.dto.ts*: Ütemezés frissítési DTO.
- **tv**: TV modul.
  - *tv.controller.spec.ts*: TV vezérlő tesztje.
  - *tv.controller.ts*: TV vezérlő.
  - *tv.module.ts*: TV modul.
  - *tv.service.spec.ts*: TV szolgáltatás tesztje.
  - *tv.service.ts*: TV szolgáltatás.
- **user**: Felhasználó modul.
  - *user.controller.spec.ts*: Felhasználó vezérlő tesztje.
  - *user.controller.ts*: Felhasználó vezérlő.
  - *user.module.ts*: Felhasználó modul.
  - *user.service.spec.ts*: Felhasználó szolgáltatás tesztje.
  - *user.service.ts*: Felhasználó szolgáltatás.
  - **dto**: Adatátviteli objektumok.
    - *update.pass.dto.ts*: Jelszó frissítési DTO.
    - *update.role.dto.ts*: Szerepkör frissítési DTO.
- **view**: Nézet modul.
  - *view.controller.spec.ts*: Nézet vezérlő tesztje.
  - *view.controller.ts*: Nézet vezérlő.
  - *view.module.ts*: Nézet modul.
  - *view.service.spec.ts*: Nézet szolgáltatás tesztje.
  - *view.service.ts*: Nézet szolgáltatás.
- **votes**: Szavazatok modul.
  - *votes.controller.spec.ts*: Szavazatok vezérlő tesztje.
  - *votes.controller.ts*: Szavazatok vezérlő.
  - *votes.module.ts*: Szavazatok modul.
  - *votes.service.spec.ts*: Szavazatok szolgáltatás tesztje.
  - *votes.service.ts*: Szavazatok szolgáltatás.
- **votingSessions**: Szavazási szekciók modul.
  - *voting.sessions.controller.spec.ts*: Szavazási szekciók vezérlő tesztje.
  - *voting.sessions.controller.ts*: Szavazási szekciók vezérlő.
  - *voting.sessions.module.ts*: Szavazási szekciók modul.
  - *voting.sessions.service.spec.ts*: Szavazási szekciók szolgáltatás.
  - *voting.sessions.service.ts*: Szavazási szekciók szolgáltatás.
  - **dto**: Adatátviteli objektumok.
    - *voting.session.dto.ts*: Szavazási szekció DTO.

- **pipe**: Pipe-ok.
  - *audio.length.validation.pipe.service.spec.ts*: Audio hossz validációs pipe tesztje.
  - *audio.length.validation.pipe.service.ts*: Audio hossz validációs pipe.
  - *pipe.module.ts*: Pipe modul.
- **type**: Típusdefiníciók.
  - *express.d.ts*: Express típusdefiníciók.
  - *jwt.d.ts*: JWT típusdefiníciók.
- **util**: Segédfüggvények.
  - *hbs.helpers.util.ts*: Tartalmazza a Handlebars segédfüggvényeit.

    - *time.parser.util.ts*: Időpont elemző segédfüggvény.
- **test**
    - *app.e2e-spec.ts*: Végpontok közötti teszt.
    - *jest-e2e.json*: Jest végpontok közötti teszt konfiguráció.
    - *test-docs.md*: Teszt dokumentáció.
- *.dockerignore*: Ez a fájl tartalmazza azokat a fájlokat és könyvtárakat, amelyeket a Docker figyelmen kívül hagy a build folyamat során.
- *.env.example*: Példa környezeti változókat tartalmazó fájl, amely segít a fejlesztőknek a saját `.env` fájljaik létrehozásában.
- *.eslintrc.js*: Az ESLint konfigurációs fájlja, amely meghatározza a kódminőségi szabályokat és beállításokat.
- *.gitignore*: Ez a fájl tartalmazza azokat a fájlokat és könyvtárakat, amelyeket a Git figyelmen kívül hagy a verziókezelés során.
- *.prettierrc*: A Prettier konfigurációs fájlja, amely meghatározza a kódformázási szabályokat és beállításokat.
- *docker-compose.dev.yml*: Docker Compose konfigurációs fájl, amely a fejlesztési környezethez szükséges szolgáltatásokat és beállításokat tartalmazza.
- *docker-compose.release.yml*: Docker Compose konfigurációs fájl, amely a kiadási környezethez szükséges szolgáltatásokat és beállításokat tartalmazza.
- *Dockerfile*: A Docker image létrehozásához szükséges utasításokat tartalmazó fájl.
- *nest-cli.json*: A NestJS CLI konfigurációs fájlja, amely a projekt beállításait tartalmazza.
- *package-lock.json*: A Node.js csomagok pontos verzióit rögzítő fájl, amely biztosítja a reprodukálható build folyamatot.
- *package.json*: A Node.js projekt konfigurációs fájlja, amely tartalmazza a projekt függőségeit és szkripteket.
- *README.md*: A projekt leírását és használati útmutatóját tartalmazó fájl.
- *sonar-project.properties*: A SonarQube konfigurációs fájlja, amely a kódminőség elemzéséhez szükséges beállításokat tartalmazza.
- *tsconfig.build.json*: A TypeScript build konfigurációs fájlja, amely meghatározza a build folyamat során használt beállításokat.
- *tsconfig.json*: A TypeScript projekt alapértelmezett konfigurációs fájlja, amely meghatározza a fordító beállításait.

Backend Tesztelés

Az alábbi szakaszban bemutatjuk a projekt ellenőrzésére használt backend teszteket. Részletesen ismertetjük az egyes fájlokat és a bennük található teszteseteket. A szerver minden modulát alapos tesztelésnek vetettük alá, amely biztosítja, hogy az összes funkció megfelelően és hibamentesen működik.

A backend tesztelési folyamatok az alábbi **két** csoportra vannak bontva:

- **Controller Tesztelés:** A controller tesztek biztosítják, hogy a service fájlokban található funkciók a kívántaknak megfelelően elérhetőek.
- **Service Tesztelés:** A service tesztek biztosítják a funkciók megfelelő működéseit.
- **Pipe Tesztelés:** A pipe tesztek garantálják, hogy a fájlok ellenőrzési folyamatai megfelelően működjenek.
- **Guard Tesztelés:** A guard tesztek biztosítják, hogy az útvonalakon elhelyezett védelmi mechanizmusok hibátlanul működjenek.

Az alábbi képen látható az összes Backend teszt lefutásának összegzése:

```
Test Suites: 23 passed, 23 total
Tests:       161 passed, 161 total
Snapshots:   0 total
Time:        27.148 s, estimated 50 s
Ran all test suites.
```

Backend Controller Tesztelés

Az alábbiakban bemutatjuk az összes Controller tesztet, amelyet a backend működésének ellenőrzésére alkalmaztunk. Minden fájl részletesen dokumentálva van, beleértve az összes benne található tesztesetet. A leírások tartalmazzák a mockolt bemeneti adatokat, valamint a várható visszatérési értékeket is.

`app.controller.spec`

- **`uploadDirect`**: A `uploadDirect` teszteset ellenőrzi, hogy az adminisztrátor közvetlenül feltölthet-e egy dalt.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
    - **file**:
      ```json
      {
        "mimetype": "audio/mpeg"
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      }
      ```
- **`renameById`**: A `renameById` teszteset ellenőrzi, hogy az adminisztrátor átnevezhet-e egy dalt azonosító alapján.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      ```

```
    }
```

- **`deleteById`**: A `deleteById` teszteset ellenőrzi, hogy az adminisztrátor törölhet-e egy dalt azonosító alapján.
  - **Mocked input data**:
    - **mockRequest**:
      ```
      json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```
      json
      {
        "success": true
      }
      ```

`auth.controller.spec`

- **AuthController -> should be defined**: Ez a teszteset ellenőrzi, hogy az `authController` megfelelően definiálva és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **None**:
      ```
      json
      {}
      ```
  - **Return value**:
    - **None**:
      ```
      json
      {}
      ```
- **AuthController -> login -> should return a valid JWT token**: Ez a teszteset ellenőrzi, hogy az `authController` megfelelően definiálva és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **mockRequest**:
      ```
      json
      {
        "body": {
          "username": "testUser",
          "password": "testPassword"
        }
      }
      ```
    - **mockUser**:
      ```
      json
      {
        "id": "1",
        "username": "testUser",
        "password": "hashedPassword"
      }
      ```
  - **Return value**:

- **result**:
  json
  ```
  {
    "access_token": "mockedJwtToken"
  }
  ```

- **AuthController -> register -> should create a new user**: Ez a teszteset ellenőrzi, hogy az `authController` `register` metódusa új felhasználót hoz létre, ha érvényes regisztrációs adatokkal látják el.
  - **Mocked input data**:
    - **mockRequest**:
      json
      ```
      {
        "body": {
          "username": "newUser",
          "password": "newPassword"
        }
      }
      ```
    - **mockUser**:
      json
      ```
      {
        "id": "2",
        "username": "newUser",
        "password": "hashedNewPassword"
      }
      ```
  - **Return value**:
    - **result**:
      json
      ```
      {
        "id": "2",
        "username": "newUser"
      }
      ```

- **AuthController -> validateUser -> should validate a user with correct credentials**: Ez a teszteset ellenőrzi, hogy az `authController` `validateUser` metódusa érvényesíti-e a felhasználót, ha helyes hitelesítő adatokkal látják el.
  - **Mocked input data**:
    - **mockRequest**:
      json
      ```
      {
        "body": {
          "username": "validUser",
          "password": "validPassword"
        }
      }
      ```
    - **mockUser**:
      json
      ```
      {
        "id": "3",
        "username": "validUser",
        "password": "hashedValidPassword"
      }
      ```
  - **Return value**:
    - **result**:

```json
{
  "id": "3",
  "username": "validUser"
}
```

- **AuthController -> validateUser -> should return null for invalid credentials**: Ez a teszteset ellenőrzi, hogy az `authController` `validateUser` metódusa null értéket ad vissza, ha érvénytelen hitelesítő adatokkal látják el.
  - **Mocked input data**:
    - **mockRequest**:

```json
{
  "body": {
    "username": "invalidUser",
    "password": "invalidPassword"
  }
}
```

  - **Return value**:
    - **result**:

```json
null
```

`pending.songs.controller.spec.ts`

- **PendingSongsController -> getAllPendingSongs -> should return all pending songs**: Ez a teszteset ellenőrzi, hogy a `PendingSongsController` `getAllPendingSongs` metódusa helyesen adja vissza az összes függőben lévő dalt. A kérést és a szolgáltatás metódusát mock-olja, hogy egy listát adjon vissza a függőben lévő dalokról, és ellenőrzi, hogy a vezérlő metódusa a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:

```json
{
  "token": { "sub": "mockedId", "username": "mockedUser", "roles": ["Admin"], "hashedPassword": "mocked-password" }
}
```

    - **mockPendingSongs**:

```json
[
  { "id": "song1", "title": "Song 1" },
  { "id": "song2", "title": "Song 2" }
]
```

  - **Return value**:
    - **result**:

```json
[
  { "id": "song1", "title": "Song 1" },
  { "id": "song2", "title": "Song 2" }
]
```

- **PendingSongsController -> approvePendingSong -> should approve a pending song**: Ez a teszteset ellenőrzi, hogy a `PendingSongsController` `approvePendingSong` metódusa helyesen hagy jóvá egy függőben lévő dalt. A

kérést és a szolgáltatás metódusát mock-olja, hogy jóváhagyjon egy dalt, és ellenőrzi, hogy a vezérlő metódusa a várt eredményt adja-e vissza.
    - **Mocked input data**:
        - **mockRequest**:
          ```json
          {
            "token": { "sub": "mockedId", "username": "mockedUser", "roles": ["Admin"], "hashedPassword": "mocked-password" }
          }
          ```
        - **mockSongId**:
          ```json
          "song1"
          ```
    - **Return value**:
        - **result**:
          ```json
          { "success": true }
          ```
- **PendingSongsController -> rejectPendingSong -> should reject a pending song**: Ez a teszteset ellenőrzi, hogy a `PendingSongsController` `rejectPendingSong` metódusa helyesen utasít-e el egy függőben lévő dalt. A kérést és a szolgáltatás metódusát mock-olja, hogy elutasítson egy dalt, és ellenőrzi, hogy a vezérlő metódusa a várt eredményt adja-e vissza.
    - **Mocked input data**:
        - **mockRequest**:
          ```json
          {
            "token": { "sub": "mockedId", "username": "mockedUser", "roles": ["Admin"], "hashedPassword": "mocked-password" }
          }
          ```
        - **mockSongId**:
          ```json
          "song1"
          ```
    - **Return value**:
        - **result**:
          ```json
          { "success": true }
          ```

`songs.controller.spec.ts`

- **SongsController -> getAllAudioInSession -> should return all audio in session for public access**: Ez a teszteset ellenőrzi, hogy a `getAllAudioInSession` metódus visszaadja-e az összes hanganyagot a nyilvános hozzáféréshez. A `songsService.getAllAudioInSession` metódust mockolja, hogy egy üres hanganyag listát adjon vissza, és ellenőrzi, hogy a `songsController.getAllAudioInSession` metódus a várt eredményt adja-e vissza.
    - **Mocked input data**:

- **mockRequest**:
  ```json
  {
    "token": {
      "sub": "mockedId",
      "username": "mockedUser",
      "roles": ["Admin"],
      "hashedPassword": "mocked-password"
    }
  }
  ```
- **Return value**:
  - **result**:
    ```json
    {
      "audios": []
    }
    ```

- **SongsController -> getAllInSession -> should return all songs in session**: Ez a teszteset ellenőrzi, hogy a `getAllInSession` metódus visszaadja-e az összes dalt a szekcióban. A `songsService.getAllInSession` metódust mockolja, hogy egy üres dal listát adjon vissza, és ellenőrzi, hogy a `songsController.getAllInSession` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "songs": []
      }
      ```
- **SongsController -> getAll -> should return all songs for admin**: Ez a teszteset ellenőrzi, hogy a `getAll` metódus visszaadja-e az összes dalt az admin számára. A `songsService.getAll` metódust mockolja, hogy egy üres dal listát adjon vissza, és ellenőrzi, hogy a `songsController.getAll` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:

- **mockRequest**:
  ```json
  {
    "token": {
      "sub": "mockedId",
      "username": "mockedUser",
      "roles": ["Admin"],
      "hashedPassword": "mocked-password"
    }
  }
  ```
- **Return value**:
  - **result**:
    ```json
    {
      "songs": []
    }
    ```
- **SongsController -> getWinner -> should return the winner song**: Ez a teszteset ellenőrzi, hogy a `getWinner` metódus visszaadja-e a nyertes dalt. A `songsService.getWinner` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `songsController.getWinner` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      }
      ```
- **SongsController -> getWinnerAudio -> should return winner audio**: Ez a teszteset ellenőrzi, hogy a `getWinnerAudio` metódus visszaadja-e a nyertes hanganyagot. A `songsService.getWinnerAudio` metódust mockolja, hogy egy `StreamableFile` objektumot adjon vissza, és ellenőrzi, hogy a `songsController.getWinnerAudio` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:

        - **mockRequest**:
          json
          ```
          {
            "token": {
              "sub": "mockedId",
              "username": "mockedUser",
              "roles": ["Admin"],
              "hashedPassword": "mocked-password"
            }
          }
          ```
    - **Return value**:
        - **result**:
          json
          ```
          {
            "type": "StreamableFile",
            "data": "audio"
          }
          ```
- **SongsController -> getAudioById -> should return audio by ID**: Ez a teszteset ellenőrzi, hogy a `getAudioById` metódus visszaadja-e a hanganyagot azonosító alapján. A `songsService.getAudioById` metódust mockolja, hogy egy `StreamableFile` objektumot adjon vissza, és ellenőrzi, hogy a `songsController.getAudioById` metódus a várt eredményt adja-e vissza.
    - **Mocked input data**:
        - **mockRequest**:
          json
          ```
          {
            "token": {
              "sub": "mockedId",
              "username": "mockedUser",
              "roles": ["Admin"],
              "hashedPassword": "mocked-password"
            }
          }
          ```
    - **Return value**:
        - **result**:
          json
          ```
          {
            "type": "StreamableFile",
            "data": "audio"
          }
          ```
- **SongsController -> updateAudioById -> should update audio for admin**: Ez a teszteset ellenőrzi, hogy a `updateAudioById` metódus frissíti-e a hanganyagot az admin számára. A `songsService.updateAudio` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `songsController.updateAudio` metódus a várt eredményt adja-e vissza.
    - **Mocked input data**:

- **mockRequest**:
  ```json
  {
    "token": {
      "sub": "mockedId",
      "username": "mockedUser",
      "roles": ["Admin"],
      "hashedPassword": "mocked-password"
    }
  }
  ```
- **Return value**:
  - **result**:
    ```json
    {
      "success": true
    }
    ```

- **SongsController -> upload -> should upload a song**: Ez a teszteset ellenőrzi, hogy a `upload` metódus feltölt-e egy dalt. A `songsService.upload` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `songsController.upload` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
    - **file**:
      ```json
      {
        "mimetype": "audio/mpeg"
      }
      ```
    - **createSongDto**:
      ```json
      {
        "title": "mocked-title"
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      }
      ```

- **SongsController -> uploadDirect -> should upload a song directly for admin**: Ez a teszteset ellenőrzi, hogy a `uploadDirect` metódus közvetlenül feltölt-e egy dalt az admin számára. A `songsService.uploadDirect` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `songsController.uploadDirect` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
    - **file**:
      ```json
      {
        "mimetype": "audio/mpeg"
      }
      ```
    - **createSongDto**:
      ```json
      {
        "title": "mocked-title"
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      }
      ```
- **SongsController -> renameById -> should rename a song for admin**: Ez a teszteset ellenőrzi, hogy a `renameById` metódus átnevez-e egy dalt az admin számára. A `songsService.renameById` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `songsController.renameById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:

- **result**:
  ```json
  {
    "success": true
  }
  ```

- **SongsController -> deleteById -> should delete a song for admin**: Ez a teszteset ellenőrzi, hogy a `deleteById` metódus töröl-e egy dalt az admin számára. A `songsService.deleteById` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `songsController.deleteById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      }
      ```

`tv.controller.spec.ts`

- **TvController -> getAll -> should return summary of votes in session**: Ez a teszteset ellenőrzi, hogy a `getSummaryOfVotesInSession` metódus visszaadja-e a szavazatok összegzését a szekcióban. A `tvService.getSummaryOfVotesInSession` metódust mockolja, hogy egy szavazatok összegzését adjon vissza, és ellenőrzi, hogy a `tvController.getSummaryOfVotesInSession` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:

- **result**:
  json
  ```
  {
    "summary": "Summary of votes"
  }
  ```

`user.controller.spec.ts`

- **UserController -> getRealNameById -> should return real name**: Ez a teszteset ellenőrzi, hogy a `getRealNameById` metódus visszaadja-e a felhasználó valódi nevét. A `userService.getRealNameById` metódust mockolja, hogy egy valódi nevet adjon vissza, és ellenőrzi, hogy a `userController.getRealNameById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      json
      ```
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      json
      ```
      {
        "realName": "mocked-name"
      }
      ```
- **UserController -> getAll -> should return all users**: Ez a teszteset ellenőrzi, hogy a `getAll` metódus visszaadja-e az összes felhasználót. A `userService.getAll` metódust mockolja, hogy egy üres felhasználó listát adjon vissza, és ellenőrzi, hogy a `userController.getAll` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      json
      ```
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      json
      ```
      {
        "users": []
      }
      ```

- **UserController -> updateUserPassword -> should return success when updating user password**: Ez a teszteset ellenőrzi, hogy a `updateUserPassword` metódus sikeresen frissíti-e a felhasználó jelszavát. A `userService.updateUserPassword` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `userController.updateUserPassword` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
    - **mockPassword**:
      ```json
      {
        "userId": "1",
        "password": "asd"
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      }
      ```
- **UserController -> updateUserRole -> should return success when updating user role**: Ez a teszteset ellenőrzi, hogy a `updateUserRole` metódus sikeresen frissíti-e a felhasználó szerepkörét. A `userService.updateUserRole` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `userController.updateUserRole` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
    - **mockRole**:
      ```json
      {
        "userId": "1",
        "role": "Admin"
      }
      ```

- **Return value**:
  - **result**:
    ```
    json
    {
      "success": true
    }
    ```

`view.controller.spec.ts`

- **ViewController -> renderSummaryOfVotesInSession -> should render summary of votes in session**: Ez a teszteset ellenőrzi, hogy a `renderSummaryOfVotesInSession` metódus visszaadja-e a szavazatok összegzését a szekcióban. A `viewService.getSummaryOfVotesInSessionData` metódust mockolja, hogy egy szavazatok összegzését adjon vissza, és ellenőrzi, hogy a `viewController.renderSummaryOfVotesInSession` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```
      json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```
      json
      "Rendered summary of votes"
      ```
- **ViewController -> renderPendingSongs -> should render pending songs**: Ez a teszteset ellenőrzi, hogy a `renderPendingSongs` metódus visszaadja-e a függőben lévő dalokat. A `viewService.getPendingSongsData` metódust mockolja, hogy egy függőben lévő dalokat adjon vissza, és ellenőrzi, hogy a `viewController.renderPendingSongs` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```
      json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```
      json
      "Rendered pending songs"
      ```

`votes.controller.spec.ts`

- **VotesController -> getSummrayOfVotesInSession -> should return summary of votes in session**: Ez a teszteset ellenőrzi, hogy a `getSummrayOfVotesInSession` metódus visszaadja-e a szavazatok összegzését a szekcióban. A `votesService.getSummaryOfVotesInSession` metódust mockolja, hogy egy szavazatok összegzését adjon vissza, és ellenőrzi, hogy a `votesController.getSummrayOfVotesInSession` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "votes": []
      }
      ```
- **VotesController -> getVotedSongsByUserId -> should return votes songs by user id**: Ez a teszteset ellenőrzi, hogy a `getVotedSongsByUserId` metódus visszaadja-e a felhasználó által szavazott dalokat. A `votesService.getVotedSongsByUserId` metódust mockolja, hogy egy dal listát adjon vissza, és ellenőrzi, hogy a `votesController.getVotedSongsByUserId` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "songs": ["Song 1", "Song 2"]
      }
      ```

- **VotesController -> voteUp -> should return success when voted up**: Ez a teszteset ellenőrzi, hogy a `voteUp` metódus sikeresen végrehajtja-e a szavazat felfelé történő leadását. A `votesService.voteUp` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `votesController.voteUp` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      }
      ```
- **VotesController -> voteDown -> should return success when voted down**: Ez a teszteset ellenőrzi, hogy a `voteDown` metódus sikeresen végrehajtja-e a szavazat lefelé történő leadását. A `votesService.voteDown` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `votesController.voteDown` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      }
      ```

`voting.sessions.controller.spec.ts`

- **VotingSessionController -> should return summary of votes in session**: Ez a teszteset ellenőrzi, hogy a `getAllSessions` metódus visszaadja-e a szavazatok összegzését a szekcióban. A `votingSessionService.getAllSessions` metódust mockolja, hogy egy üres szavazatok összegzését adjon vissza, és ellenőrzi, hogy a `votingSessionController.getAllSessions` metódus a várt eredményt adja-e vissza.

- **Mocked input data**:
  - **mockRequest**:
    ```json
    {
      "token": {
        "sub": "mockedId",
        "username": "mockedUser",
        "roles": ["Admin"],
        "hashedPassword": "mocked-password"
      }
    }
    ```
- **Return value**:
  - **result**:
    ```json
    {
      "sessions": []
    }
    ```

- **VotingSessionController -> should return votes songs by user id**: Ez a teszteset ellenőrzi, hogy a `createSession` metódus visszaadja-e a felhasználó által szavazott dalokat. A `votingSessionService.createSession` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `votingSessionController.createSession` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
    - **mockSession**:
      ```json
      {
        "songIds": ["1", "2"],
        "start": "2020-01-01",
        "end": "2021-01-01"
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      }
      ```
- **VotingSessionController -> should return success when voted up**: Ez a teszteset ellenőrzi, hogy a `updateSessionById` metódus sikeresen frissíti-e a szavazat felfelé történő leadását. A `votingSessionService.updateSessionById` metódust mockolja, hogy egy

sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `votingSessionController.updateSessionById` metódus a várt eredményt adja-e vissza.

- **Mocked input data**:
  - **mockRequest**:
    ```json
    {
      "token": {
        "sub": "mockedId",
        "username": "mockedUser",
        "roles": ["Admin"],
        "hashedPassword": "mocked-password"
      }
    }
    ```
  - **mockSession**:
    ```json
    {
      "songIds": ["1", "2"],
      "start": "2020-01-01",
      "end": "2021-01-01"
    }
    ```
- **Return value**:
  - **result**:
    ```json
    {
      "success": true
    }
    ```

- **VotingSessionController -> should return success when voted down**: Ez a teszteset ellenőrzi, hogy a `deleteSessionById` metódus sikeresen végrehajtja-e a szavazat lefelé történő leadását. A `votingSessionService.deleteSessionById` metódust mockolja, hogy egy sikeres üzenetet adjon vissza, és ellenőrzi, hogy a `votingSessionController.deleteSessionById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "success": true
      }
      ```

`snipper.controller.spec.ts`

- **SnipperController -> should be defined**: A teszt ellenőrzi, hogy a `SnipperController` definiálva van-e.
  - **Mocked input data**:
    - **module**:

```json
{
  "controllers": ["SnipperController"],
  "providers": [
    {
      "provide": "SnipperService",
      "useValue": {
        "addSong": "jest.fn().mockResolvedValue('')"
      }
    },
    "PrismaConfigService",
    "ConfigService",
    "JwtService"
  ]
}
```

  - **Return value**:
    - **snipperController**:

```json
{
  "toBeDefined": true
}
```

- **SnipperController -> addSong -> should return an empty response when song is successfully added**: A teszt ellenőrzi, hogy a `SnipperController` üres választ ad vissza, amikor egy dal sikeresen hozzáadásra kerül.
  - **Mocked input data**:
    - **mockRequest**:

```json
{
  "token": {
    "sub": "mockedId",
    "username": "mockedUser",
    "roles": ["Admin"],
    "hashedPassword": "mocked-password"
  }
}
```

    - **mockRequestDto**:

```json
{
  "ytUrl":
"https://www.youtube.com/watch?v=8sgycukafqQ&list=RD6vNsAH
xJXwE&index=3",
  "from": 2,
  "to": 10,
  "title": "alma"
}
```

  - **Return value**:
    - **result**: 200 OK

Backend Service Tesztelés

Az alábbiakban bemutatjuk az összes Service tesztet, amelyet a backend működésének ellenőrzésére alkalmaztunk. Minden fájl részletesen dokumentálva van, beleértve az összes benne található tesztesetet. A leírások tartalmazzák a mockolt bemeneti adatokat, valamint a várható visszatérési értékeket is.

`app.service.spec.ts`

- **AppService -> should be defined**: Ez a teszteset ellenőrzi, hogy az `AppService` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **N/A**:
      ```json
      {}
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "defined": true
      }
      ```
- **AppService -> getHello -> should return "Hello World!" when getHello is called**: Ez a teszteset ellenőrzi, hogy a `getHello` metódus visszaadja-e a "Hello World!" szöveget, amikor meghívják.
  - **Mocked input data**:
    - **N/A**:
      ```json
      {}
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "message": "Hello World!"
      }
      ```

`auth.service.spec.ts`

- **AuthService -> should be defined**: Ez a teszteset ellenőrzi, hogy az `AuthService` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **N/A**:
      ```json
      {}
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "defined": true
      }
      ```

- **AuthService -> login -> should return JWT token upon successfull login**: Ez a teszteset ellenőrzi, hogy a `login` metódus visszaadja-e a JWT tokent sikeres bejelentkezés esetén. A `prisma.user.findFirstOrThrow` metódust mockolja, hogy egy felhasználói adatot adjon vissza, és ellenőrzi, hogy a `authService.login` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockedLoginDto**:
      ```json
      {
        "username": "mocked-user",
        "password": "mocked-password"
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "access_token": "test-token"
      }
      ```
- **AuthService -> login -> should throw FORBIDDEN if invalid username or password**: Ez a teszteset ellenőrzi, hogy a `login` metódus `FORBIDDEN` hibát dob-e érvénytelen felhasználónév vagy jelszó esetén. A `prisma.user.findFirstOrThrow` metódust mockolja, hogy egy felhasználói adatot adjon vissza, és ellenőrzi, hogy a `authService.login` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockedLoginDto**:
      ```json
      {
        "username": "mocked-user",
        "password": "mocked-password"
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "status": 403,
        "message": "Invalid username or password"
      }
      ```
- **AuthService -> login -> should throw NOT_FOUND if user not found**: Ez a teszteset ellenőrzi, hogy a `login` metódus `NOT_FOUND` hibát dob-e, ha a felhasználó nem található. A `prisma.user.findFirstOrThrow` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `authService.login` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockedLoginDto**:
      ```json
      {
        "username": "mocked-user",
        "password": "mocked-password"
      }
      ```

- **Return value**:
  - **error**:
    json
    ```
    {
      "status": 404,
      "message": "Failed to find user or invalid
    authentication data"
    }
    ```

- **AuthService -> register -> should return JWT token upon successful register**: Ez a teszteset ellenőrzi, hogy a `register` metódus visszaadja-e a JWT tokent sikeres regisztráció esetén. A `prisma.user.findFirst`, `prisma.kreta.findFirstOrThrow`, és `prisma.user.create` metódusokat mockolja, hogy felhasználói adatokat adjon vissza, és ellenőrzi, hogy a `authService.register` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockedRegisterDto**:
      json
      ```
      {
        "username": "new-user",
        "password": "new-password",
        "email": "new-email@test.com",
        "om": "72548167135"
      }
      ```
  - **Return value**:
    - **result**:
      json
      ```
      {
        "access_token": "test-token"
      }
      ```
- **AuthService -> register -> should return INTERNAL_SERVER_ERROR if checking existing user fails**: Ez a teszteset ellenőrzi, hogy a `register` metódus `INTERNAL_SERVER_ERROR` hibát dob-e, ha a meglévő felhasználó ellenőrzése sikertelen. A `prisma.user.findFirst` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `authService.register` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockedRegisterDto**:
      json
      ```
      {
        "username": "new-user",
        "password": "new-password",
        "email": "new-email@test.com",
        "om": "72548167135"
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "status": 500,
        "message": "Something went wrong"
      }
      ```

- **AuthService -> register -> should return INTERNAL_SERVER_ERROR if someone has already registered under those information**: Ez a teszteset ellenőrzi, hogy a `register` metódus `INTERNAL_SERVER_ERROR` hibát dob-e, ha valaki már regisztrált ezekkel az adatokkal. A `prisma.user.findFirst` metódust mockolja, hogy felhasználói adatot adjon vissza, és ellenőrzi, hogy a `authService.register` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockedRegisterDto**:
      ```json
      {
        "username": "existing-user",
        "password": "password",
        "email": "existing-email@test.com",
        "om": "72548167135"
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "status": 500,
        "message": "Seems like someone has already registered under this information"
      }
      ```
- **AuthService -> register -> should return INTERNAL_SERVER_ERROR if prisma cannot find the OM in the database**: Ez a teszteset ellenőrzi, hogy a `register` metódus `INTERNAL_SERVER_ERROR` hibát dob-e, ha a prisma nem találja az OM azonosítót az adatbázisban. A `prisma.kreta.findFirstOrThrow` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `authService.register` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockedRegisterDto**:
      ```json
      {
        "username": "new-user",
        "password": "new-password",
        "email": "new-email@test.com",
        "om": "72548167135"
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "status": 500,
        "message": "Can't find this OM id in the database"
      }
      ```
- **AuthService -> register -> should return INTERNAL_SERVER_ERROR if prisma fails to create user**: Ez a teszteset ellenőrzi, hogy a `register` metódus `INTERNAL_SERVER_ERROR` hibát dob-e, ha a prisma nem tudja létrehozni a felhasználót. A `prisma.user.create` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `authService.register` metódus a várt hibát dobja-e.
  - **Mocked input data**:

- **mockedRegisterDto**:
  ```json
  {
    "username": "new-user",
    "password": "new-password",
    "email": "new-email@test.com",
    "om": "72578167135"
  }
  ```
  - **Return value**:
    - **error**:
      ```json
      {
        "status": 500,
        "message": "Failed to register user bad OM"
      }
      ```

`pending.songs.service.spec.ts`

- **PendingSongsService -> should be defined**: Ez a teszteset ellenőrzi, hogy a `PendingSongsService` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **N/A**:
      ```json
      {}
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "defined": true
      }
      ```
- **PendingSongsService -> getAll -> should return all pending songs**: Ez a teszteset ellenőrzi, hogy a `getAll` metódus visszaadja-e az összes függőben lévő dalt. A `prisma.pendingSong.findMany` metódust mockolja, hogy egy listát adjon vissza a függőben lévő dalokról, és ellenőrzi, hogy a `service.getAll` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:

- **result**:
  json
  ```
  [
    {
      "id": "1",
      "title": "Song 1",
      "uploadedBy": {
        "kreta": {
          "name": "John Doe"
        }
      }
    }
  ]
  ```

- **PendingSongsService -> getAll -> should throw an error when no pending songs are found**: Ez a teszteset ellenőrzi, hogy a `getAll` metódus hibát dob-e, ha nem találhatóak függőben lévő dalok. A `prisma.pendingSong.findMany` metódust mockolja, hogy egy üres listát adjon vissza, és ellenőrzi, hogy a `service.getAll` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "status": 404,
        "message": "No pending songs found"
      }
      ```
- **PendingSongsService -> getAll -> should throw an error when an internal error occurs**: Ez a teszteset ellenőrzi, hogy a `getAll` metódus hibát dob-e, ha belső hiba történik. A `prisma.pendingSong.findMany` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `service.getAll` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:

- **error**:
  json
  ```
  {
    "status": 500,
    "message": "Error fetching pending songs: Database
  error"
  }
  ```

- **PendingSongsService -> getAudioById -> should return a streamable file**: Ez a teszteset ellenőrzi, hogy a `getAudioById` metódus visszaad-e egy streamelhető fájlt. A `prisma.pendingSong.findUnique` metódust mockolja, hogy egy dalt adjon vissza, és ellenőrzi, hogy a `service.getAudioById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **result**:
      json
      ```
      {
        "type": "StreamableFile"
      }
      ```
- **PendingSongsService -> getAudioById -> should throw an error when song is not found**: Ez a teszteset ellenőrzi, hogy a `getAudioById` metódus hibát dob-e, ha a dal nem található. A `prisma.pendingSong.findUnique` metódust mockolja, hogy null értéket adjon vissza, és ellenőrzi, hogy a `service.getAudioById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "status": 404,
        "message": "Song not found"
      }
      ```

- **PendingSongsService -> getAudioById -> should throw an error when an internal error occurs**: Ez a teszteset ellenőrzi, hogy a `getAudioById` metódus hibát dob-e, ha belső hiba történik. A `prisma.pendingSong.findUnique` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `service.getAudioById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "status": 500,
        "message": "Error fetching song: Database error"
      }
      ```
- **PendingSongsService -> approveById -> should approve a pending song**: Ez a teszteset ellenőrzi, hogy az `approveById` metódus jóváhagyja-e a függőben lévő dalt. A `prisma.pendingSong.findUnique`, `prisma.song.create`, és `prisma.pendingSong.delete` metódusokat mockolja, hogy egy dalt adjon vissza, és ellenőrzi, hogy a `service.approveById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "message": "Peding song with id 1 sucessfully approved"
      }
      ```
- **PendingSongsService -> approveById -> should throw an error when pending song is not found**: Ez a teszteset ellenőrzi, hogy az `approveById` metódus hibát dob-e, ha a függőben lévő dal nem található. A `prisma.pendingSong.findUnique` metódust mockolja, hogy null értéket adjon vissza, és ellenőrzi, hogy a `service.approveById` metódus a várt hibát dobja-e.
  - **Mocked input data**:

- **mockRequestUser**:
  ```json
  {
    "token": {
      "username": "testUser",
      "sub": "123"
    }
  }
  ```
- **Return value**:
  - **error**:
    ```json
    {
      "status": 404,
      "message": "Pending song not found"
    }
    ```

- **PendingSongsService -> approveById -> should throw an error when an internal error occurs**: Ez a teszteset ellenőrzi, hogy az `approveById` metódus hibát dob-e, ha belső hiba történik. A `prisma.pendingSong.findUnique` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `service.approveById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "status": 500,
        "message": "Error fetching pending song: Database error"
      }
      ```
- **PendingSongsService -> disapproveById -> should disapprove a pending song**: Ez a teszteset ellenőrzi, hogy a `disapproveById` metódus elutasítja-e a függőben lévő dalt. A `prisma.pendingSong.findFirstOrThrow` és `prisma.pendingSong.delete` metódusokat mockolja, hogy egy dalt adjon vissza, és ellenőrzi, hogy a `service.disapproveById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```

- **Return value**:
  - **result**:
    json
    ```
    {
      "message": "Peding song with id 1 sucessfully
    disapproved"
    }
    ```

- **PendingSongsService -> disapproveById -> should throw an error when pending song is not found**: Ez a teszteset ellenőrzi, hogy a `disapproveById` metódus hibát dob-e, ha a függőben lévő dal nem található. A `prisma.pendingSong.findFirstOrThrow` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `service.disapproveById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "status": 404,
        "message": "Error fetching pending song: Record not
      found"
      }
      ```
- **PendingSongsService -> disapproveById -> should throw an error when an internal error occurs**: Ez a teszteset ellenőrzi, hogy a `disapproveById` metódus hibát dob-e, ha belső hiba történik. A `prisma.pendingSong.findFirstOrThrow` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `service.disapproveById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "status": 500,
        "message": "Error fetching pending song: Database error"
      }
      ```

`songs.service.spec.ts`

- **SongsService -> should be defined**: Ez a teszteset ellenőrzi, hogy a `SongsService` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **service**:
      ```json
      {
        "defined": true
      }
      ```
- **SongsService -> getAllAudioInSession -> should return a zip file of songs in session**: Ez a teszteset ellenőrzi, hogy a `getAllAudioInSession` metódus visszaadja-e a szekcióban lévő dalok zip fájlját.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockResponse**:
      ```json
      {
        "attachment": "songs.zip",
        "pipe": true
      }
      ```
    - **mockVotingSession**:
      ```json
      {
        "id": "sessionId",
        "songs": [
          {
            "id": "songId",
            "title": "Test Song",
            "songBucket": {
              "path": "path/to/song.mp3"
            }
          }
        ]
      }
      ```
  - **Return value**:

- **result**:
  json
  ```
  {
    "truthy": true
  }
  ```

- **SongsService -> getAllAudioInSession -> should throw an error when no active voting session is found**: Ez a teszteset ellenőrzi, hogy a `getAllAudioInSession` metódus hibát dob-e, ha nincs aktív szavazási szekció.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockResponse**:
      json
      ```
      {
        "attachment": "songs.zip",
        "pipe": true
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "No active voting session found",
        "status": 404
      }
      ```
- **SongsService -> getAllInSession -> should return a list of songs in session**: Ez a teszteset ellenőrzi, hogy a `getAllInSession` metódus visszaadja-e a szekcióban lévő dalok listáját.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockVotingSession**:
      json
      ```
      {
        "id": "sessionId",
        "songs": [
          {
            "id": "songId",
            "title": "Test Song",
            "songBucket": {
              "path": "path/to/song.mp3"
      ```

```
        }
      }
    ]
  }
```

- **Return value**:
  - **result**:

```json
{
  "sessionId": "sessionId",
  "songs": [
    {
      "songId": "songId",
      "songTitle": "Test Song"
    }
  ]
}
```

- **SongsService -> getAllInSession -> should throw an error when no active voting session is found**: Ez a teszteset ellenőrzi, hogy a `getAllInSession` metódus hibát dob-e, ha nincs aktív szavazási szekció.
  - **Mocked input data**:
    - **mockRequestUser**:

```json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

  - **Return value**:
    - **error**:

```json
{
  "message": "No active voting session found",
  "status": 404
}
```

- **SongsService -> getAll -> should return a list of all songs**: Ez a teszteset ellenőrzi, hogy a `getAll` metódus visszaadja-e az összes dalt.
  - **Mocked input data**:
    - **mockRequestUser**:

```json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

- **mockSong**:
  json
  ```
  {
    "id": "songId",
    "title": "Test Song",
    "songBucket": {
      "path": "path/to/song.mp3"
    }
  }
  ```
- **Return value**:
  - **result**:
    json
    ```
    [
      {
        "id": "songId",
        "title": "Test Song"
      }
    ]
    ```
- **SongsService -> getAll -> should throw an error when no songs are found**: Ez a teszteset ellenőrzi, hogy a `getAll` metódus hibát dob-e, ha nem találhatóak dalok.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "No songs found",
        "status": 404
      }
      ```
- **SongsService -> getWinner -> should return the winner song**: Ez a teszteset ellenőrzi, hogy a `getWinner` metódus visszaadja-e a nyertes dalt.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```

- **mockVotingSession**:
  json
  ```
  {
    "id": "sessionId",
    "songs": [
      {
        "id": "songId",
        "title": "Test Song",
        "songBucket": {
          "path": "path/to/song.mp3"
        }
      }
    ]
  }
  ```
- **mockSong**:
  json
  ```
  {
    "id": "songId",
    "title": "Test Song",
    "songBucket": {
      "path": "path/to/song.mp3"
    }
  }
  ```

- **Return value**:
  - **result**:
    json
    ```
    {
      "id": "songId",
      "title": "Test Song",
      "songBucket": {
        "path": "path/to/song.mp3"
      }
    }
    ```

- **SongsService -> getWinner -> should throw an error when no finished voting session is found**: Ez a teszteset ellenőrzi, hogy a `getWinner` metódus hibát dob-e, ha nincs befejezett szavazási szekció.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "No finished voting session found",
        "status": 500
      }
      ```

- **SongsService -> getWinnerAudio -> should return winner audio**: Ez a teszteset ellenőrzi, hogy a `getWinnerAudio` metódus visszaadja-e a nyertes hanganyagot.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockVotingSession**:
      ```json
      {
        "id": "sessionId",
        "songs": [
          {
            "id": "songId",
            "title": "Test Song",
            "songBucket": {
              "path": "path/to/song.mp3"
            }
          }
        ]
      }
      ```
    - **mockSong**:
      ```json
      {
        "id": "songId",
        "title": "Test Song",
        "songBucket": {
          "path": "path/to/song.mp3"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "type": "StreamableFile"
      }
      ```
- **SongsService -> getWinnerAudio -> should throw an error when no finished voting session is found**: Ez a teszteset ellenőrzi, hogy a `getWinnerAudio` metódus hibát dob-e, ha nincs befejezett szavazási szekció.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```

- **Return value**:
  - **error**:
    json
    ```
    {
      "message": "No finished voting session found",
      "status": 500
    }
    ```
- **SongsService -> getAudioById -> should return a StreamableFile for the given song id**: Ez a teszteset ellenőrzi, hogy a `getAudioById` metódus visszaad-e egy streamelhető fájlt a megadott dal azonosító alapján.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockSong**:
      json
      ```
      {
        "id": "songId",
        "title": "Test Song",
        "songBucket": {
          "path": "path/to/song.mp3"
        }
      }
      ```
  - **Return value**:
    - **result**:
      json
      ```
      {
        "type": "StreamableFile"
      }
      ```
- **SongsService -> getAudioById -> should throw an error when song is not found**: Ez a teszteset ellenőrzi, hogy a `getAudioById` metódus hibát dob-e, ha a dal nem található.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "Song not found",
        "status": 404
      }
      ```

```
    }
```

- **SongsService -> updateAudio -> should update audio**: Ez a teszteset ellenőrzi, hogy a `updateAudio` metódus frissíti-e a hanganyagot.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockResult**:
      ```json
      {
        "message": "Success"
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "message": "Success"
      }
      ```
- **SongsService -> updateAudio -> should throw an error when update fails**: Ez a teszteset ellenőrzi, hogy a `updateAudio` metódus hibát dob-e, ha a frissítés sikertelen.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "Error updating audio: Update failed",
        "status": 500
      }
      ```
- **SongsService -> startAudio -> should start audio**: Ez a teszteset ellenőrzi, hogy a `startAudio` metódus elindítja-e a hanganyagot.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
      ```

```
    }
  }
```

- **mockResult**:

```json
{
  "message": "Success"
}
```

  - **Return value**:
    - **result**:

```json
{
  "message": "Success"
}
```

- **SongsService -> startAudio -> should throw an error when start fails**: Ez a teszteset ellenőrzi, hogy a `startAudio` metódus hibát dob-e, ha az indítás sikertelen.
  - **Mocked input data**:
    - **mockRequestUser**:

```json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

  - **Return value**:
    - **error**:

```json
{
  "message": "Error starting audio: Start failed",
  "status": 500
}
```

- **SongsService -> stopAudio -> should stop audio**: Ez a teszteset ellenőrzi, hogy a `stopAudio` metódus leállítja-e a hanganyagot.
  - **Mocked input data**:
    - **mockRequestUser**:

```json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

    - **mockResult**:

```json
{
  "message": "Success"
}
```

  - **Return value**:

- **result**:
  json
  ```
  {
    "message": "Success"
  }
  ```

- **SongsService -> stopAudio -> should throw an error when stop fails**: Ez a teszteset ellenőrzi, hogy a `stopAudio` metódus hibát dob-e, ha a leállítás sikertelen.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "Error stopping audio: Stop failed",
        "status": 500
      }
      ```
- **SongsService -> upload -> should upload a song**: Ez a teszteset ellenőrzi, hogy a `upload` metódus feltölt-e egy dalt.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockFile**:
      json
      ```
      {
        "path": "path/to/song.mp3"
      }
      ```
    - **mockCreateSongDto**:
      json
      ```
      {
        "title": "Test Song"
      }
      ```
    - **mockResult**:
      json
      ```
      {
        "id": "songId",
        "title": "Test Song"
      }
      ```
  - **Return value**:

- **result**:
  json
  ```
  {
    "id": "songId",
    "title": "Test Song"
  }
  ```

- **SongsService -> upload -> should throw an error when upload fails**: Ez a teszteset ellenőrzi, hogy a `upload` metódus hibát dob-e, ha a feltöltés sikertelen.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockFile**:
      json
      ```
      {
        "path": "path/to/song.mp3"
      }
      ```
    - **mockCreateSongDto**:
      json
      ```
      {
        "title": "Test Song"
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "Error uploading song into pending songs: Upload failed",
        "status": 500
      }
      ```
- **SongsService -> uploadDirect -> should upload a song directly**: Ez a teszteset ellenőrzi, hogy a `uploadDirect` metódus közvetlenül feltölt-e egy dalt.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockFile**:
      json
      ```
      {
        "path": "path/to/song.mp3"
      }
      ```

- **mockCreateSongDto**:
  ```json
  {
    "title": "Test Song"
  }
  ```
- **mockResult**:
  ```json
  {
    "id": "songId",
    "title": "Test Song"
  }
  ```

- **Return value**:
  - **result**:
    ```json
    {
      "id": "songId",
      "title": "Test Song"
    }
    ```

- **SongsService -> uploadDirect -> should throw an error when direct upload fails**: Ez a teszteset ellenőrzi, hogy a `uploadDirect` metódus hibát dob-e, ha a közvetlen feltöltés sikertelen. A `prisma.song.create` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `service.uploadDirect` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockFile**:
      ```json
      {
        "path": "path/to/song.mp3"
      }
      ```
    - **mockCreateSongDto**:
      ```json
      {
        "title": "Test Song"
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "Error uploading song directly into songs: Upload failed",
        "statusCode": 500
      }
      ```

- **SongsService -> renameById -> should rename a song**: Ez a teszteset ellenőrzi, hogy a `renameById` metódus átnevez-e egy dalt az admin számára. A `prisma.song.findUnique` és `prisma.song.update` metódusokat mockolja, hogy egy dalt adjon vissza, és ellenőrzi, hogy a `service.renameById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockSong**:
      json
      ```
      {
        "id": "songId",
        "title": "Test Song",
        "songBucket": {
          "path": "path/to/song.mp3"
        }
      }
      ```
  - **Return value**:
    - **result**:
      json
      ```
      {
        "message": "Song renamed successfully to New Title"
      }
      ```
- **SongsService -> renameById -> should throw an error when rename fails**: Ez a teszteset ellenőrzi, hogy a `renameById` metódus hibát dob-e, ha az átnevezés sikertelen. A `prisma.song.findUnique` és `prisma.song.update` metódusokat mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `service.renameById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockSong**:
      json
      ```
      {
        "id": "songId",
        "title": "Test Song",
        "songBucket": {
          "path": "path/to/song.mp3"
        }
      ```

```
    }
```

- **Return value**:
  - **error**:

```json
{
  "message": "Error renaming song: Rename failed",
  "statusCode": 500
}
```

- **SongsService -> deleteById -> should delete a song**: Ez a teszteset ellenőrzi, hogy a `deleteById` metódus töröl-e egy dalt az admin számára. A `prisma.song.findUnique` és `prisma.song.delete` metódusokat mockolja, hogy egy dalt adjon vissza, és ellenőrzi, hogy a `service.deleteById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:

```json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

    - **mockSong**:

```json
{
  "id": "songId",
  "title": "Test Song",
  "songBucket": {
    "path": "path/to/song.mp3"
  }
}
```

  - **Return value**:
    - **result**:

```json
{
  "message": "Song deleted successfully"
}
```

- **SongsService -> deleteById -> should throw an error when delete fails**: Ez a teszteset ellenőrzi, hogy a `deleteById` metódus hibát dob-e, ha a törlés sikertelen. A `prisma.song.findUnique` és `prisma.song.delete` metódusokat mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `service.deleteById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:

```json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

- **mockSong**:
  json
  ```
  {
    "id": "songId",
    "title": "Test Song",
    "songBucket": {
      "path": "path/to/song.mp3"
    }
  }
  ```
- **Return value**:
  - **error**:
    json
    ```
    {
      "message": "Error deleting song: Delete failed",
      "statusCode": 500
    }
    ```

`tv.service.spec.ts`

- **TvService -> should be defined**: Ez a teszteset ellenőrzi, hogy a `TvService` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **mockRequest**:
      json
      ```
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **service**:
      json
      ```
      {
        "defined": true
      }
      ```
- **TvService -> getSummaryOfVotesInSession -> should return summary of votes when a session is active**: Ez a teszteset ellenőrzi, hogy a `getSummaryOfVotesInSession` metódus visszaadja-e a szavazatok összegzését, amikor egy szekció aktív.
  - **Mocked input data**:
    - **mockRequest**:
      json
      ```
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```

- **prisma.votingSession.findFirst**:
  ```json
  {
    "id": 1,
    "start": "2023-01-01T00:00:00.000Z",
    "end": "2023-01-01T00:00:00.000Z",
    "songs": [
      {"id": 1, "title": "Song A"},
      {"id": 2, "title": "Song B"}
    ],
    "Vote": [
      {"songId": 1},
      {"songId": 1},
      {"songId": 2}
    ]
  }
  ```

- **Return value**:
  - **result**:
    ```json
    {
      "sessionId": 1,
      "songs": [
        {"songId": 1, "songTitle": "Song A", "voteCount": 2},
        {"songId": 2, "songTitle": "Song B", "voteCount": 1}
      ]
    }
    ```

- **TvService -> getSummaryOfVotesInSession -> should throw NOT_FOUND if no active session exists**: Ez a teszteset ellenőrzi, hogy a `getSummaryOfVotesInSession` metódus `NOT_FOUND` hibát dob-e, ha nincs aktív szavazási szekció.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
    - **prisma.votingSession.findFirst**:
      ```json
      null
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "No active voting session found",
        "status": 404
      }
      ```

- **TvService -> getSummaryOfVotesInSession -> should throw INTERNAL_SERVER_ERROR if database query fails**: Ez a teszteset ellenőrzi, hogy a `getSummaryOfVotesInSession` metódus `INTERNAL_SERVER_ERROR` hibát dob-e, ha az adatbázis lekérdezés sikertelen.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
    - **prisma.votingSession.findFirst**:
      ```json
      {
        "error": "Database error"
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "Error fetching currently running voting session: Database error",
        "status": 500
      }
      ```
- **TvService -> getSummaryOfVotesInSession -> should throw INTERNAL_SERVER_ERROR if vote counting fails**: Ez a teszteset ellenőrzi, hogy a `getSummaryOfVotesInSession` metódus `INTERNAL_SERVER_ERROR` hibát dob-e, ha a szavazatok számlálása sikertelen.
  - **Mocked input data**:
    - **mockRequest**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```

- **prisma.votingSession.findFirst**:
  json
  ```
  {
    "id": 1,
    "start": "2023-01-01T00:00:00.000Z",
    "end": "2023-01-01T00:00:00.000Z",
    "songs": [
      {"id": 1, "title": "Song A"}
    ],
    "Vote": null
  }
  ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "Error counting votes:",
        "status": 500
      }
      ```

`user.service.spec.ts`

- **UserService -> should be defined**: Ellenőrzi, hogy a `UserService` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **module**:
      json
      ```
      {
        "providers": [
          "UserService",
          {
            "provide": "PrismaConfigService",
            "useValue": {
              "user": {
                "findUnique": "jest.fn()",
                "findMany": "jest.fn",
                "findFirstOrThrow": "jest.fn()",
                "update": "jest.fn()"
              }
            }
          }
        ]
      }
      ```
  - **Return value**:
    - **service**:
      json
      ```
      {
        "defined": true
      }
      ```
- **UserService -> getRealNameById -> should return real name when user is found**: Ellenőrzi, hogy a `getRealNameById` metódus visszaadja-e a felhasználó valódi nevét, ha a felhasználó megtalálható.
  - **Mocked input data**:
    - **mockRequestUser**:
      json

```
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

- **mockUser**:
  json
```
{
  "kreta": {
    "name": "John Doe"
  }
}
```

- **Return value**:
  - **result**:
    json
```
{
  "realName": "John Doe"
}
```

- **UserService -> getRealNameById -> should throw an error when userId is missing**: Ellenőrzi, hogy a `getRealNameById` metódus hibát dob-e, ha a felhasználói azonosító hiányzik.
  - **Mocked input data**:
    - **requestUserWithoutId**:
      json
```
{
  "token": {
    "username": "testUser"
  }
}
```
  - **Return value**:
    - **error**:
      json
```
{
  "message": "Missing userId",
  "status": 400
}
```

- **UserService -> getRealNameById -> should throw an error when no user is found**: Ellenőrzi, hogy a `getRealNameById` metódus hibát dob-e, ha a felhasználó nem található.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
```
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```
  - **Return value**:

- **error**:
  ```json
  {
    "message": "No real name found by this user's OM",
    "status": 204
  }
  ```

- **UserService -> getRealNameById -> should throw an error when an internal error occurs**: Ellenőrzi, hogy a `getRealNameById` metódus hibát dob-e, ha belső hiba történik.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:

- **error**:
  ```json
  {
    "message": "Error fetching real name of user: Database error",
    "status": 500
  }
  ```

- **UserService -> getAll -> should return all users**: Ellenőrzi, hogy a `getAll` metódus visszaadja-e az összes felhasználót.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockUsers**:
      ```json
      [
        {
          "id": "1",
          "username": "user1",
          "email": "user1@example.com",
          "roles": [
            {
              "role": "admin"
            }
          ],
          "kreta": {
            "name": "User One",
            "om": "12345"
          }
        },
        {
          "id": "2",
          "username": "user2",
          "email": "user2@example.com",
          "roles": [
            {
              "role": "user"
            }
          ],
          "kreta": {
            "name": "User Two",
            "om": "67890"
          }
        }
      ]
      ```
  - **Return value**:

- **result**:
  json
  ```
  [
    {
      "id": "1",
      "username": "user1",
      "email": "user1@example.com",
      "role": "admin",
      "kreta": {
        "name": "User One",
        "om": "12345"
      }
    },
    {
      "id": "2",
      "username": "user2",
      "email": "user2@example.com",
      "role": "user",
      "kreta": {
        "name": "User Two",
        "om": "67890"
      }
    }
  ]
  ```

- **UserService -> getAll -> should throw an error when no users are found**:
  Ellenőrzi, hogy a `getAll` metódus hibát dob-e, ha nem találhatóak felhasználók.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "No users found",
        "status": 204
      }
      ```
- **UserService -> getAll -> should throw an error when an internal error occurs**:
  Ellenőrzi, hogy a `getAll` metódus hibát dob-e, ha belső hiba történik.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      ```

```
    }
```

- **Return value**:
  - **error**:
    ```json
    {
      "message": "Error fetching all users: Database error",
      "status": 500
    }
    ```

- **UserService -> updateUserPassword -> should update user password**: Ellenőrzi, hogy a `updateUserPassword` metódus frissíti-e a felhasználó jelszavát.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockUpdatePassDto**:
      ```json
      {
        "userId": "123",
        "password": "newPassword123"
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "message": "Password updated for user: testUser"
      }
      ```

- **UserService -> updateUserPassword -> should throw an error when user is not found**: Ellenőrzi, hogy a `updateUserPassword` metódus hibát dob-e, ha a felhasználó nem található.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockUpdatePassDto**:
      ```json
      {
        "userId": "123",
        "password": "newPassword123"
      }
      ```
  - **Return value**:

- **error**:
  json
  ```
  {
    "message": "User not found: User not found",
    "status": 500
  }
  ```

- **UserService -> updateUserPassword -> should throw an error when an internal error while finding user**: Ellenőrzi, hogy a `updateUserPassword` metódus hibát dob-e, ha belső hiba történik a felhasználó keresése közben.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockUpdatePassDto**:
      json
      ```
      {
        "userId": "123",
        "password": "newPassword123"
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "User not found: Database error",
        "status": 500
      }
      ```
- **UserService -> updateUserRole -> should update user role**: Ellenőrzi, hogy a `updateUserRole` metódus frissíti-e a felhasználó szerepkörét.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockUpdateRoleDto**:
      json
      ```
      {
        "userId": "123",
        "role": "Admin"
      }
      ```
  - **Return value**:

- **result**:
  json
  ```
  {
    "message": "Role updated for user: testUser"
  }
  ```

- **UserService -> updateUserRole -> should throw an error when user is not found**: Ellenőrzi, hogy a `updateUserRole` metódus hibát dob-e, ha a felhasználó nem található.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockUpdateRoleDto**:
      json
      ```
      {
        "userId": "123",
        "role": "Admin"
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "User not found: User not found",
        "status": 500
      }
      ```
- **UserService -> updateUserRole -> should throw an error when an internal error while finding user**: Ellenőrzi, hogy a `updateUserRole` metódus hibát dob-e, ha belső hiba történik a felhasználó keresése közben.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockUpdateRoleDto**:
      json
      ```
      {
        "userId": "123",
        "role": "Admin"
      }
      ```
  - **Return value**:

- **error**:
  json
  ```
  {
    "message": "User not found: Database error",
    "status": 500
  }
  ```

`view.service.spec.ts`

- **ViewService -> should be defined**: Ellenőrzi, hogy a `ViewService` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **mockPrismaService**:
      json
      ```
      {
        "votingSession": {
          "findFirst": "jest.fn()"
        },
        "pendingSong": {
          "findMany": "jest.fn()"
        }
      }
      ```
  - **Return value**:
    - **service**:
      json
      ```
      {
        "defined": true
      }
      ```
- **ViewService -> getSummaryOfVotesInSessionData -> should return summary of votes in session**: Ellenőrzi, hogy a `getSummaryOfVotesInSessionData` metódus visszaadja-e a szavazatok összegzését a szekcióban.
  - **Mocked input data**:
    - **mockRequest**:
      json
      ```
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
    - **mockVotingSession**:
      json
      ```
      {
        "id": "mockedSessionId",
        "songs": [
          { "id": "song1", "title": "Song 1" },
          { "id": "song2", "title": "Song 2" }
        ],
        "Vote": [
          { "songId": "song1" },
          { "songId": "song1" },
          { "songId": "song2" }
      ```

```
    ]
  }
```

- **Return value**:
  - **result**:
    json
    ```
    {
      "sessionId": "mockedSessionId",
      "songs": [
        { "songId": "song1", "songTitle": "Song 1",
    "voteCount": 2 },
        { "songId": "song2", "songTitle": "Song 2",
    "voteCount": 1 }
      ]
    }
    ```

- **ViewService -> getSummaryOfVotesInSessionData -> should throw error if no active voting session is found**: Ellenőrzi, hogy a `getSummaryOfVotesInSessionData` metódus hibát dob-e, ha nincs aktív szavazási szekció.
  - **Mocked input data**:
    - **mockRequest**:
      json
      ```
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "No active voting session found"
      }
      ```
- **ViewService -> getPendingSongsData -> should return pending songs data**: Ellenőrzi, hogy a `getPendingSongsData` metódus visszaadja-e a függőben lévő dalokat.
  - **Mocked input data**:
    - **mockRequest**:
      json
      ```
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```

- **mockPendingSongs**:
  json
  ```
  [
    { "id": "song1", "title": "Song 1" },
    { "id": "song2", "title": "Song 2" }
  ]
  ```
  - **Return value**:
    - **result**:
      json
      ```
      {
        "pendingSongs": [
          { "id": "song1", "title": "Song 1" },
          { "id": "song2", "title": "Song 2" }
        ]
      }
      ```
- **ViewService -> getPendingSongsData -> should throw error if no pending songs are found**: Ellenőrzi, hogy a `getPendingSongsData` metódus hibát dob-e, ha nem találhatóak függőben lévő dalok.
  - **Mocked input data**:
    - **mockRequest**:
      json
      ```
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "No pending songs found"
      }
      ```

`votes.service.spec.ts`

- **VotesService -> should be defined**: Ellenőrzi, hogy a `VotesService` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:

- **result**:
  ```json
  {}
  ```

- **VotesService -> getSummaryOfVotesInSession -> should return summary of votes in session**: Ellenőrzi, hogy a `getSummaryOfVotesInSession` metódus visszaadja-e a szavazatok összegzését a szekcióban. A `prisma.votingSession.findFirst` metódust mockolja, hogy egy szavazási szekciót adjon vissza, és ellenőrzi, hogy a `service.getSummaryOfVotesInSession` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockVotingSession**:
      ```json
      {
        "id": "1",
        "start": "2022-11-30T00:00:00.000Z",
        "end": "2024-11-30T00:00:00.000Z",
        "songs": [],
        "Vote": []
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "sessionId": "1",
        "songs": []
      }
      ```

- **VotesService -> getSummaryOfVotesInSession -> should throw an error when no active voting session is found**: Ellenőrzi, hogy a `getSummaryOfVotesInSession` metódus hibát dob-e, ha nincs aktív szavazási szekció. A `prisma.votingSession.findFirst` metódust mockolja, hogy null értéket adjon vissza, és ellenőrzi, hogy a `service.getSummaryOfVotesInSession` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:

- **error**:
  json
  ```
  {
    "message": "No active voting session found",
    "status": 404
  }
  ```

- **VotesService -> getSummaryOfVotesInSession -> should throw an error when an internal error occurs**: Ellenőrzi, hogy a `getSummaryOfVotesInSession` metódus hibát dob-e, ha belső hiba történik. A `prisma.votingSession.findFirst` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `service.getSummaryOfVotesInSession` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "Error fetching currently running voting
      session: Database error",
        "status": 500
      }
      ```

- **VotesService -> getVotedSongsByUserId -> should return voted songs by user id**: Ellenőrzi, hogy a `getVotedSongsByUserId` metódus visszaadja-e a felhasználó által szavazott dalokat. A `prisma.votingSession.findFirst` és `prisma.vote.findMany` metódusokat mockolja, hogy egy szavazási szekciót és szavazatokat adjon vissza, és ellenőrzi, hogy a `service.getVotedSongsByUserId` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockVotingSession**:
      json
      ```
      {
        "id": "1",
        "start": "2022-11-30T00:00:00.000Z",
        "end": "2024-11-30T00:00:00.000Z",
        "songs": [],
      ```

```
      "Vote": []
    }
```

- **mockVotes**:

```
json
[
  { "songId": "1" },
  { "songId": "2" }
]
```

  - **Return value**:
    - **result**:

```
json
["1", "2"]
```

- **VotesService -> getVotedSongsByUserId -> should throw an error when no votes are found**: Ellenőrzi, hogy a `getVotedSongsByUserId` metódus hibát dob-e, ha nem találhatóak szavazatok. A `prisma.vote.findMany` metódust mockolja, hogy üres listát adjon vissza, és ellenőrzi, hogy a `service.getVotedSongsByUserId` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:

```
json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

  - **Return value**:
    - **error**:

```
json
{
  "message": "No active voting session found",
  "status": 404
}
```

- **VotesService -> getVotedSongsByUserId -> should throw an error when an internal error occurs**: Ellenőrzi, hogy a `getVotedSongsByUserId` metódus hibát dob-e, ha belső hiba történik. A `prisma.votingSession.findFirst` és `prisma.vote.findMany` metódusokat mockolja, hogy egy szavazási szekciót és hibát dobjon, és ellenőrzi, hogy a `service.getVotedSongsByUserId` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:

```
json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

- **mockVotingSession**:
  json
  ```
  {
    "id": "1",
    "start": "2022-11-30T00:00:00.000Z",
    "end": "2024-11-30T00:00:00.000Z",
    "songs": [],
    "Vote": []
  }
  ```
- **Return value**:
  - **error**:
    json
    ```
    {
      "message": "Error fetching votes: Database error",
      "status": 500
    }
    ```

- **VotesService -> voteUp -> should create a vote for a song**: Ellenőrzi, hogy a `voteUp` metódus létrehoz-e egy szavazatot egy dalra. A `prisma.votingSession.findFirst`, `prisma.vote.findFirst`, és `prisma.vote.create` metódusokat mockolja, hogy egy szavazási szekciót és szavazatot adjon vissza, és ellenőrzi, hogy a `service.voteUp` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockVotingSession**:
      json
      ```
      {
        "id": "1",
        "start": "2022-11-30T00:00:00.000Z",
        "end": "2024-11-30T00:00:00.000Z",
        "songs": [],
        "Vote": []
      }
      ```
  - **Return value**:
    - **result**:
      json
      ```
      {
        "message": "Vote successfully created for song: 1"
      }
      ```

- **VotesService -> voteUp -> should throw an error when user already voted for the song**: Ellenőrzi, hogy a `voteUp` metódus hibát dob-e, ha a felhasználó már szavazott a dalra. A `prisma.votingSession.findFirst` és `prisma.vote.findFirst` metódusokat mockolja, hogy egy szavazási szekciót és szavazatot adjon vissza, és ellenőrzi, hogy a `service.voteUp` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockVotingSession**:
      json
      ```
      {
        "id": "1",
        "start": "2022-11-30T00:00:00.000Z",
        "end": "2024-11-30T00:00:00.000Z",
        "songs": [],
        "Vote": []
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "User already voted for song: 1",
        "status": 400
      }
      ```
- **VotesService -> voteUp -> should throw an error when an internal error occurs**: Ellenőrzi, hogy a `voteUp` metódus hibát dob-e, ha belső hiba történik. A `prisma.votingSession.findFirst`, `prisma.vote.findFirst`, és `prisma.vote.create` metódusokat mockolja, hogy egy szavazási szekciót és hibát dobjon, és ellenőrzi, hogy a `service.voteUp` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```

- **mockVotingSession**:
  ```json
  {
    "id": "1",
    "start": "2022-11-30T00:00:00.000Z",
    "end": "2024-11-30T00:00:00.000Z",
    "songs": [],
    "Vote": []
  }
  ```
- **Return value**:
  - **error**:
    ```json
    {
      "message": "Error creating vote: Database error",
      "status": 500
    }
    ```

- **VotesService -> voteDown -> should delete a vote for a song**: Ellenőrzi, hogy a `voteDown` metódus töröl-e egy szavazatot egy dalra. A `prisma.votingSession.findFirst`, `prisma.vote.findFirst`, és `prisma.vote.delete` metódusokat mockolja, hogy egy szavazási szekciót és szavazatot adjon vissza, és ellenőrzi, hogy a `service.voteDown` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockVotingSession**:
      ```json
      {
        "id": "1",
        "start": "2022-11-30T00:00:00.000Z",
        "end": "2024-11-30T00:00:00.000Z",
        "songs": [],
        "Vote": []
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "message": "Sucessfully deleted vote for song: 1"
      }
      ```
- **VotesService -> voteDown -> should throw an error when no existing vote is found**: Ellenőrzi, hogy a `voteDown` metódus hibát dob-e, ha nem található meglévő szavazat. A `prisma.votingSession.findFirst` és `prisma.vote.findFirst` metódusokat mockolja, hogy egy szavazási szekciót és null értéket adjon vissza, és ellenőrzi, hogy a `service.voteDown` metódus a várt hibát dobja-e.

- **Mocked input data**:
  - **mockRequestUser**:
    ```json
    {
      "token": {
        "username": "testUser",
        "sub": "123"
      }
    }
    ```
  - **mockVotingSession**:
    ```json
    {
      "id": "1",
      "start": "2022-11-30T00:00:00.000Z",
      "end": "2024-11-30T00:00:00.000Z",
      "songs": [],
      "Vote": []
    }
    ```
- **Return value**:
  - **error**:
    ```json
    {
      "message": "No existing vote found",
      "status": 404
    }
    ```

- **VotesService -> voteDown -> should throw an error when an internal error occurs**: Ellenőrzi, hogy a `voteDown` metódus hibát dob-e, ha belső hiba történik. A `prisma.votingSession.findFirst`, `prisma.vote.findFirst`, és `prisma.vote.delete` metódusokat mockolja, hogy egy szavazási szekciót és hibát dobjon, és ellenőrzi, hogy a `service.voteDown` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
    - **mockVotingSession**:
      ```json
      {
        "id": "1",
        "start": "2022-11-30T00:00:00.000Z",
        "end": "2024-11-30T00:00:00.000Z",
        "songs": [],
        "Vote": []
      }
      ```
  - **Return value**:

- **error**:
  ```json
  {
    "message": "Error deleting vote for vote down: Database error",
    "status": 500
  }
  ```

voting.sessions.service.spec.ts

- **VotingSessionService -> should be defined**: Ellenőrzi, hogy a `VotingSessionService` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {}
      ```
- **VotingSessionService -> getAllSessions -> should return all sessions**: Ellenőrzi, hogy a `getAllSessions` metódus visszaadja-e az összes szavazási szekciót. A `prisma.votingSession.findMany` metódust mockolja, hogy egy szavazási szekció listát adjon vissza, és ellenőrzi, hogy a `service.getAllSessions` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      [
        {
          "id": "1",
          "start": "2020-01-01",
          "end": "2021-01-01",
          "songs": [],
          "Vote": []
        }
      ]
      ```

- **VotingSessionService -> getAllSessions -> should throw an error when no sessions are found**: Ellenőrzi, hogy a `getAllSessions` metódus hibát dob-e, ha nem találhatóak szavazási szekciók. A `prisma.votingSession.findMany` metódust mockolja, hogy üres listát adjon vissza, és ellenőrzi, hogy a `service.getAllSessions` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "No sessions found",
        "status": 404
      }
      ```
- **VotingSessionService -> getAllSessions -> should throw an error when an internal error occurs**: Ellenőrzi, hogy a `getAllSessions` metódus hibát dob-e, ha belső hiba történik. A `prisma.votingSession.findMany` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `service.getAllSessions` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "Error fetching sessions: Database error",
        "status": 500
      }
      ```
- **VotingSessionService -> createSession -> should create a new session**: Ellenőrzi, hogy a `createSession` metódus létrehoz-e egy új szavazási szekciót. A `prisma.song.findMany`, `prisma.votingSession.findFirst`, és `prisma.votingSession.create` metódusokat mockolja, hogy dalokat és szavazási szekciót adjon vissza, és ellenőrzi, hogy a `service.createSession` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:

- **mockSessionDto**:
  json
  ```
  {
    "start": "2020-01-01",
    "end": "2021-01-01",
    "songIds": ["1", "2"]
  }
  ```
- **mockRequestUser**:
  json
  ```
  {
    "token": {
      "username": "testUser",
      "sub": "123"
    }
  }
  ```

- **Return value**:
  - **result**:
    json
    ```
    {
      "message": "Successfully created new session:
    {\"id\":\"1\",\"start\":\"2020-01-01\",\"end\":\"2021-01-
    01\",\"songs\":[],\"Vote\":[]}"
    }
    ```

- **VotingSessionService -> createSession -> should throw an error when start time is later than end time**: Ellenőrzi, hogy a `createSession` metódus hibát dob-e, ha a kezdési idő későbbi, mint a befejezési idő. A `service.createSession` metódust hívja meg egy érvénytelen időtartammal, és ellenőrzi, hogy a várt hibát dobja-e.
  - **Mocked input data**:
    - **invalidSessionDto**:
      json
      ```
      {
        "start": "2022-01-01",
        "end": "2021-01-01",
        "songIds": ["1", "2"]
      }
      ```
    - **mockRequestUser**:
      json
      ```
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "The start time cannot be later than the end
      time",
        "status": 400
      }
      ```

- **VotingSessionService -> createSession -> should throw an error when overlapping session is found**: Ellenőrzi, hogy a `createSession` metódus hibát dob-e, ha átfedő szavazási szekció található. A `prisma.votingSession.findFirst` metódust mockolja, hogy egy meglévő szavazási szekciót adjon vissza, és ellenőrzi, hogy a `service.createSession` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockSessionDto**:
      ```json
      {
        "start": "2020-01-01",
        "end": "2021-01-01",
        "songIds": ["1", "2"]
      }
      ```
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "The duration of the voting session overlaps with an existing voting session",
        "status": 409
      }
      ```
- **VotingSessionService -> createSession -> should throw an error when an internal error occurs**: Ellenőrzi, hogy a `createSession` metódus hibát dob-e, ha belső hiba történik. A `prisma.song.findMany`, `prisma.votingSession.findFirst`, és `prisma.votingSession.create` metódusokat mockolja, hogy dalokat és hibát dobjon, és ellenőrzi, hogy a `service.createSession` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockSessionDto**:
      ```json
      {
        "start": "2020-01-01",
        "end": "2021-01-01",
        "songIds": ["1", "2"]
      }
      ```
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```

```
    }
```

- **Return value**:
  - **error**:

```json
{
  "message": "Error creating voting session: Database error",
  "status": 500
}
```

- **VotingSessionService -> updateSessionById -> should update a session by id**: Ellenőrzi, hogy a `updateSessionById` metódus frissíti-e a szavazási szekciót azonosító alapján. A `prisma.votingSession.findFirst`, `prisma.song.findMany`, és `prisma.votingSession.update` metódusokat mockolja, hogy dalokat és szavazási szekciót adjon vissza, és ellenőrzi, hogy a `service.updateSessionById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockSessionDto**:

```json
{
  "start": "2020-01-01",
  "end": "2021-01-01",
  "songIds": ["1", "2"]
}
```

    - **mockRequestUser**:

```json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

  - **Return value**:
    - **result**:

```json
{
  "message": "Successfully updated session: {\"id\":\"1\",\"start\":\"2020-01-01\",\"end\":\"2021-01-01\",\"songs\":[],\"Vote\":[]}"
}
```

- **VotingSessionService -> updateSessionById -> should throw an error when start time is later than end time**: Ellenőrzi, hogy a `updateSessionById` metódus hibát dob-e, ha a kezdési idő későbbi, mint a befejezési idő. A `service.updateSessionById` metódust hívja meg egy érvénytelen időtartammal, és ellenőrzi, hogy a várt hibát dobja-e.
  - **Mocked input data**:
    - **invalidSessionDto**:

```json
{
  "start": "2022-01-01",
  "end": "2021-01-01",
  "songIds": ["1", "2"]
```

```
    }
```

- **mockRequestUser**:

```json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

  - **Return value**:
    - **error**:

```json
{
  "message": "The start time cannot be later than the end time",
  "status": 400
}
```

- **VotingSessionService -> updateSessionById -> should throw an error when overlapping session is found**: Ellenőrzi, hogy a `updateSessionById` metódus hibát dob-e, ha átfedő szavazási szekció található. A `prisma.votingSession.findFirst` metódust mockolja, hogy egy meglévő szavazási szekciót adjon vissza, és ellenőrzi, hogy a `service.updateSessionById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockSessionDto**:

```json
{
  "start": "2020-01-01",
  "end": "2021-01-01",
  "songIds": ["1", "2"]
}
```

    - **mockRequestUser**:

```json
{
  "token": {
    "username": "testUser",
    "sub": "123"
  }
}
```

  - **Return value**:
    - **error**:

```json
{
  "message": "The duration of the voting session overlaps with an existing voting session",
  "status": 409
}
```

- **VotingSessionService -> updateSessionById -> should throw an error when an internal error occurs**: Ellenőrzi, hogy a `updateSessionById` metódus hibát dob-e, ha belső hiba történik. A `prisma.votingSession.findFirst`, `prisma.song.findMany`, és `prisma.votingSession.update` metódusokat mockolja, hogy dalokat és hibát dobjon, és ellenőrzi, hogy a `service.updateSessionById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockSessionDto**:
      ```json
      {
        "start": "2020-01-01",
        "end": "2021-01-01",
        "songIds": ["1", "2"]
      }
      ```
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "Error updating voting session: Database error",
        "status": 500
      }
      ```
- **VotingSessionService -> deleteSessionById -> should delete a session by id**: Ellenőrzi, hogy a `deleteSessionById` metódus törli-e a szavazási szekciót azonosító alapján. A `prisma.votingSession.findUnique` és `prisma.votingSession.delete` metódusokat mockolja, hogy egy szavazási szekciót adjon vissza, és ellenőrzi, hogy a `service.deleteSessionById` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {
        "message": "Successfully deleted session: 1"
      }
      ```

- **VotingSessionService -> deleteSessionById -> should throw an error when session is not found**: Ellenőrzi, hogy a `deleteSessionById` metódus hibát dob-e, ha a szavazási szekció nem található. A `prisma.votingSession.findUnique` metódust mockolja, hogy null értéket adjon vissza, és ellenőrzi, hogy a `service.deleteSessionById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "Voting session not found",
        "status": 404
      }
      ```
- **VotingSessionService -> deleteSessionById -> should throw an error when an internal error occurs**: Ellenőrzi, hogy a `deleteSessionById` metódus hibát dob-e, ha belső hiba történik. A `prisma.votingSession.findUnique` és `prisma.votingSession.delete` metódusokat mockolja, hogy egy szavazási szekciót és hibát dobjon, és ellenőrzi, hogy a `service.deleteSessionById` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **mockRequestUser**:
      ```json
      {
        "token": {
          "username": "testUser",
          "sub": "123"
        }
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "Error deleting voting session: Database error",
        "status": 500
      }
      ```

`snipper.service.spec.ts`

- **SnipperService -> addSong -> should throw an error if time delta is less than 5 seconds**: Ez a teszteset azt ellenőrzi, hogy a `SnipperService` addSong metódusa hibát dob-e, ha a megadott időintervallum kevesebb, mint 5 másodperc.
  - **Mocked input data**:
    - **songDto**:
      json
      ```
      {
        "ytUrl":
      "https://www.youtube.com/watch?v=JVpTp8IHdEg&list=RD6vNsAH
      xJXwE&index=10",
        "from": 0,
        "to": 4,
        "title": "Test Song"
      }
      ```
    - **request**:
      json
      ```
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "From-to time delta cannot be greater than 15
      seconds & cannot be less than 5",
        "status": 400
      }
      ```
- **SnipperService -> addSong -> should throw an error if time delta is greater than 15 seconds**: Ez a teszteset azt ellenőrzi, hogy a `SnipperService` addSong metódusa hibát dob-e, ha a megadott időintervallum nagyobb, mint 15 másodperc.
  - **Mocked input data**:
    - **songDto**:
      json
      ```
      {
        "ytUrl":
      "https://www.youtube.com/watch?v=JVpTp8IHdEg&list=RD6vNsAH
      xJXwE&index=10",
        "from": 0,
        "to": 18,
        "title": "Test Song"
      }
      ```

- **request**:
  json
  ```
  {
    "token": {
      "sub": "mockedId",
      "username": "mockedUser",
      "roles": ["Admin"],
      "hashedPassword": "mocked-password"
    }
  }
  ```
- **Return value**:
  - **error**:
    json
    ```
    {
      "message": "From-to time delta cannot be greater than 15 seconds & cannot be less than 5",
      "status": 400
    }
    ```

- **SnipperService -> addSong -> should throw an error if yt-dlp fails to get video info**: Ez a teszteset azt ellenőrzi, hogy a `SnipperService` addSong metódusa hibát dob-e, ha a `yt-dlp` parancs nem tudja lekérni a videó információit.
  - **Mocked input data**:
    - **songDto**:
      json
      ```
      {
        "ytUrl": "https://www.youtube.com/watch?v=JVpTp8IHdEg&list=RD6vNsAHxJXwE&index=10",
        "from": 0,
        "to": 10,
        "title": "Test Song"
      }
      ```
    - **request**:
      json
      ```
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **error**:
      json
      ```
      {
        "message": "Failed to get details about the youtube video",
        "status": 500
      }
      ```

- **SnipperService -> addSong -> should throw an error if video duration is greater than 10 minutes**: Ez a teszteset azt ellenőrzi, hogy a `SnipperService` addSong metódusa hibát dob-e, ha a videó hossza nagyobb, mint 10 perc.
  - **Mocked input data**:
    - **songDto**:
      ```json
      {
        "ytUrl":
      "https://www.youtube.com/watch?v=JVpTp8IHdEg&list=RD6vNsAH
      xJXwE&index=10",
        "from": 0,
        "to": 10,
        "title": "Test Song"
      }
      ```
    - **request**:
      ```json
      {
        "token": {
          "sub": "mockedId",
          "username": "mockedUser",
          "roles": ["Admin"],
          "hashedPassword": "mocked-password"
        }
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "message": "The maximum length ot the video cannot be
      greater than 10 minutes",
        "status": 400
      }
      ```
- **SnipperService -> addSong -> should download and convert the video to mp3**: Ez a teszteset azt ellenőrzi, hogy a `SnipperService` addSong metódusa sikeresen letölti és konvertálja a videót mp3 formátumba, majd hozzáadja az adatbázishoz.
  - **Mocked input data**:
    - **songDto**:
      ```json
      {
        "ytUrl":
      "https://www.youtube.com/watch?v=JVpTp8IHdEg&list=RD6vNsAH
      xJXwE&index=10",
        "from": 0,
        "to": 10,
        "title": "Test Song"
      }
      ```

- **request**:
  json
  ```
  {
    "token": {
      "sub": "mockedId",
      "username": "mockedUser",
      "roles": ["Admin"],
      "hashedPassword": "mocked-password"
    }
  }
  ```

– **Return value**:

- **result**:
  json
  ```
  {}
  ```

Backend Pipe Tesztelés

Az alábbiakban bemutatjuk az összes Pipe tesztet, amelyet a backend működésének ellenőrzésére alkalmaztunk. Minden fájl részletesen dokumentálva van, beleértve az összes benne található tesztesetet. A leírások tartalmazzák a mockolt bemeneti adatokat, valamint a várható visszatérési értékeket is.

`audio.length.validation.pipe.service.spec.ts`

- **AudioLengthValidatorPipe -> should be defined**: Ellenőrzi, hogy az `AudioLengthValidatorPipe` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **mockMetadataParser**:
      ```json
      {
        "parseFile": {}
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      {}
      ```
- **AudioLengthValidatorPipe -> should return true if file duration is within limit**: Ellenőrzi, hogy az `isValid` metódus igaz értéket ad-e vissza, ha a fájl hossza a megadott határon belül van. A `mockMetadataParser.parseFile` metódust mockolja, hogy egy 8 másodperces fájl hosszt adjon vissza, és ellenőrzi, hogy a `pipe.isValid` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **file**:
      ```json
      {
        "originalname": "test.mp3",
        "path": "/path/to/file.mp3"
      }
      ```
    - **mockMetadataParser**:
      ```json
      {
        "parseFile": {
          "format": {
            "duration": 8
          }
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      true
      ```

- **AudioLengthValidatorPipe -> should return false if file duration exceeds limit**: Ellenőrzi, hogy az `isValid` metódus hamis értéket ad-e vissza, ha a fájl hossza meghaladja a megadott határt. A `mockMetadataParser.parseFile` metódust mockolja, hogy egy 20 másodperces fájl hosszt adjon vissza, és ellenőrzi, hogy a `pipe.isValid` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **file**:
      ```json
      {
        "originalname": "test.mp3",
        "path": "/path/to/file.mp3"
      }
      ```
    - **mockMetadataParser**:
      ```json
      {
        "parseFile": {
          "format": {
            "duration": 20
          }
        }
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      false
      ```
- **AudioLengthValidatorPipe -> should return false if file has no path**: Ellenőrzi, hogy az `isValid` metódus hamis értéket ad-e vissza, ha a fájlnak nincs elérési útja. A `pipe.isValid` metódust hívja meg egy elérési út nélküli fájllal, és ellenőrzi, hogy a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **file**:
      ```json
      {
        "originalname": "test.mp3"
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      false
      ```
- **AudioLengthValidatorPipe -> should return false if an error occurs during parsing**: Ellenőrzi, hogy az `isValid` metódus hamis értéket ad-e vissza, ha hiba történik a fájl elemzése során. A `mockMetadataParser.parseFile` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy a `pipe.isValid` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **file**:
      ```json
      {
        "originalname": "test.mp3",
        "path": "/path/to/file.mp3"
      }
      ```

        - **mockMetadataParser**:
          ```json
          {
            "parseFile": {
              "error": "Parsing error"
            }
          }
          ```
    - **Return value**:
        - **result**:
          ```json
          false
          ```
- **AudioLengthValidatorPipe -> should return proper error message**: Ellenőrzi, hogy a `buildErrorMessage` metódus a megfelelő hibaüzenetet adja-e vissza. A `pipe.buildErrorMessage` metódust hívja meg egy fájllal, és ellenőrzi, hogy a várt hibaüzenetet adja-e vissza.
    - **Mocked input data**:
        - **file**:
          ```json
          {
            "originalname": "test.mp3"
          }
          ```
    - **Return value**:
        - **errorMessage**:
          ```json
          "The audio file test.mp3 exceeds the maximum allowed
          length of 15 seconds."
          ```

Backend Guard Tesztelés

Az alábbiakban bemutatjuk az összes Guard tesztet, amelyet a backend működésének ellenőrzésére alkalmaztunk. Minden fájl részletesen dokumentálva van, beleértve az összes benne található tesztesetet. A leírások tartalmazzák a mockolt bemeneti adatokat, valamint a várható visszatérési értékeket is.

`auth.guard.spec.ts`

- **AuthGuard -> should be defined**: Ellenőrzi, hogy az `AuthGuard` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **jwtService**:
      ```json
      {}
      ```
    - **reflector**:
      ```json
      {}
      ```
  - **Return value**:
    - **result**:
      ```json
      {}
      ```
- **AuthGuard -> should allow access to public endpoints**: Ellenőrzi, hogy az `AuthGuard` hozzáférést enged-e a nyilvános végpontokhoz. A `reflector.getAllAndOverride` metódust mockolja, hogy igaz értéket adjon vissza, és ellenőrzi, hogy az `authGuard.canActivate` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **context**:
      ```json
      {
        "switchToHttp": {
          "getRequest": {
            "cookies": {}
          }
        },
        "getHandler": {},
        "getClass": {}
      }
      ```
    - **reflector**:
      ```json
      {
        "getAllAndOverride": true
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      true
      ```

- **AuthGuard -> should throw UnauthorizedException if token is missing**: Ellenőrzi, hogy az `AuthGuard` `UnauthorizedException` hibát dob-e, ha a token hiányzik. A `reflector.getAllAndOverride` metódust mockolja, hogy hamis értéket adjon vissza, és ellenőrzi, hogy az `authGuard.canActivate` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **context**:
      ```json
      {
        "switchToHttp": {
          "getRequest": {
            "cookies": {}
          }
        },
        "getHandler": {},
        "getClass": {}
      }
      ```
    - **reflector**:
      ```json
      {
        "getAllAndOverride": false
      }
      ```
  - **Return value**:
    - **error**:
      ```json
      {
        "name": "UnauthorizedException"
      }
      ```
- **AuthGuard -> should throw HttpException if token is invalid**: Ellenőrzi, hogy az `AuthGuard` `HttpException` hibát dob-e, ha a token érvénytelen. A `reflector.getAllAndOverride` metódust mockolja, hogy hamis értéket adjon vissza, és a `jwtService.verifyAsync` metódust mockolja, hogy hibát dobjon, és ellenőrzi, hogy az `authGuard.canActivate` metódus a várt hibát dobja-e.
  - **Mocked input data**:
    - **context**:
      ```json
      {
        "switchToHttp": {
          "getRequest": {
            "cookies": {
              "token": "invalid-token"
            }
          }
        },
        "getHandler": {},
        "getClass": {}
      }
      ```
    - **reflector**:
      ```json
      {
        "getAllAndOverride": false
      ```

```
}
```

- **jwtService**:

```json
{
  "verifyAsync": {
    "error": "Invalid token"
  }
}
```

  - **Return value**:
    - **error**:

```json
{
  "name": "HttpException"
}
```

- **AuthGuard -> should allow access if token is valid**: Ellenőrzi, hogy az `AuthGuard` hozzáférést enged-e, ha a token érvényes. A `reflector.getAllAndOverride` metódust mockolja, hogy hamis értéket adjon vissza, és a `jwtService.verifyAsync` metódust mockolja, hogy egy felhasználói adatot adjon vissza, és ellenőrzi, hogy az `authGuard.canActivate` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **context**:

```json
{
  "switchToHttp": {
    "getRequest": {
      "cookies": {
        "token": "valid-token"
      },
      "token": {}
    }
  },
  "getHandler": {},
  "getClass": {}
}
```

    - **reflector**:

```json
{
  "getAllAndOverride": false
}
```

    - **jwtService**:

```json
{
  "verifyAsync": {
    "username": "testuser"
  }
}
```

  - **Return value**:
    - **result**:

```json
true
```

`role.guard.spec.ts`

- **RolesGuard -> should be defined**: Ellenőrzi, hogy a `RolesGuard` megfelelően definiálva van-e és példányosítva van-e a teszt modulban.
  - **Mocked input data**:
    - **reflector**:
      ```json
      {}
      ```
  - **Return value**:
    - **result**:
      ```json
      {}
      ```
- **RolesGuard -> should allow access if no roles are required**: Ellenőrzi, hogy a `RolesGuard` hozzáférést enged-e, ha nincs szükség szerepkörökre. A `reflector.getAllAndOverride` metódust mockolja, hogy undefined értéket adjon vissza, és ellenőrzi, hogy a `rolesGuard.canActivate` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **context**:
      ```json
      {
        "switchToHttp": {
          "getRequest": {
            "token": {
              "roles": []
            }
          }
        },
        "getHandler": {
          "name": "Handler"
        },
        "getClass": {}
      }
      ```
    - **reflector**:
      ```json
      {
        "getAllAndOverride": null
      }
      ```
  - **Return value**:
    - **result**:
      ```json
      true
      ```
- **RolesGuard -> should allow access if user has required roles**: Ellenőrzi, hogy a `RolesGuard` hozzáférést enged-e, ha a felhasználónak megvannak a szükséges szerepkörök. A `reflector.getAllAndOverride` metódust mockolja, hogy az `Admin` szerepkört adja vissza, és ellenőrzi, hogy a `rolesGuard.canActivate` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:

- **context**:
  ```json
  {
    "switchToHttp": {
      "getRequest": {
        "token": {
          "roles": ["Admin"]
        }
      }
    },
    "getHandler": {
      "name": "Handler"
    },
    "getClass": {}
  }
  ```
- **reflector**:
  ```json
  {
    "getAllAndOverride": ["Admin"]
  }
  ```

- **Return value**:
  - **result**:
    ```json
    true
    ```

- **RolesGuard -> should deny access if user does not have required roles**: Ellenőrzi, hogy a `RolesGuard` megtagadja-e a hozzáférést, ha a felhasználónak nincsenek meg a szükséges szerepkörök. A `reflector.getAllAndOverride` metódust mockolja, hogy az `Admin` szerepkört adja vissza, és ellenőrzi, hogy a `rolesGuard.canActivate` metódus a várt eredményt adja-e vissza.
  - **Mocked input data**:
    - **context**:
      ```json
      {
        "switchToHttp": {
          "getRequest": {
            "token": {
              "roles": ["User"]
            }
          }
        },
        "getHandler": {
          "name": "Handler"
        },
        "getClass": {}
      }
      ```
    - **reflector**:
      ```json
      {
        "getAllAndOverride": ["Admin"]
      }
      ```
  - **Return value**:
    - **result**: ***false***

Nevezetes algoritmusok

A nevezetes algoritmusok hangsúlyozzák a projekt kulcsfontosságú algoritmusait, amelyek kulcsfontosságú szerepet játszanak a projekt működésében és teljesítményében. Az algoritmusok leírása és kódja segít megérteni azok működését és hatását a projektre. Az alábbi algoritmusok, ahol lehetséges backend-frontend párokat képezve, a projekt kulcsfontosságú részeit mutatják be.

Minden részleg végén megtekintheti magát a kódot is kép formájában.

Regisztrációs algoritmus

Backenden

Az `AuthService` osztály `register` metódusa egy új felhasználó regisztrációját végzi el. A metódus első lépésként naplózza a regisztrációs kísérletet a kapott `registerDto` objektum JSON formátumú adataival. Ezután a `prisma.user` szolgáltatás segítségével ellenőrzi, hogy létezik-e már felhasználó a megadott felhasználónévvel vagy OM azonosítóval. Ha létezik ilyen felhasználó, akkor egy `HttpException` kivételt dob, jelezve, hogy a felhasználó már regisztrálva van. Ha nem található ilyen felhasználó, akkor a `prisma.kreta` szolgáltatás segítségével ellenőrzi, hogy az OM azonosító megtalálható-e az adatbázisban. Ha nem található, akkor egy másik `HttpException` kivételt dob, jelezve, hogy az OM azonosító nem található az adatbázisban. Ha az OM azonosító megtalálható, akkor a `bcrypt` könyvtár segítségével hasheli a felhasználó jelszavát, majd létrehozza az új felhasználót a `prisma.user` szolgáltatás segítségével. Az új felhasználó létrehozása során a felhasználónév, jelszó, email és az OM azonosító is mentésre kerül, valamint a felhasználó szerepköre is beállításra kerül. Ha a felhasználó létrehozása sikeres, akkor egy JWT token generálódik a felhasználó adataival, és ez a token beállításra kerül a válasz sütiében. Végül a metódus naplózza a sikeres regisztrációt és visszaadja az access token-t. Az algoritmus célja, hogy biztonságosan és hatékonyan kezelje az új felhasználók regisztrációját, ellenőrizve a meglévő felhasználókat és az OM azonosítókat, valamint biztosítva a jelszavak biztonságos tárolását és a felhasználók hitelesítését.

Maga a kód:

```ts
/**
* Registers a new user.
* @param {RegisterDto} registerDto - The registration data transfer object.
* @param {RequestUser} request - The request user object.
* @param {Response} response - The response object to set cookies.
* @returns {Promise<object>} The access token.
* @throws {HttpException} If the user already exists or registration fails.
*/
async register(registerDto: RegisterDto, request: RequestUser, response: Response): Promise<object> {
this.logger.verbose(`Register attempt by ${JSON.stringify(registerDto)}`);

    const user = await this.prisma.user
        .findFirst({
            where: {
                OR: [
                    {
                        username: registerDto.username,
                    },
                    {
                        kreta: {
                            om: BigInt(registerDto.om),
                        },
                    },
                ],
            },
        })
        .catch((e) => {
            this.logger.error(`Something went wrong when checking the new users username and OM ${JSON.stringify(registerDto)}`, e);
            throw new HttpException(`Something went wrong`, HttpStatus.INTERNAL_SERVER_ERROR);
        });

    if (user) {
        this.logger.error(`User already registered ${JSON.stringify(registerDto)}`);
        throw new HttpException(`Seems like someone has already registered under this information`, HttpStatus.INTERNAL_SERVER_ERROR)
    }

    await this.prisma.kreta
        .findFirstOrThrow({
            where: {
                om: BigInt(registerDto.om),
            },
        })
        .catch((e) => {
            this.logger.error(`Failed to register user ${JSON.stringify(registerDto)}`, e);
            throw new HttpException(`Can't find this OM id in the database`, HttpStatus.INTERNAL_SERVER_ERROR);
        });

    const hashedPassword = await bcrypt.hash(registerDto.password, 10);

    const newUser = await this.prisma.user
        .create({
            data: {
                username: registerDto.username,
                password: hashedPassword,
                email: registerDto.email,
                kreta: {
                    connect: {
                        om: BigInt(registerDto.om),
                    },
                },
                roles: {
                    connectOrCreate: {
                        where: {
                            role: RoleEnum.User,
                        },
                        create: {
                            role: RoleEnum.User,
                        },
                    },
                },
            },
        })
        .catch((e) => {
            this.logger.error(`Failed to register user ${JSON.stringify(registerDto)}`, e);
            throw new HttpException(`Failed to register user bad OM`, HttpStatus.INTERNAL_SERVER_ERROR);
        });

    const payload: JwtPayload = {
        sub: newUser.id,
        username: newUser.username,
        hashedPassword: newUser.password,
        roles: [RoleEnum.User],
    };

    const token = await this.jwtService.signAsync(payload);

    response.cookie('token', token, {
        httpOnly: false,
        secure: !process.env.DEV,
        domain: process.env.CORS_DOMAIN,
        sameSite: 'none',
    });

    this.logger.verbose(`Successfully registered user ${JSON.stringify(newUser)}`);

    return { access_token: token };
}
```

Frontenden

A `useRegisterStore` egy Pinia store, amely a felhasználói regisztráció kezelésére szolgál egy Vue.js alkalmazásban. Az algoritmus célja, hogy a felhasználó által megadott regisztrációs adatokat elküldje a szervernek, és a sikeres regisztráció után beállítsa a hitelesítési sütit, majd átirányítsa a felhasználót a főoldalra. A `fetchRegister` függvény először ellenőrzi, hogy a megadott jelszó és a jelszó megerősítése megegyeznek-e. Ha nem, akkor hibát dob. Ezután egy HTTP POST kérést küld a szervernek a regisztrációs adatokkal, és a válaszból JSON formátumban kapja meg az adatokat. Ha a válasz nem sikeres, akkor a szerver által küldött hibaüzenetet dobja. Ha a válasz sikeres, akkor a kapott hozzáférési tokent beállítja egy sütiben, amely egy hétig érvényes, és a sütit a megadott domainre és biztonsági beállításokkal állítja be. Végül a felhasználót átirányítja a főoldalra. Az algoritmus célja, hogy biztonságosan és hatékonyan kezelje a felhasználói regisztrációt, ellenőrizve a jelszavak egyezését, és biztosítva a felhasználó hitelesítését a sütik segítségével.

Maga a kód:

```
export const useRegisterStore = defineStore('register', () => {
  const url = import.meta.env.VITE_API_URL
  const isTestEnvironment = import.meta.env.VITE_TEST === 'true';
  const domain = import.meta.env.VITE_COOKIE_DOMAIN;
  const logger = serviceLogger('useRegisterStore');

  const cookies = useCookies()
  const router = useRouter()

  async function fetchRegister(credentials: { username: string, password: string, email: string, om: string }, passwordConfirmation:
    try {
      if (credentials.password !== passwordConfirmation) {
        throw new Error('A jelszavak nem egyeznek meg!')
      }

      const response = await fetch(`${url}/api/auth/register`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify(credentials)
      })
      const data = await response.json()

      if (!response.ok) {
        throw new Error(data.message)
      }

      logger.debug('Response:', data)

      cookies.set('token', data.access_token, {
        expires: new Date(Date.now() + 1000 * 60 * 60 * 24 * 7),
        secure: !isTestEnvironment,
        domain,
        sameSite: 'none'
      })
      await router.push('/')
    } catch (error) {
      throw new Error((error as Error).message)
    }
  }

  return {
    fetchRegister
  }
})
```

Zenefeltöltési algoritmus

Backenden

Az `upload` metódus a `SongsController` osztályban egy új dal feltöltését kezeli egy adott felhasználó által. A metódus a `POST` HTTP metódust használja, és a `@HttpCode(HttpStatus.OK)` annotációval jelzi, hogy a sikeres válasz státuszkódja 200 lesz. A `@UseFilters(DeleteFileOnErrorFilter)` annotáció biztosítja, hogy ha hiba történik a fájl feltöltése közben, akkor a feltöltött fájl törlésre kerül. A `@UseInterceptors(FileInterceptor('file'))` annotáció a fájl feltöltését kezeli, és a `@UploadedFile` paraméterrel kapja meg a feltöltött fájlt. A `ParseFilePipe` validátorokkal ellenőrzi, hogy a fájl típusának `audio/mpeg`-nek kell lennie, és a fájl hossza nem haladhatja meg a 15 másodpercet. A metódus először naplózza a feltöltött fájlt és a felhasználói tokent. Ezután meghívja a `SongsService` osztály `upload` metódusát, amely a feltöltött fájlt a `pendingSong` adatbázis táblába menti. A `SongsService` osztály `upload` metódusa először naplózza a feltöltési kísérletet, majd a `prisma.pendingSong.create` metódussal létrehozza az új bejegyzést az adatbázisban. Ha hiba történik, akkor naplózza a hibát és `HttpException` kivételt dob. Ha a feltöltés sikeres, akkor naplózza a sikeres feltöltést és visszaadja az új fájl adatait. Az algoritmus célja, hogy biztonságosan és hatékonyan kezelje a dalok feltöltését, ellenőrizve a fájl típusát és hosszát, valamint biztosítva a fájlok megfelelő tárolását az adatbázisban.

Maga a kód:

```
@Post('audio')
@HttpCode(HttpStatus.OK)
@UseFilters(DeleteFileOnErrorFilter)
@UseInterceptors(FileInterceptor('file'))
async upload(
    @Body() createSongDto: CreateSongDto,
    @Req() request: RequestUser,
    @UploadedFile(
        new ParseFilePipe({
            validators: [new FileTypeValidator({ fileType: 'audio/mpeg' }), new AudioLengthValidatorPipe({ length: 15 })],
        }),
    )
    file: Express.Multer.File,
): Promise<Prisma.Args<typeof this.prisma.pendingSong, 'create'>['data']> {
    this.logger.debug(`File uploaded successfully: ${JSON.stringify(file)}, ${JSON.stringify(request.token)}`);
    return this.appService.upload(createSongDto, request, file);
}

async upload(
    createSongDto: CreateSongDto,
    request: RequestUser,
    file: Express.Multer.File,
): Promise<Prisma.Args<typeof this.prisma.pendingSong, 'create'>['data']> {
    this.logger.verbose(`Uploading song into pending songs by user: ${JSON.stringify(request.token.username)}`);

    const newFile = await this.prisma.pendingSong
        .create({
            data: {
                title: createSongDto.title,
                uploadedBy: {
                    connect: {
                        id: request.token.sub,
                    },
                },
                songBucket: {
                    create: {
                        path: file.path,
                    },
                },
            },
        })
        .catch((error) => {
            this.logger.error(`Error uploading song into pending songs by user: ${JSON.stringify(request.token.username)}`);
            throw new HttpException(`Error uploading song into pending songs: ${error.message}`, HttpStatus.INTERNAL_SERVER_ERROR);
        });

    this.logger.verbose(`Successfully uploaded song into pending songs by user: ${JSON.stringify(request.token.username)}`);

    return newFile;
}
```

Frontenden

A `useUploadInputStore` egy Pinia store, amely a fájlok feltöltésének kezelésére szolgál egy Vue.js alkalmazásban. Az algoritmus célja, hogy a felhasználó által kiválasztott fájlt elküldje a szervernek, és a sikeres feltöltés után naplózza az eredményt. A store négy fő funkciót tartalmaz: `setUrl`, `setFile`, `uploadFile`, és a fájl és URL állapotának kezelésére szolgáló reaktív változókat. A `setUrl` függvény beállítja a feltöltési URL-t, míg a `setFile` függvény beállítja a feltöltendő fájlt. Az `uploadFile` függvény egy `FormData` objektumot hoz létre, amely tartalmazza a fájlt és annak nevét. Ezután egy HTTP POST kérést küld a megadott URL-re, a fájl adataival a kérés törzsében. A válasz JSON formátumban érkezik, és ha a válasz nem sikeres, akkor a szerver által küldött hibaüzenetet dobja. Ha a válasz sikeres, a fájl változó nullára állítódik, és a sikeres feltöltést naplózza. Az algoritmus célja, hogy biztonságosan és hatékonyan kezelje a fájlok feltöltését, ellenőrizve a válasz sikerességét, és biztosítva a megfelelő naplózást a hibák és sikeres feltöltések esetén.

Maga a kód:

```ts
export const useUploadInputStore = defineStore('uploadInput', () => {
  const logger = serviceLogger('uploadInputStore')

  const url = ref<string>('')

  const file = ref<File | null>(null)

  function setUrl(newUrl: string) {
    url.value = newUrl
  }

  function setFile(newFile: File) {
    file.value = newFile
  }

  async function uploadFile() {
    const body = new FormData()
    body.append('file', file.value as Blob)
    body.append('title', file.value?.name as string)

    try {
      const response = await fetch(url.value, {
        method: 'POST',
        body,
        credentials: 'include'
      })
      file.value = null
      const data = await response.json()
      logger.debug(`File uploaded: ${JSON.stringify(response)}`)
      if (!response.ok) {
        throw new Error(data.message)
      }
    } catch (error) {
      const err = error as Error
      throw new Error(`Failed to upload file: ${err.message}`)
    }

  }

  return {
    file,
    uploadFile,
    setUrl,
    setFile
  }
})
```

Szavazási algoritmus

Backenden

A `voteUp` és `voteDown` metódusok a `VotesService` osztályban a felhasználói szavazatok kezelésére szolgálnak egy adott dalra vonatkozóan. A `voteUp` metódus célja, hogy egy felhasználó szavazatát hozzáadja egy dalhoz. Először naplózza a szavazási kísérletet, majd ellenőrzi, hogy a felhasználói azonosító (`userId`) jelen van-e a kérésben. Ha hiányzik, `HttpException` kivételt dob. Ezután ellenőrzi, hogy a felhasználó már szavazott-e az adott dalra az aktuális szavazási szekcióban. Ha már szavazott, újabb `HttpException` kivételt dob. Ha még nem szavazott, létrehoz egy új szavazatot a `prisma.vote.create` metódussal, és naplózza a sikeres szavazást. A `voteDown` metódus célja, hogy egy felhasználó szavazatát eltávolítsa egy dalról. Először naplózza a szavazat eltávolítási kísérletet, majd ellenőrzi, hogy a felhasználói azonosító jelen van-e a kérésben. Ezután ellenőrzi, hogy létezik-e már szavazat az adott dalra az aktuális szavazási szekcióban. Ha nem található szavazat, `HttpException` kivételt dob. Ha található szavazat, törli azt a `prisma.vote.delete` metódussal, és naplózza a sikeres törlést. Mindkét metódus célja, hogy biztonságosan és hatékonyan kezelje a felhasználói szavazatokat, ellenőrizve a meglévő szavazatokat és biztosítva a megfelelő naplózást a hibák és sikeres műveletek esetén.

Maga a kód:

```
/**
 * Vote up on song
 * @param id - id of the song
 * @param request - user request
 * @returns
 */
async voteUp(id: string, request: RequestUser): Promise<string> {
    this.logger.verbose(`Voteing up on song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);

    const userId = request.token.sub;

    if (!userId) throw new HttpException(`Missing userId`, HttpStatus.BAD_REQUEST);

    const existingVote = await this.prisma.vote
        .findFirst({
            where: {
                userId,
                songId: id,
                sessionId: await this.getCurrentVotingSessionId(),
            },
        })
        .catch((error) => {
            this.logger.error(`Error fetching existing vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);
            throw new HttpException(`Error fetching existing vote ${error}`, HttpStatus.BAD_REQUEST);
        });

    if (existingVote) throw new HttpException(`User already voted for song: ${id}`, HttpStatus.BAD_REQUEST);

    await this.prisma.vote
        .create({
            data: {
                userId,
                songId: id,
                sessionId: await this.getCurrentVotingSessionId(),
            },
        })
        .catch((error) => {
            this.logger.error(`Error creating vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);
            throw new HttpException(`Error creating vote: ${error.message}`, HttpStatus.INTERNAL_SERVER_ERROR);
        });

    this.logger.verbose(`Successfully created vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);

    return JSON.stringify({ message: `Vote successfully created for song: ${id}` });
}

async voteDown(id: string, request: RequestUser): Promise<string> {
    this.logger.verbose(`Voteing down on song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);

    const userId = request.token.sub;

    const existingVote = await this.prisma.vote
        .findFirst({
            where: {
                userId,
                songId: id,
                sessionId: await this.getCurrentVotingSessionId(),
            },
        })
        .catch((error) => {
            this.logger.error(`Error fetching existing vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);
            throw new HttpException(`Error fetching existing vote for vote down: ${error.message}`, HttpStatus.INTERNAL_SERVER_ERROR);
        });

    if (!existingVote) throw new HttpException(`No existing vote found`, HttpStatus.NOT_FOUND);

    await this.prisma.vote
        .delete({
            where: {
                id: existingVote.id,
            },
        })
        .catch((error) => {
            this.logger.error(`Error deleting vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);
            throw new HttpException(`Error deleting vote for vote down: ${error.message}`, HttpStatus.INTERNAL_SERVER_ERROR);
        });

    this.logger.verbose(`Successfully deleted vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);

    return JSON.stringify({ message: `Sucessfully deleted vote for song: ${id}` });
}
```

Frontenden

A `useVoteSongStore` egy Pinia store, amely a felhasználói szavazatok kezelésére szolgál egy Vue.js alkalmazásban. Az algoritmus célja, hogy a felhasználók szavazatait kezelje dalokra vonatkozóan, lehetővé téve számukra, hogy szavazzanak egy dalra (voteUp) vagy visszavonják a szavazatukat (voteDown). A store négy fő funkciót tartalmaz: `fetchVotesByUser`, `voteUp`, `voteDown`, valamint a szavazatok és a szavazási állapot (`hasVotes`) kezelésére szolgáló reaktív változókat. A `fetchVotesByUser` függvény HTTP GET kérést küld a szervernek, hogy lekérje a felhasználó aktuális szavazatait. Ha a válasz státuszkódja 404, akkor a `hasVotes` értéke hamisra áll, és a `votes` üres tömbre állítódik. Ha a válasz sikeres, a `hasVotes` igazra áll, és a `votes` változó a kapott adatokat tartalmazza. A `voteUp` függvény HTTP POST kérést küld a szervernek, hogy egy új szavazatot hozzon létre egy adott dalra. Ha a válasz nem sikeres, hibát dob. A `voteDown` függvény HTTP DELETE kérést küld a szervernek, hogy törölje a felhasználó szavazatát egy adott dalról. Ha a válasz nem sikeres, hibát dob. Az algoritmus célja, hogy biztonságosan és hatékonyan kezelje a felhasználói szavazatokat, ellenőrizve a válaszok sikerességét, és biztosítva a megfelelő naplózást a hibák és sikeres műveletek esetén.

Maga a kód:

```ts
export const useVoteSongStore = defineStore('voteSong', () => {
    const logger = serviceLogger('voteSongStore');
    const url = import.meta.env.VITE_API_URL;

    const hasVotes = ref(false);

    const votes = ref<string[]>(['']);

    async function fetchVotesByUser() {
        try {
            const response = await fetch(`${url}/api/votes/current-user`, {
                credentials: 'include',
            });
            const data = await response.json();
            logger.debug(`Votes: ${JSON.stringify(data)}`);
            logger.debug(`Response status ${response.status}`);
            if (response.status === 404) {
                logger.debug(`Recieved ${response.status} status`);
                hasVotes.value = false;
                votes.value = [''];
                logger.debug(`Has votes: ${hasVotes.value}`);
                return;
            }
            if (!response.ok) {
                throw new Error(`Failed to fetch votes ${data.message}`);
            }
            hasVotes.value = true;
            votes.value = data;
            logger.debug(`Has votes: ${hasVotes.value}`);
        } catch (err) {
            const error = err as Error;
            logger.error(`Error fetching votes: ${error.message}`);
            throw new Error(`Error fetching votes: ${error.message}`);
        }
    }

    async function voteUp(id: string) {
        try {
            // Optional. Only put this to use if you want to limit every users ability to only vote for 1 song in each session.
            // if (hasVotes.value) {
            //   votes.value.map(songId => voteDown(songId))
            // }
            const response = await fetch(`${url}/api/votes?id=${id}`, {
                method: 'POST',
                credentials: 'include',
            });
            const data = await response.json();
            if (!response.ok) {
                throw new Error(`Failed to vote up ${data.message}`);
            }
            logger.debug(`Has votes ${hasVotes.value}`);
        } catch (err) {
            const error = err as Error;
            logger.error(`Error voting up: ${error.message}`);
            throw new Error(`Error voting up: ${error.message}`);
        }
    }

    async function voteDown(id: string) {
        try {
            const response = await fetch(`${url}/api/votes?id=${id}`, {
                method: 'DELETE',
                credentials: 'include',
            });
            const data = await response.json();
            if (!response.ok) {
                throw new Error(`Failed to vote up ${data.message}`);
            }
        } catch (err) {
            const error = err as Error;
            logger.error(`Error voting up: ${error.message}`);
            throw new Error(`Error voting up: ${error.message}`);
        }
    }

    return {
        votes,
        hasVotes,
        fetchVotesByUser,
        voteUp,
        voteDown,
    };
});
```

Statisztikáző algoritmus

Backenden

A `getSummaryOfVotesInSession` metódus a `TvService` osztályban a jelenlegi szavazási szekció összesített szavazatainak lekérésére szolgál. Az algoritmus célja, hogy összegyűjtse és visszaadja a szavazatok számát minden egyes dalra vonatkozóan az aktuális szavazási szekcióban. A metódus először naplózza a szavazatok összegzésének lekérési kísérletét, majd lekéri az aktuális dátumot és időt. Ezután a `prisma.votingSession.findFirst` metódussal megkeresi az aktuális szavazási szekciót, amelynek kezdete kisebb vagy egyenlő az aktuális dátummal és vége nagyobb vagy egyenlő az aktuális dátummal. Ha hiba történik a szekció lekérése közben, naplózza a hibát és `HttpException` kivételt dob. Ha nem található aktív szavazási szekció, `HttpException` kivételt dob. Ha sikeresen lekérte a szavazási szekciót, a metódus megpróbálja összeszámolni a szavazatokat minden egyes dalra vonatkozóan. A `votingSession.songs.map` metódussal végigmegy a dalokon, és megszámolja, hogy hány szavazat érkezett minden egyes dalra a `votingSession.Vote.filter` metódussal. Az eredményeket egy objektumba rendezi, amely tartalmazza a dal azonosítóját, címét és a szavazatok számát. Ezután a dalokat a szavazatok száma szerint csökkenő sorrendbe rendezi. Ha hiba történik a szavazatok összeszámolása közben, naplózza a hibát és `HttpException` kivételt dob. Ha sikeresen összeszámolta a szavazatokat, naplózza a sikeres műveletet, és visszaadja az összesített szavazatok JSON formátumú eredményét, amely tartalmazza a szavazási szekció azonosítóját és a dalok szavazatainak összegzését. Az algoritmus célja, hogy biztonságosan és hatékonyan kezelje a szavazatok összesítését, ellenőrizve a szavazási szekció érvényességét és biztosítva a megfelelő naplózást a hibák és sikeres műveletek esetén.

Maga a kód:

```
/**
 * Vote up on song
 * @param id - id of the song
 * @param request - user request
 * @returns
 */
async voteUp(id: string, request: RequestUser): Promise<string> {
    this.logger.verbose(`Voteing up on song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);

    const userId = request.token.sub;

    if (!userId) throw new HttpException(`Missing userId`, HttpStatus.BAD_REQUEST);

    const existingVote = await this.prisma.vote
        .findFirst({
            where: {
                userId,
                songId: id,
                sessionId: await this.getCurrentVotingSessionId(),
            },
        })
        .catch((error) => {
            this.logger.error(`Error fetching existing vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)
            throw new HttpException(`Error fetching existing vote ${error}`, HttpStatus.BAD_REQUEST);
        });

    if (existingVote) throw new HttpException(`User already voted for song: ${id}`, HttpStatus.BAD_REQUEST);

    await this.prisma.vote
        .create({
            data: {
                userId,
                songId: id,
                sessionId: await this.getCurrentVotingSessionId(),
            },
        })
        .catch((error) => {
            this.logger.error(`Error creating vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);
            throw new HttpException(`Error creating vote: ${error.message}`, HttpStatus.INTERNAL_SERVER_ERROR);
        });

    this.logger.verbose(`Successfully created vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);

    return JSON.stringify({ message: `Vote successfully created for song: ${id}` });
}

async voteDown(id: string, request: RequestUser): Promise<string> {
    this.logger.verbose(`Voteing down on song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);

    const userId = request.token.sub;

    const existingVote = await this.prisma.vote
        .findFirst({
            where: {
                userId,
                songId: id,
                sessionId: await this.getCurrentVotingSessionId(),
            },
        })
        .catch((error) => {
            this.logger.error(`Error fetching existing vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)
            throw new HttpException(`Error fetching existing vote for vote down: ${error.message}`, HttpStatus.INTERNAL_SERVER_ERROR)
        });

    if (!existingVote) throw new HttpException(`No existing vote found`, HttpStatus.NOT_FOUND);

    await this.prisma.vote
        .delete({
            where: {
                id: existingVote.id,
            },
        })
        .catch((error) => {
            this.logger.error(`Error deleting vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);
            throw new HttpException(`Error deleting vote for vote down: ${error.message}`, HttpStatus.INTERNAL_SERVER_ERROR);
        });

    this.logger.verbose(`Successfully deleted vote for song with id: ${id} by user: ${JSON.stringify(request.token.username)}`);

    return JSON.stringify({ message: `Sucessfully deleted vote for song: ${id}` });
}
```

Frontenden

A `useTvStore` egy Pinia store, amely a TV szavazási szekció összefoglalójának kezelésére szolgál egy Vue.js alkalmazásban. Az algoritmus célja, hogy a felhasználók számára megjelenítse az aktuális szavazási szekcióban lévő dalok listáját és azok szavazatszámát, valamint minden dalhoz egy véletlenszerű színt rendeljen. A store négy fő elemet tartalmaz: a `songs` és `colors` reaktív változókat, a `getRandomColor` függvényt, valamint a `fetchSummaryOfVotesInSession` aszinkron függvényt. A `songs` változó egy tömböt tartalmaz, amely az egyes dalok azonosítóját, címét és szavazatszámát tárolja. A `getRandomColor` függvény egy véletlenszerű színt generál minden dalhoz, amelyet a `colors` változó tárol. A `fetchSummaryOfVotesInSession` függvény HTTP GET kérést küld a szervernek, hogy lekérje az aktuális szavazási szekció összefoglalóját. Ha a válasz státuszkódja 404, akkor a `songs` változó üres tömbre állítódik, és a `logger` naplózza a hibát. Ha a válasz sikeres, a `songs` változó a kapott adatokat tartalmazza, és a `colors` változó minden dalhoz egy véletlenszerű színt rendel. Az algoritmus célja, hogy biztonságosan és hatékonyan kezelje a szavazási szekció összefoglalójának lekérését, ellenőrizve a válaszok sikerességét, és biztosítva a megfelelő naplózást a hibák és sikeres műveletek esetén.

Maga a kód:

```ts
export const useVoteSongStore = defineStore('voteSong', () => {
    const logger = serviceLogger('voteSongStore');
    const url = import.meta.env.VITE_API_URL;

    const hasVotes = ref(false);

    const votes = ref<string[]>(['']);

    async function fetchVotesByUser() {
        try {
            const response = await fetch(`${url}/api/votes/current-user`, {
                credentials: 'include',
            });
            const data = await response.json();
            logger.debug(`Votes: ${JSON.stringify(data)}`);
            logger.debug(`Response status ${response.status}`);
            if (response.status === 404) {
                logger.debug(`Recieved ${response.status} status`);
                hasVotes.value = false;
                votes.value = [''];
                logger.debug(`Has votes: ${hasVotes.value}`);
                return;
            }
            if (!response.ok) {
                throw new Error(`Failed to fetch votes ${data.message}`);
            }
            hasVotes.value = true;
            votes.value = data;
            logger.debug(`Has votes: ${hasVotes.value}`);
        } catch (err) {
            const error = err as Error;
            logger.error(`Error fetching votes: ${error.message}`);
            throw new Error(`Error fetching votes: ${error.message}`);
        }
    }

    async function voteUp(id: string) {
        try {
            // Optional. Only put this to use if you want to limit every users ability to only vote for 1 song in each session.
            // if (hasVotes.value) {
            //   votes.value.map(songId => voteDown(songId))
            // }
            const response = await fetch(`${url}/api/votes?id=${id}`, {
                method: 'POST',
                credentials: 'include',
            });
            const data = await response.json();
            if (!response.ok) {
                throw new Error(`Failed to vote up ${data.message}`);
            }
            logger.debug(`Has votes ${hasVotes.value}`);
        } catch (err) {
            const error = err as Error;
            logger.error(`Error voting up: ${error.message}`);
            throw new Error(`Error voting up: ${error.message}`);
        }
    }

    async function voteDown(id: string) {
        try {
            const response = await fetch(`${url}/api/votes?id=${id}`, {
                method: 'DELETE',
                credentials: 'include',
            });
            const data = await response.json();
            if (!response.ok) {
                throw new Error(`Failed to vote up ${data.message}`);
            }
        } catch (err) {
            const error = err as Error;
            logger.error(`Error voting up: ${error.message}`);
            throw new Error(`Error voting up: ${error.message}`);
        }
    }

    return {
        votes,
        hasVotes,
        fetchVotesByUser,
        voteUp,
        voteDown,
    };
});
```

Event Bus algoritmus

Az Event Bus egy olyan kommunikációs minta, amely lehetővé teszi az alkalmazás különböző komponensei közötti események továbbítását és kezelését. Az Event Bus központi szerepet játszik az események küldésében és fogadásában, így a komponensek közvetlen kommunikáció nélkül is tudnak egymással interakcióba lépni. Az események küldésekor az Event Bus értesíti az összes feliratkozott komponenst, amelyek reagálhatnak az adott eseményre. Ez a minta segít a laza csatolás megvalósításában és az alkalmazás modularitásának növelésében.

Maga a kód:

```
const eventBus = ref(new Map<string, Set<Function>>())

export function on(event: string, callback: Function) {
  console.log('Registering event', event)
  if (!eventBus.value.has(event)) {
    eventBus.value.set(event, new Set())
  }
  eventBus.value.get(event)?.add(callback)
}

export function emit(event: string, ...args: any[]) {
  console.log('Emitting event', event, args)
  const callbacks = eventBus.value.get(event)
  if (callbacks) {
    callbacks.forEach(callback => callback(...args))
  }
}
```

Fejlesztési lehetőségek

Az alábbiakban három különböző fejlesztési lehetőséget mutatunk be, amelyek mindegyike egyedi módon járulna hozzá a felhasználói élmény javításához és az automatizáció hatékonyságának növeléséhez. Ezek a megoldások különböző területeken nyújtanának innovatív és kényelmes funkcionalitást, legyen szó videószerkesztésről, zenei lejátszás menedzseléséről vagy egyéb interaktív szolgáltatásokról.

A fejlesztés várhatóan a következő előnyöket kínálná a felhasználói élmény és a biztonság javítása érdekében:

- **Egyszerű használat:** A felhasználóbarát felület és az intuitív funkciók révén a felhasználók könnyedén navigálhatnának és végezhetnék el a szükséges műveleteket anélkül, hogy bonyolult szoftvereket kellene használniuk.
- **Gyors hozzáférés:** Az alkalmazás hatékony letöltési és vágási funkciói lehetővé tennék, hogy a felhasználók időt takarítsanak meg, és azonnal hozzáférjenek a kívánt tartalomhoz.
- **Biztonságos adatkezelés:** A fejlesztés során kiemelt figyelmet fordítanának a biztonságos adatkezelésre, beleértve a felhasználói adatok védelmét és a biztonságos kommunikációs csatornák használatát.
- **Valós idejű frissítések:** A WebSocket technológia segítségével az alkalmazás valós idejű frissítéseket és értesítéseket biztosítana, így a felhasználók mindig naprakészek lehetnének a legújabb változásokkal kapcsolatban.
- **Hibakezelés és megbízható működés:** A fejlesztés során nagy hangsúlyt fektetnének a hibakezelésre és a stabil működésre, hogy az alkalmazás zökkenőmentesen és megbízhatóan üzemelhessen minden körülmények között.

Python alapú lejátszó alkalmazás

A Python-alapú lejátszó alkalmazás egy olyan bővítés lenne, amely megkönnyítené a kezelők munkáját azáltal, hogy a hangfájlokat automatikusan, a megfelelő időben játszaná le. Emellett egy felhasználóbarát felületet biztosítana, ahol egyetlen kattintással frissíteni lehetne a lejátszandó zenét az új nyertes dalra, valamint lehetőség lenne az eredeti alkalmazás egyszerű ki- és bekapcsolására.

A rendszer WebSocket kapcsolaton keresztül kommunikálna a szerverrel, amely lehetővé tenné a valós idejű frissítéseket és vezérlést.

A fejlesztés során a következő technológiákat és könyvtárakat lehetne használni:

- `pygame` a hangfájlok lejátszásához
- `schedule` az ütemezett feladatok kezeléséhez
- `requests` a szerverrel való kommunikációhoz
- `socketio` a WebSocket kapcsolatok kezeléséhez
- `dotenv` a környezeti változók kezeléséhez

Az alkalmazás egy Python szkript formájában valósulna meg, amely folyamatosan futna, figyelve az ütemezett időpontokat és a szerverről érkező frissítéseket. A kezelők könnyedén konfigurálhatnák az alkalmazást a `.env` fájl segítségével, ahol megadhatnák a szerver URL-jét, a WebSocket útvonalát és az API kulcsot.

Az alábbiakban bemutatjuk azokat a végpontokat, amelyeket a Python szkript használna, és amelyek az adminisztrátorok számára lennének elérhetők. Ezek a végpontok lehetővé tennék a hangfájlok frissítését, valamint a lejátszás elindítását és leállítását a szerveren:

Az `/api/server/update` végpont a hangfájl frissítésére szolgálna a szerveren. Amikor egy adminisztrátor GET kérést küldene erre a végpontra, a szerver elindítaná a hangfájl frissítésének folyamatát. Ez magában foglalná a WebSocket átjáróval való kommunikációt annak érdekében, hogy az új hangfájl megfelelően frissüljön és készen álljon a lejátszásra. A végpont válaszként jelezné, hogy a frissítési művelet sikeres volt-e vagy sem.

Az `/api/server/start` végpont a hangfájl lejátszásának elindítására szolgálna a szerveren. Amikor egy adminisztrátor GET kérést küldene erre a végpontra, a szerver elindítaná a hang lejátszását. Ez magában foglalná a megfelelő hangerő beállítását és annak biztosítását, hogy a hangfájl az elejétől kezdve lejátsszák. A végpont válaszként jelezné, hogy a lejátszás sikeresen elindult-e.

Az `/api/server/stop` végpont a hangfájl lejátszásának leállítására szolgálna a szerveren. Amikor egy adminisztrátor GET kérést küldene erre a végpontra, a szerver leállítaná a hang lejátszását. Ez magában foglalná a hangerő elnémítását és a hangfájl lejátszásának megszakítását. A végpont válaszként jelezné, hogy a lejátszás sikeresen leállt-e.

Integrált csengőhang felvevő és vágó

Ez a fejlesztési lehetőség egy olyan alkalmazás, amely lehetővé teszi a felhasználók számára, hogy hangfelvételeket készítsenek és vágjanak meg. Az alkalmazás célja, hogy egyszerű és hatékony módon segítse a felhasználókat a hangfelvételek készítésében és vágásában, anélkül, hogy bonyolult szoftvereket kellene használniuk. Az alkalmazás különösen hasznos lehet azok számára, akik gyorsan és könnyedén szeretnének csengőhangot szerkeszteni. Az egyszerű felhasználói felület és a könnyen használható eszközök révén az alkalmazás mindenki számára elérhetővé teszi a hangfelvételek szerkesztését, függetlenül a technikai tudástól.

A felhasználók által látható felvételi gomb mellett kapna helyet, ahol a felhasználók elindíthatják a hangfelvételt. Az alkalmazás rögzíti a hangot a mikrofonról, majd lehetőséget ad a felhasználóknak a felvétel vágására. A vágás funkció lehetővé teszi a felhasználók számára, hogy kiválasszák a felvétel kezdeti és végpontját. Ezek után az alkalmazás fel fogja tölteni a vágott hanganyagot, az ő nevükben, az elfogadásra váró csengőhangok közé.

A felhasználók számára elérhető feltöltési gomb mellett egy hiperhivatkozás formájában kapna helyet, amelyre kattintva átirányításra kerülnének erre az oldalra.

Keycloak integráció

A Keycloak egy nyílt forráskódú azonosítás- és hozzáféréskezelő rendszer, amely lehetőséget biztosítana nemcsak a felhasználók autentikációjára, hanem az autorizáció kezelésére is. Az integráció során a Keycloak szolgálna az autentikáció és az autorizáció kezelésére a Pollák Csengő rendszerben. Ezáltal a felhasználók biztonságosan és egyszerűen léphetnének be az alkalmazásba, valamint hozzáférhetnének a számukra engedélyezett funkciókhoz és adatokhoz.

Adatbázis

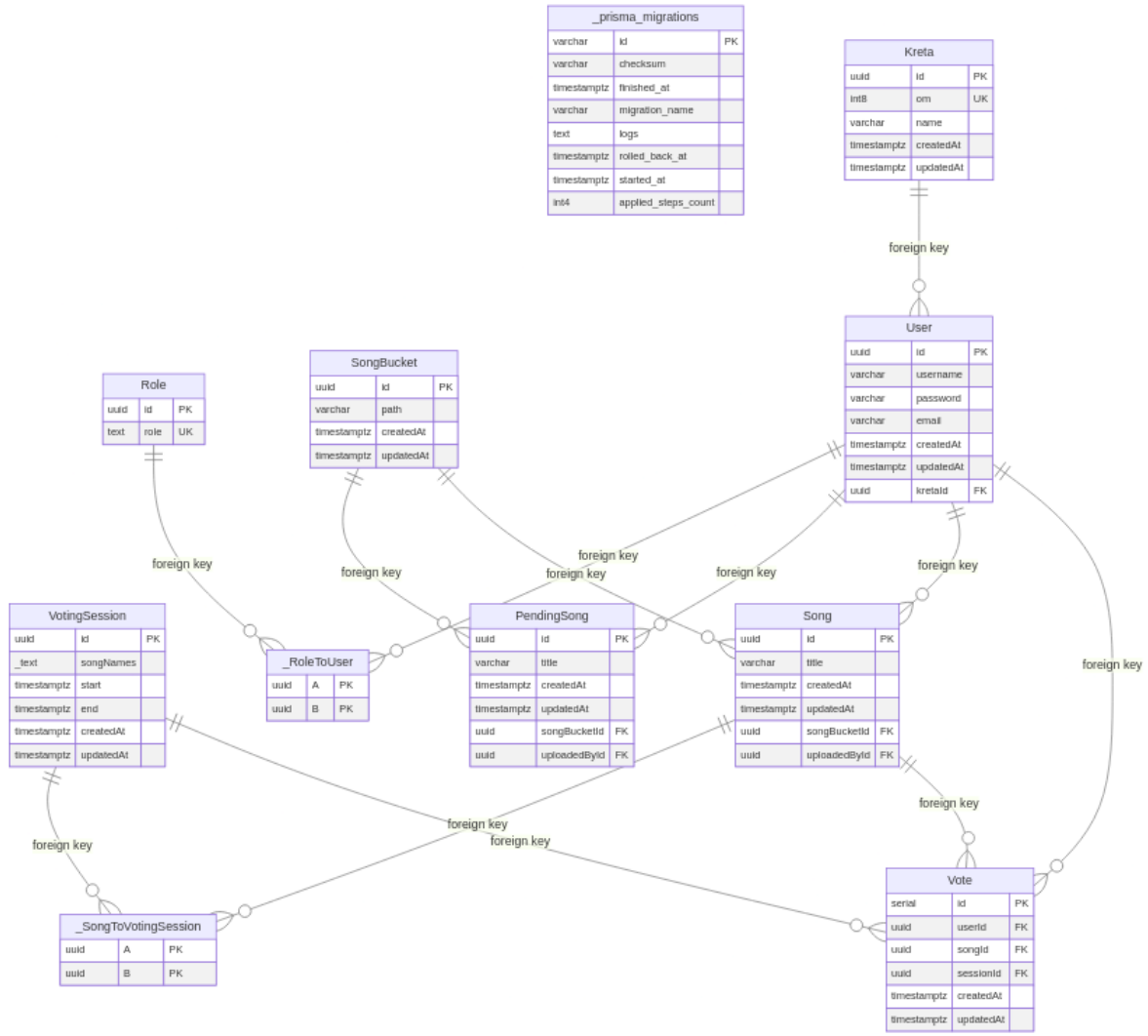
A PostgreSQL-t választottuk más adatbázisok, mint például a MySQL helyett számos előnye miatt. Először is, a PostgreSQL egy rendkívül robusztus és megbízható adatbázis-kezelő rendszer, amely hosszú évek óta bizonyítja megbízhatóságát és stabilitását. Az ACID (Atomicity, Consistency, Isolation, Durability) tulajdonságok támogatása garantálja az adatok integritását és konzisztenciáját.

Továbbá, a PostgreSQL kiválóan támogatja a fejlett adatbázis-funkciókat, mint például a JSON, XML és hstore típusok kezelését, ami rendkívül hasznos a modern webes alkalmazások fejlesztésében. Ezen kívül támogatja a teljes szöveg keresést és az adattárolást komplex adatszerkezetekkel, ami lehetővé teszi az adatok rugalmasabb kezelését és lekérdezését.

A PostgreSQL nyílt forráskódú, így teljesen ingyenesen használható és testreszabható az igényeinknek megfelelően. Az aktív közösségi támogatásnak köszönhetően gyors hibajavításokra és rendszeres frissítésekre számíthatunk. Emellett széles körben kompatibilis különböző operációs rendszerekkel és programozási nyelvekkel, így könnyen integrálható a meglévő rendszereinkkel.

Végül, a PostgreSQL kiváló skálázhatósága és teljesítménye lehetővé teszi az adatok hatékony kezelését nagy terhelés mellett is. Ezek az előnyök mind hozzájárulnak ahhoz, hogy a PostgreSQL ideális választás legyen az adatbázis-kezelési feladatainkhoz.

ER-Diagram



Az ER diagram (Entity-Relationship Diagram) egy grafikus ábrázolás, amely megjeleníti a valós világban létező entitások és azok közötti kapcsolatok kapcsolatát. Ezek a diagramok rendkívül hasznosak az adatbázis-tervezés során, mivel segítenek megérteni és dokumentálni az adatbázis struktúráját.

Az ER diagramok három fő összetevőből állnak: entitásokból, attribútumokból és kapcsolatokból. Az entitások olyan objektumok, amelyek léteznek és egyedileg azonosíthatók, mint például egy "Felhasználó" az alkalmazásodban. Az attribútumok az

entitások tulajdonságait jelentik, például egy "Felhasználó" esetében ez lehet a név, e-mail cím és jelszó. A kapcsolatok az entitások közötti kapcsolatokat jelentik, például egy "Rendelés" entitás kapcsolódhat egy "Felhasználó" entitáshoz.

Az ER diagramok több szempontból is fontosak. Először is, segítenek az adatbázis-tervezőknek megérteni és modellezni az adatbázis struktúráját és logikáját. Emellett lehetővé teszik az adatbázis-tervezők és fejlesztők számára, hogy könnyebben kommunikáljanak az adatbázis szerkezetéről és a kapcsolatok működéséről. Az ER diagramok átlátható és strukturált dokumentációt biztosítanak, amely segíti az adatbázis karbantartását és bővítését. Végül, segítenek azonosítani és megoldani az adatbázissal kapcsolatos problémákat, mivel világosan ábrázolják az összes entitást és kapcsolatot.

Az ER diagramok különböző típusai közé tartozik a koncepcionális, a logikai és a fizikai diagram. A koncepcionális diagram magas szintű ábrázolást nyújt az adatbázis főbb entitásairól és azok kapcsolatairól, és inkább az üzleti logikára összpontosít. A logikai diagram részletesen ábrázolja az entitások és kapcsolatok logikai struktúráját, de nem tartalmazza az adatbázis specifikus részleteit. A fizikai diagram a tényleges adatbázis szerkezetet ábrázolja, beleértve a táblákat, oszlopokat és indexeket.

Az ER diagram készítésének lépései közé tartozik az entitások azonosítása, az entitásokhoz tartozó tulajdonságok meghatározása, az entitások közötti kapcsolatok azonosítása és ábrázolása, valamint az összes entitás, attribútum és kapcsolat grafikus ábrázolása.

Normalizálás

A normalizálás lényege, hogy az adatbázisokat olyan formába hozzuk, amely minimalizálja az adat redundanciáját és maximalizálja az adat integritását. Alapvetően ez az eljárás az adatbázis szerkezetének javítására irányul, hogy elkerüljük a felesleges adatismétléseket és az inkonzisztenciákat.

A saját vizsgaremekünknél a normalizálás nagy feladat volt, mivel sok különböző adatot kellett egy összetett struktúrába szervezni. Kezdetben az adatok több táblázatban voltak szétszórva, ami számos kihívást jelentett a redundancia és az adatintegritás szempontjából.

A normalizálás folyamata során lépésről lépésre haladtunk előre. Először az első normálformát alkalmaztuk, amely az összes táblát úgy alakította át, hogy minden mező atomi legyen, azaz minden mezőben csak egy érték szerepeljen. Ezután a második normálformára léptünk át, amely biztosította, hogy minden nem kulcsmező függjön a teljes elsődleges kulcstól, nem csak annak egy részétől. Végül a harmadik normálformát alkalmaztuk, ahol kiküszöböltük az átmeneti függőségeket is, így minden nem kulcsmező közvetlenül az elsődleges kulcstól függött.

Az eredmény egy tiszta, jól szervezett adatbázis lett, amelyben minimális volt az adat redundancia, és maximális az adatintegritás. A legmagasabb normalizálási szint elérése nemcsak hogy könnyebbé teszi az adatkezelést, hanem javítja a rendszer teljesítményét és megbízhatóságát is.

Táblák

Kreta tábla

A `Kreta` tábla egy olyan entitást reprezentál, amely felhasználókhoz kapcsolódik. Minden rekord egyedi és az `id` mezővel van azonosítva, amely egy alapértelmezettként generált UUID. Az `om` mező egy egyedi BigInt érték, amely valószínűleg jelentős azonosítóként szolgál minden Kreta entitás számára. A `name` mező a Kreta entitás nevét tárolja, legfeljebb 255 karakter hosszúságú szövegként.

A `user` mező egy egy-egy lehetséges kapcsolatot alakít ki a `User` táblával, jelezve, hogy minden Kreta entitás egyetlen felhasználóhoz kapcsolható. A `createdAt` és `updatedAt` mezők időbélyegek, amelyek rögzítik, hogy mikor hozták létre és mikor frissítették utoljára a Kreta entitást. Ezek a mezők alapértelmezés szerint az aktuális időbélyeget kapják a létrehozás időpontjában.

User

A `User` tábla a felhasználókat reprezentálja. Minden felhasználó egyedi és az `id` mezővel van azonosítva, amely egy alapértelmezettként generált UUID. A `username`, `password` és `email` mezők a felhasználónevet, jelszót és e-mail címet tárolják, mindhárom mező legfeljebb 255 karakter hosszúságú szövegként.

A `kreta` mező egy egy-egy kapcsolatot hoz létre a `Kreta` táblával, jelezve, hogy minden felhasználóhoz tartozhat egy Kreta entitás. A `roles` mező a felhasználó szerepeit tárolja, több szereppel való kapcsolattal. A `createdAt` és `updatedAt` mezők időbélyegek,

amelyek rögzítik, hogy mikor hozták létre és mikor frissítették utoljára a felhasználói rekordot. Ezek a mezők alapértelmezés szerint az aktuális időbélyeget kapják a létrehozás időpontjában.

A `Vote` mező a felhasználó által leadott szavazatokat, a `songs` mező a felhasználó által feltöltött dalokat, míg a `pendingSongs` mező az elbírálásra váró dalokat tárolja. A `kretaId` mező az adott felhasználóhoz tartozó Kreta entitás azonosítója.

Song tábla

A `Song` tábla a dalokat reprezentálja. Minden dal egyedi, és az `id` mezővel van azonosítva, amely egy alapértelmezettként generált UUID. A `title` mező a dal címét tárolja, legfeljebb 255 karakter hosszúságú szövegként. A `songBucket` mező egy kapcsolatot hoz létre a `SongBucket` táblával, jelezve, hogy minden dalhoz tartozik egy dal csomag.

A `createdAt` és `updatedAt` mezők időbélyegek, amelyek rögzítik, hogy mikor hozták létre és mikor frissítették utoljára a dalt. Ezek a mezők alapértelmezés szerint az aktuális időbélyeget kapják a létrehozás időpontjában. Az `uploadedBy` mező egy kapcsolatot hoz létre a `User` táblával, jelezve, hogy ki töltötte fel a dalt.

A `songBucketId` és az `uploadedById` mezők az adott dalhoz tartozó dal csomag és a feltöltő felhasználó azonosítói. A `votingSession` és a `Vote` mezők a dalhoz kapcsolódó szavazási üléseket és szavazatokat tárolják.

PendingSong tábla

A `PendingSong` tábla az elbírálásra váró dalokat tartalmazza. Minden elbírálásra váró dal egyedi, és az `id` mezővel van azonosítva, amely egy alapértelmezettként generált UUID. A `title` mező a dal címét tárolja, legfeljebb 255 karakter hosszúságú szövegként. A `songBucket` mező egy kapcsolatot hoz létre a `SongBucket` táblával, jelezve, hogy minden elbírálásra váró dalhoz tartozik egy dal csomag.

A `createdAt` és `updatedAt` mezők időbélyegek, amelyek rögzítik, hogy mikor hozták létre és mikor frissítették utoljára az elbírálásra váró dalt. Ezek a mezők alapértelmezés szerint az aktuális időbélyeget kapják a létrehozás időpontjában. Az `uploadedBy` mező egy kapcsolatot hoz létre a `User` táblával, jelezve, hogy ki töltötte fel az elbírálásra váró dalt.

A songBucketId és az uploadedById mezők az adott elbírálásra váró dalhoz tartozó dal csomag és a feltöltő felhasználó azonosítói.

SongBucket tábla

A SongBucket tábla a dal csomagokat tartalmazza. Minden dal csomag egyedi, és az id mezővel van azonosítva, amely egy alapértelmezettként generált UUID. A path mező a dal csomag útvonalát tárolja, legfeljebb 500 karakter hosszúságú szövegként. A song és a pendingSong mezők kapcsolódnak a Song és a PendingSong táblákhoz, jelezve, hogy minden dal csomaghoz tartozhat egy dal vagy egy elbírálásra váró dal.

A createdAt és updatedAt mezők időbélyegek, amelyek rögzítik, hogy mikor hozták létre és mikor frissítették utoljára a dal csomagot. Ezek a mezők alapértelmezés szerint az aktuális időbélyeget kapják a létrehozás időpontjában.

Vote tábla

A Vote tábla a szavazatokat reprezentálja. Minden szavazat egyedi, és az id mezővel van azonosítva, amely egy automatikusan növekvő egész szám (Int). A userId, songId, és sessionId mezők az adott szavazathoz tartozó felhasználó, dal és szavazási ülés azonosítói, amelyek UUID-ként vannak megadva.

A createdAt és updatedAt mezők időbélyegek, amelyek rögzítik, hogy mikor hozták létre és mikor frissítették utoljára a szavazatot. Ezek a mezők alapértelmezés szerint az aktuális időbélyeget kapják a létrehozás időpontjában.

A Sound, User, és Session mezők kapcsolatok a Song, User, és VotingSession táblákhoz, jelezve, hogy minden szavazat egy dalhoz, egy felhasználóhoz és egy szavazási üléshez tartozik. Ezek a mezők gondoskodnak arról, hogy a szavazatok törlődjenek, ha a kapcsolódó dal, felhasználó vagy szavazási ülés törlésre kerül.

VotingSession tábla

A VotingSession tábla a szavazási üléseket reprezentálja. Minden szavazási ülés egyedi, és az id mezővel van azonosítva, amely egy alapértelmezettként generált UUID. A songs mező a szavazási üléshez kapcsolódó dalokat tárolja, míg a songNames mező a dalok neveit tartalmazza szövegként.

A start és end mezők az adott szavazási ülés kezdetét és végét rögzítik, időbélyegekkel. A createdAt és updatedAt mezők időbélyegek, amelyek rögzítik, hogy mikor hozták

létre és mikor frissítették utoljára a szavazási ülést. Ezek a mezők alapértelmezés szerint az aktuális időbélyeget kapják a létrehozás időpontjában.

A `Vote` mező kapcsolódik a `Vote` táblához, jelezve, hogy minden szavazási üléshez tartoznak szavazatok.

Role tábla

A `Role` tábla a felhasználói szerepeket tárolja. Minden szerep egyedi, és az `id` mezővel van azonosítva, amely egy alapértelmezettként generált UUID. A `role` mező egy egyedi szöveges érték, amely magát a szerepkört jelöli.

A `users` mező kapcsolódik a `User` táblához, jelezve, hogy minden szerephez tartozhatnak felhasználók.

Kapcsolótáblák

_SongToVotingSession tábla

A `_SongToVotingSession` tábla a `Song` és a `VotingSession` táblák közötti kapcsolatot reprezentálja. Minden rekord egyedi és az `A` és `B` mezőkkel van azonosítva, amelyek UUID-ként vannak megadva. Az `A` mező a `Song` tábla azonosítóját, míg a `B` mező a `VotingSession` tábla azonosítóját tárolja. Ez a tábla biztosítja, hogy egy dal több szavazási üléshez is kapcsolódhasson, és egy szavazási ülés több dalt is tartalmazhasson.

_RoleToUser tábla

A `_RoleToUser` tábla a `Role` és a `User` táblák közötti kapcsolatot reprezentálja. Minden rekord egyedi és az `A` és `B` mezőkkel van azonosítva, amelyek UUID-ként vannak megadva. Az `A` mező a `Role` tábla azonosítóját, míg a `B` mező a `User` tábla azonosítóját tárolja. Ez a tábla biztosítja, hogy egy felhasználó több szerepkörrel is rendelkezhet, és egy szerepkör több felhasználóhoz is kapcsolódhat.

_prisma_migrations tábla

A `_prisma_migrations` tábla a Prisma migrációkat tárolja. Minden rekord egyedi és az `id` mezővel van azonosítva, amely egy varchar típusú érték. A `checksum` mező a migráció ellenőrző összegét tárolja, míg a `finished_at`, `rolled_back_at`, és `started_at` mezők időbélyegek, amelyek rögzítik, hogy mikor fejeződött be, mikor lett visszavonva, és mikor kezdődött el a migráció. A `migration_name` mező a migráció nevét tárolja, a `logs` mező pedig a migráció naplóit tartalmazza. Az `applied_steps_count` mező egy egész szám, amely a végrehajtott lépések számát tárolja.

Felhasználói dokumentáció

Rövid bemutatás

Üdvözölöm!

A **Pollák Csengő** webalapú alkalmazást tartja a kezében, amely lehetőséget biztosít az iskolák számára online szavazások lebonyolítására. A diákok ezen szavazások segítségével hozzájárulhatnak egy kényelmesebb és vonzóbb iskola környezet kialakításához, folyamatosan változó csengőhangok választásával.

A szoftver alapvető funkciói között szerepel a belépés és regisztráció lehetősége. Ha még nem regisztrált, egyszerűen létrehozhatja a fiókját, majd bejelentkezhet az oldalra, hogy hozzáférést kapjon a funkciókhoz.

A weboldalon történő belépést követően lehetősége van kedvenc zenéinek feltöltésére, amelyeket a weboldal kezelője bírál el, és szükség esetén bevon a szavazásba. Ezen kívül megtekintheti az előző szavazás nyertes zenéjét, valamint amennyiben aktív szavazás folyik, lehetősége van a zenéket egyesével meghallgatni, és szavazni az Ön által kedvelt darabokra.

Amennyiben Admin jogosultsággal rendelkező fiókkal rendelkezik, a felhasználói ikonra kattintva megjelenik az „Admin” gomb, amely segítségével elérheti a kezelőpanelt. A kezelőpanel oldalán lehetősége nyílik az összes elfogadott zene megtekintésére és meghallgatására, valamint az elfogadásra váró zenék megtekintésére és meghallgatására. Ezen kívül lehetősége van a szavazások és a felhasználók kezelésére is.

Az alábbiakban megismerheti a szoftver működési elveit, melyeket szemléltető képekkel is illusztráltunk. Minden bekezdés végén található képes illusztráció is az egyszerűbb bemutatás érdekében, ezeket helyből a „**(☆)**” megnyomásával is elérheti.

Ha bármilyen kérdése van az alkalmazással kapcsolatban, forduljon hozzánk bizalommal az alábbi e-mail címen: janipatrik138@gmail.com.

Köszönjük, hogy a Pollák Csengőt választotta! Élvezze a zenéket és hozza létre együtt a diákokkal tökéletes iskola hangulatot!

Üdvözlettel, Pollák Csengő csapata.

Harver szükséglet átlagos felhasználónak

A Pollák Csengő webalkamazás viszonylag mérsékelt hardverigényekkel rendelkezik amely hasonló a napjainkba használt modern oldalkhoz mint például a Facebook, de a jobb felhasználói élmény érdekében az alábbi minimum és ajánlott konfigurációk figyelembevételével kell biztosítani a megfelelő eszközöket:

Processzor (CPU):

- Minimum: 1.6 GHz-es kétmagos processzor (Intel Core i3, AMD Ryzen 3)
- Ajánlott: 2.5 GHz vagy gyorsabb négymagos processzor (Intel Core i5 / i7, AMD Ryzen 5)

Memória (RAM):

- Minimum: 4 GB RAM
- Ajánlott: 8 GB RAM vagy több

Videokártya (GPU):

- Minimum: Integrált videokártya (Intel HD Graphics, AMD Vega)
- Ajánlott: Dedikált videokártya (NVIDIA GeForce GTX Széria, AMD Radeon RX Széria)

Tárhely:

- Minimum: 20 GB szabad hely a rendszer és az alkalmazások számára
- Ajánlott: SSD meghajtó, 256 GB vagy több

Képernyőfelbontás:

- Minimum: 600x600
- Ajánlott: 1920x1080 (Full HD) vagy magasabb

Szoftver szükséglet átlagos felhasználónak

A Pollák Csengő webalkalmazás optimális működéséhez a következő szoftverek és technológiai környezetek szükségesek:

Webböngészők és Verziók:

- Google Chrome: Minimum 85-ös verzió, ajánlott 110+ verzió
- Mozilla Firefox: Minimum 80+ verzió, ajánlott 100+ verzió
- Microsoft Edge: Minimum 85-ös verzió, ajánlott 110+ verzió
- Safari: Minimum 13-as verzió, ajánlott 14+ verzió

A böngészők legfrissebb verzióinak használatát javasoljuk, mivel azok biztosítják a legújabb biztonsági frissítéseket és a legjobb teljesítményt.

Operációs rendszerek:

- Windows: Minimum 10-es verzió, ajánlott 11-es verzió
- macOS: Minimum Mojave (10.14), ajánlott Monterey (12.0) vagy újabb
- Linux: Ubuntu 20.04 vagy újabb verzió

JavaScript, TypeScript és HTML5 Támogatás:

A Pollák Csengő alkalmazás a Vue.js, JavaScript/TypeScript és HTML5 technológiákat használ a dinamikus funkciók és interaktív elemek biztosításához. A böngészőknek támogatniuk kell a JavaScript/TypeScript futtatását, valamint az HTML5 szabványokat a helyes működéshez.

Biztonság és Adatvédelem:

A Pollák Csengő alkalmazás biztonságos használata érdekében javasoljuk a következő biztonsági intézkedések betartását:

- HTTPS: Az alkalmazás minden adatátvitele titkosított HTTPS kapcsolaton keresztül történik. Győződjön meg arról, hogy böngészője támogatja az SSL/TLS titkosítást.
- Cookie-k: Az alkalmazás használatának részeként a böngészőnek támogatnia kell a cookie-kat, hogy a felhasználói élmény zökkenőmentes legyen (pl. a bejelentkezett állapot tárolása).

Specifikus hardver szükséglet a futtatáshoz

A Pollák Csengő webalkalmazás futtatásához a következő specifikus hardverek és beállítások ajánlottak:

Számítógép vagy Laptop es estén (Ajánlott):

- **Time Zone**: Central Europe Standard Time
- **Installed Physical Memory (RAM)**: 8 GB
- **Hyper-V - VM Monitor Mode Extensions**: Yes

Internetkapcsolat (Ajánlott):

- 10 Mbps vagy gyorsabb

Bizonyosodjon meg róla, hogy a szükséges hardverek és eszközök rendelkezésre állnak a Pollák Csengő alkalmazás futtatásához. Bármely más hardver vagy eszköz használata a Pollák Csengő alkalmazás futtatása során nem garantált, és problémákhoz vezethetnek.

Specifikus szoftver szükséglet a futtatáshoz

A Pollák Csengő alkalmazás futtatásához külöféle szoftverek és technológiai környezetek lehetnek szükségesek a futtatás módjától függően. Bizonyosodjon meg róla, hogy a szükséges szoftverek és verziók telepítve vannak a rendszerén a futtatás módjának szükségletében, mielőtt elkezdené a Pollák Csengő alkalmazás futtatását. Bármely más szoftver verzió használatával a Pollák Csengő alkalmazás futtatása nem garantált, és problémákhoz vezethet.

Letöltés, Telepítés és Elindítás

Letöltés

A Pollák Csengő webalkalmazás letölthető az alábbi módokon a GitHub repository-ból:

- **A következő linken:** Pollák Csengő GitHub Repository
- **A következő git parancs futtatásával a mappában, ahova a projektet szeretné letölteni:** *git clone https://github.com/Pollak-Projects/csengo-ts.git*

A szoftver **három** különböző módon indítható el:

Telepítés, Elindítás

- **Docker segítségével:** Ez a legegyszerűbb és leggyorsabb indítási mód, mivel csupán egy Docker Desktop telepítése szükséges a működéséhez.
- **Docker nélkül:** Ebben az esetben elkerülhető a Docker használata, azonban hosszabb telepítési folyamatra és több szoftver csomag telepítésére van szükség.
- **Fejlesztői módban való indítás:** Ezt az indítási módot a fejlesztők számára alakították ki, hogy egyszerűbben tudják módosítani a szoftvert. A pontos lépések és beállítások leírását a fejlesztői dokumentáció tartalmazza.

Elindítás Docker segítségével

Futtatási szoftver követelmények:

- **Windows:** Docker Desktop
- **Linux, MacOS:** Docker Desktop, Docker-Compose

A követelmények telepitése után a következő lépésekkel tudja elinditani a szoftvert:

- **Módszer 1:** Nyissa meg a `csengo-ts` mappát és indítsa el a **`run.bat`** fájlt.
- **Módszer 2:** Nyisson meg egy parancssort git repository törzs mappájában, ahol a `docker-compose.dev.v2.yml` nevű fájl található majd futtasa az alábbi parancsot: *docker-compose -f docker-compose.dev.v2.yml up -d*

Ezekkel a lépésekkel sikeresen futtathatja a Pollák Csengő alkalmazást a Docker segítségével.

A következő URL-eken hozzáférhet a Pollák Csengő szolgáltatásaihoz:

- pgAdmin adatbázis kezelő: http://localhost:8081
- Csengő szerver: http://localhost:3300
- Csengő weboldal: http://localhost:8080

Az alkalmazás fejlesztése/tesztelése után, a `run_cleanup.bat` fájl segítségével törölheti a generált Docker `container`-eket, `network`-öket, `image`-ket és `volume`-okat.

Elindítás Docker segítsége nélkül

Futtatási szoftver követelmények

- **Microsoft Windows 11 Pro 10.0.26100 Build 26100**
- **Node.js v22.12.0**
- **npm 11.0.0**

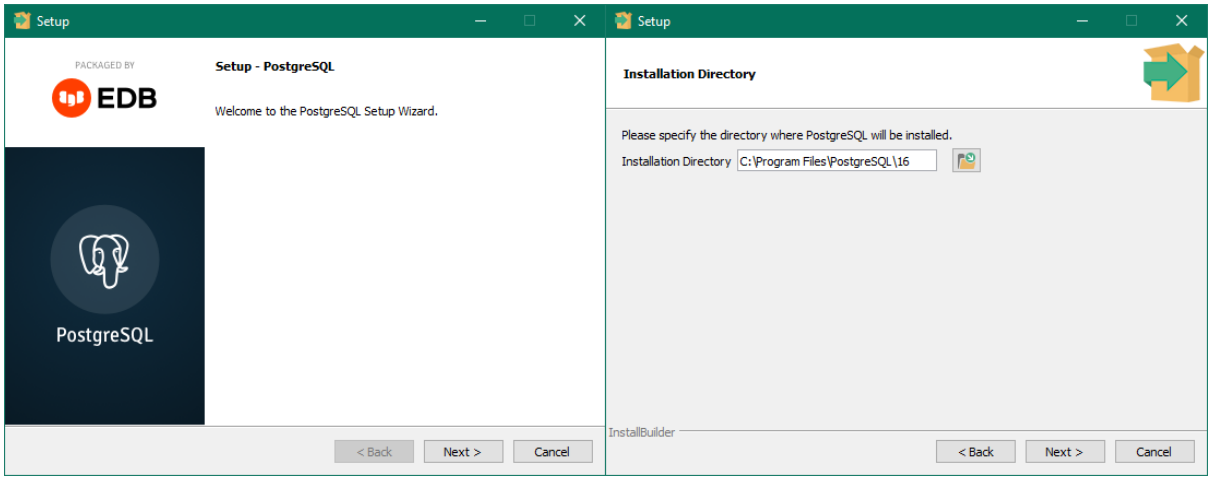
Fontos: A `run_dev.bat` fájl, minden lefutásnál, vissza fogja állítani az adatbázist a `./assets/initdb_bat_local.sql` tartalmára, amennyiben ez nem kívánatos, futtassa manuálisan az alkalmazást, melynek lépéseit a fejlesztői dokumentációban találhatja.

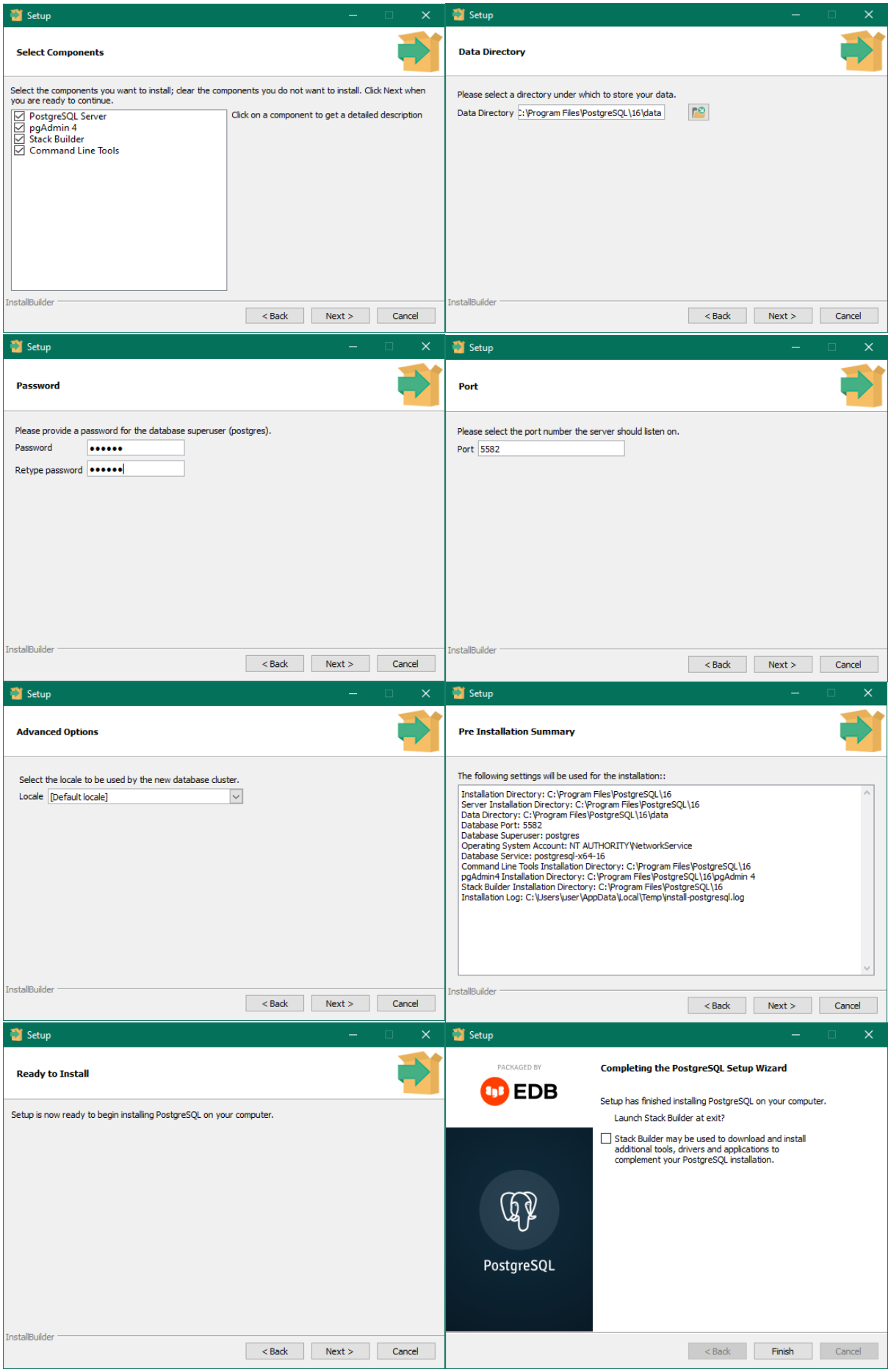
A követelmények telepitése után a következő lépésekkel tudja elinditani a szoftvert:

1. Nyissa meg a `csengo-ts` mappát.
2. Futtassa a **`run_dev.bat`** nevű batch file-t a szerver és a klines elindításához.
3. Töltse ki az alábbi adatokkal a megjelenő Postgres telepítőt:

- Location: `C:\Program Files\PostgreSQL\16`
- Port: `5582`
- Password: `csengo`
- Locale: `Default locale`
- Az utolsó ablakban kapcsolja ki a `Stack Builder` opciót.

Az alábbi képeken láthatóak a Postgres telepítő ablakai, amelyeket a fenti adatokkal kell kitölteni:




Setup
Select Components
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
PostgreSQL Server
pgAdmin 4
Stack Builder
Command Line Tools
Click on a component to get a detailed description
InstallBuilder
< Back
Next >
Cancel
Setup
Data Directory
Please select a directory under which to store your data.
Data Directory C:\Program Files\PostgreSQL\16\data
InstallBuilder
< Back
Next >
Cancel
Setup
Password
Please provide a password for the database superuser (postgres).
Password
Retype password
InstallBuilder
< Back
Next >
Cancel
Setup
Port
Please select the port number the server should listen on.
Port 5582
InstallBuilder
< Back
Next >
Cancel
Setup
Advanced Options
Select the locale to be used by the new database cluster.
Locale [Default locale]
InstallBuilder
< Back
Next >
Cancel
Setup
Pre Installation Summary
The following settings will be used for the installation::
Installation Directory: C:\Program Files\PostgreSQL\16
Server Installation Directory: C:\Program Files\PostgreSQL\16
Data Directory: C:\Program Files\PostgreSQL\16\data
Database Port: 5582
Database Superuser: postgres
Operating System Account: NT AUTHORITY\NetworkService
Database Service: postgresql-x64-16
Command Line Tools Installation Directory: C:\Program Files\PostgreSQL\16
pgAdmin4 Installation Directory: C:\Program Files\PostgreSQL\16\pgAdmin 4
Stack Builder Installation Directory: C:\Program Files\PostgreSQL\16
Installation Log: C:\Users\user\AppData\Local\Temp\install-postgresql.log
InstallBuilder
< Back
Next >
Cancel
Setup
Ready to Install
Setup is now ready to begin installing PostgreSQL on your computer.
InstallBuilder
< Back
Next >
Cancel
Setup
PACKAGED BY
EDB
PostgreSQL
Completing the PostgreSQL Setup Wizard
Setup has finished installing PostgreSQL on your computer.
Launch Stack Builder at exit?
Stack Builder may be used to download and install additional tools, drivers and applications to complement your PostgreSQL installation.
< Back
Finish
Cancel

Amennyiben a batch fájl nem tudja letölteni a PostgreSQL telepítőt, a következő lépéseket kell követni:

1. Erről a weboldalról töltse le a PostgreSQL 16.8-as verzióját: https://www.enterprisedb.com/downloads/postgres-postgresql-downloads
2. Nevezze át a letöltött fájlt `postgresql-16.8-1-windows-x64.exe`-re.
3. Majd helyezze az átnevezett fájlt a `csengo-ts` mappába.
4. Futtassa a `run_dev.bat` fájlt újra.

Amennyiben a batch fálj nem tud csatlakozni az adatbázishoz, a következő lépéseket kell követni:

1. Ha nem sikerült a PostgreSQL telepítése, a `run_dev.bat` fájlal, telepítse azt manuális módon a `postgresql-16.8-1-windows-x64.exe` fájl segítségével, a fentebb lévő adatokkal.
2. Nyissa meg a `pgAdmin 4` nevű alkalmazást, majd a már meglévő adatbázis kapcsolatra kattintva, nyisson egy `psql` konzolt, mely az imént telepített adatbázishoz kapcsolódik.
3. Másolja be a `run_dev.bat` fájl által generált `csengo-ts/assets/initdb_bat_local.sql` fájl tartalmát a `psql` konzolba.

Miután a PostgreSQL telepítése sikeresen befejeződött, a `run_dev.bat` fájl automatikusan elindítja a szerver és a kliens alkalmazást.

Az alkalmazás fejlesztése/tesztelése után, a `run_dev_cleanup.bat` fájl segítségével törölheti PostgreSQL alkalmazást és az adatbázist. Ehhez kövesse a Batch fájl futtatásakor látható utasításokat.

A Pollák Csengő alkalmazás futtatása után a következő URL-eken érhető el:

- Csengő szerver: http://localhost:3300
- Csengő weboldal: http://localhost:3000

A következő felhasználói adatokkal tud belépni a Csengő weboldalra a tesztelés elvégzéséhez:

- Admin jogosultságú felhasználó:
  - Felhasználónév: `admin`
  - Jelszó: `admin`
- Felhasználó jogosultságú felhasználók:
  - Tamáskovits Gyula Ákos nevű felhasználó:
    - Felhasználónév: `Tamaskovits`
    - Jelszó: `tamaskovits`
  - Jani Patrik nevű felhasználó:
    - Felhasználónév: `Janipatrik`
    - Jelszó: `janipatrik`
  - Barna Máté nevű felhasználó:
    - Felhasználónév: `Barnamate`
    - Jelszó: `barnamate`

A regisztráció teszteléséhez a következő adatokat tudja használni:

- Felhasználónév: `teszt`
- Email cím: `teszt@csengo.dev`
- OM azonosító: `74444444444`
- Jelszó: `teszt123456`

Ismertetés

Az alábbi részleg az alkalmazás részletes bemutatását tartalmazza. Ismerteti annak felépítését, főbb funkcióit és azok szerepét. Minden bekezdés végén találhat illusztrációs képeket amelyek szemléltetik és bemutatják a leírtakat. **Ezeket a dokumentáció bármely pontjáról elérheti hiperhivatkozással a „☆" emojira kattintva.**

A webalkalmazás lebontása:

- **Toast** [Értesítési ablak]
- **Regisztráció** [Oldal]
- **Bejelentkezés** [Oldal]
- **Kezdőlap** [Oldal]
- **Vágó** [Oldal]
- **Tv** [Oldal]
- **Admin** [Oldal]
    - **Zenék** [Menüpont]
    - **Elfogadásra** váró zenék [Menüpont]
    - **Felhasználók** [Menüpont]
    - **Szavazások** [Menüpont]
    - **Egyebek** [Menüpont]

A webalkalmazás szerver oldali komponenseinek lebontása:

- **Tv** [Oldal]
- **Pending Songs** [Oldal]

Toast értesítési ablak

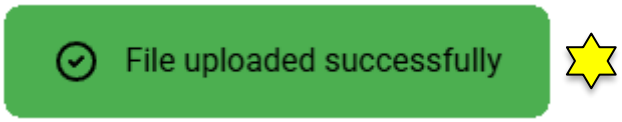
A Toast menü a háttérben zajló eseményekről tájékoztatja a felhasználót. Minden értesítés a képernyő jobb alsó sarkában jelenik meg. A színek a következő módon jelzik az értesítés típusát:

- **Zöld (☆)**: Sikeres művelet, amely megerősíti, hogy a felhasználó által végrehajtott művelet sikeresen befejeződött.
- **Sárga (☆)**: Figyelmeztetés, amely jelzi, hogy bár a művelet végrehajtása megtörtént, valamilyen probléma merült fel.
- **Piros (☆)**: Hiba, amely arra utal, hogy valamilyen probléma történt, és a művelet nem sikerült.

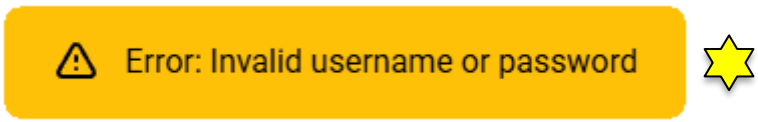
Minden értesítés tartalmazza a pontos üzenetet, amely leírja a történt eseményt vagy hibát. A felugró menü mindig 3 másopercig látható, azután automatikusan eltűnik.

Illusztrációs képek:

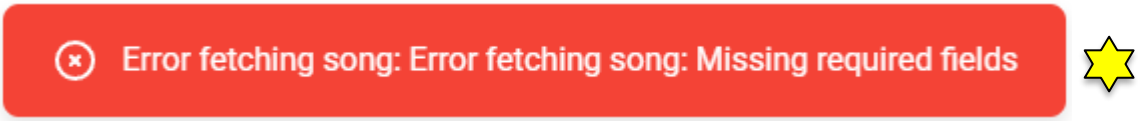
Sikeres művelet esetén megjelenő Toast ablak.



Figyelmeztetés esetén megjelenő Toast ablak.



Sikertelen művelet esetén megjelenő Toast ablak.



Regisztrációs oldal

A regisztrációs oldal biztosítja a gyors és egyszerű regisztrációs folyamatot. Az oldal a következő útvonalon érhető el: **[/register]**.

Az oldal megnyitásakor a felhasználók egy központi elrendezésű felületet látnak, amely öt beviteli mezőt és egy gombot tartalmaz. A beviteli mezők a következő adatokat és kikötéseket kérik a felhasználótól:

- Felhasználónév - ☆
    - Nem lehet üres
    - Minimum 5 karakter hosszú
    - Maximum 30 karakter hosszú
- **Email cím -** ☆
    - Nem lehet üres
    - Csak érvényes email cím lehet
- **OM azonosító -** ☆
    - Nem lehet üres
    - Csak szám lehet
    - Csak érvényes, adatbázisban feljegyett azonosító lehet
- **Jelszó -** ☆
    - Nem lehet üres
    - Minimum 6 karakter hosszú
    - Maximum 20 karakter hosszú
- Jelszó megerősítése - ☆
    - Egyezzen meg a Jelszó mezőbe beírt jelszóval

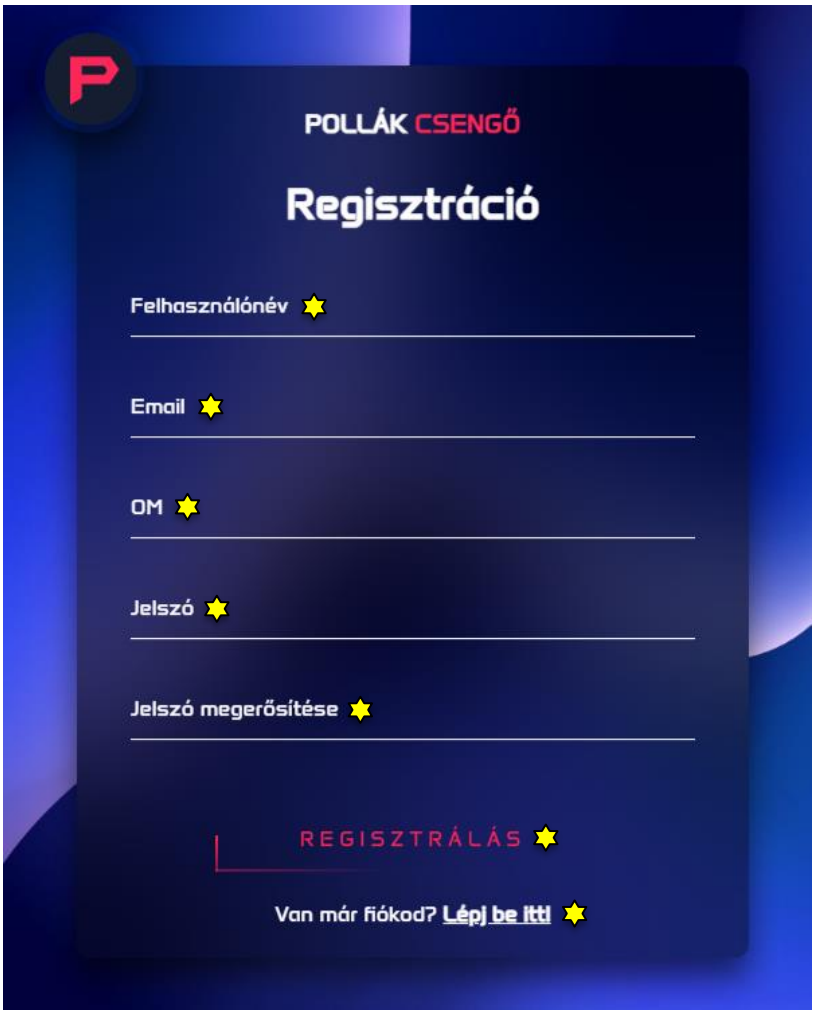
Minden mező felett látható a szükséges adat megnevezése.

A regisztrációs űrlap kitöltését követően, ha a felhasználó rákattint a **Regisztrálás gombra** (☆), sikeres regisztráció esetén a felhasználó átirányításra kerül a főoldalra. Ha a regisztráció nem sikerül, egy hibaüzenet jelenik meg egy toast értesítés formájában a képernyő jobb alsó sarkában, amely tartalmazza a hiba okát.

A gomb alatt látható egy hiperhivatkozás is amely biztosítja az egyerű átlépést a bejelentkezési oldalra. Ezt a "**Lépj be itt!**" (☆) szövegre kattintva tudja a felhasználó elérni.

Regisztrált fiók nélkül a felhasználó nemtudja elérni az oldal többi funkcionalitását, csakis a bejelentkezési és regisztrációs felületeteket.

Illusztrációs kép:



Bejelentkezési oldal

A bejelentkezési oldal biztosítja a gyors és egyszerű hozzáférést az oldal többi funkciójához. Az oldal a következő útvonalon érhető el: **[/login].**

Az oldal megnyitásakor a felhasználók egy központi elrendezésű felületet látnak, amely kettő beviteli mezőt és egy gombot tartalmaz. A beviteli mezők a következő adatokat kérik a felhasználótól:

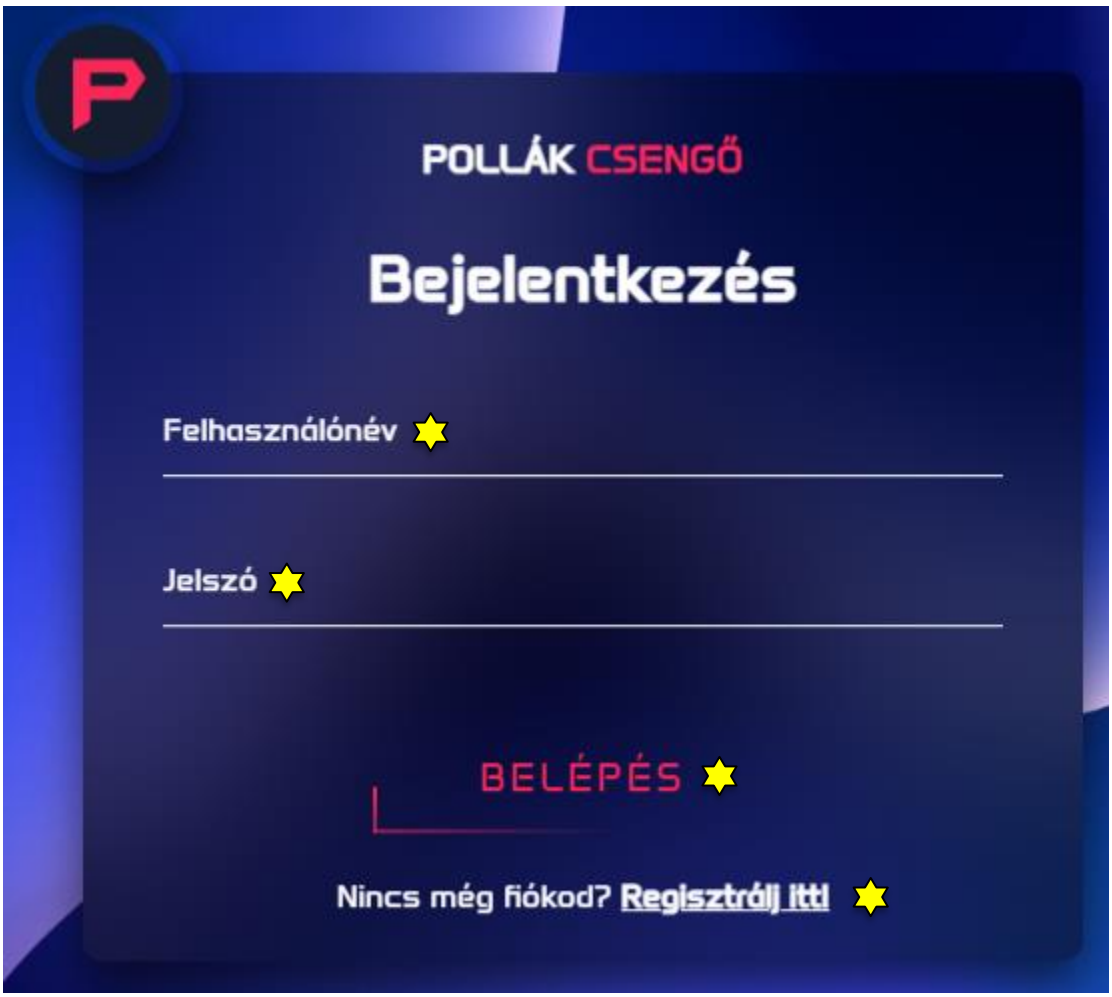
- **Felhasználónév -** ☆
  - Nem lehet üres
  - Minimum 5 karakter hosszú
  - Maximum 30 karakter hosszú
- **Jelszó -** ☆
  - Nem lehet üres
  - Minimum 6 karakter hosszú
  - Maximum 20 karakter hosszú

Minden mező felett látható a szükséges adat megnevezése.

A bejelentkezési űrlap kitöltését követően, ha a felhasználó rákattint a **Belépés gombra** (☆), sikeres bejelentkezés esetén a felhasználó átirányításra kerül a főoldalra. Ha a bejelentkezés nem sikerül, egy hibaüzenet jelenik meg egy toast értesítés formájában a képernyő jobb alsó sarkában, amely tartalmazza a hiba okát.

A gomb alatt látható egy hiperhivatkozás is amely biztosítja az egyerű átlépést a regisztrációs oldalra. Ezt a **"Lépj be itt!"** (☆) szövegre kattintva tudja a felhasználó elérni.

Illusztrációs kép:



Kezdőlap

A kezdőlap biztosítja a kezelőfelületet a felhasználók számára ahol zenéket tölthetnek fel, szavazatnak és megtekinthetik az előző győztes zenét. Az oldal a következő útvonalon érhető el: **[/]**.

A kezdőlap megnyitásakor a felhasználó az alábbi elemeket látja:

A **bal felső** sarokban egy **köszöntő üzenet** (☆) jelenik meg a felhasználó valódi nevével, amelyet az adatbázisból kap meg az OM azonosító alapján.

A **jobb felső** sarokban található egy **Felhasználói ikon** (☆). Erre kattintva két lehetőség jelenik meg:

- **Admin** (☆): Ez a gomb csak akkor látható, ha a felhasználó admin ranggal rendelkezik. Rákattintva az admin felületre navigálhat.
- **Kijelentkezés** (☆): Ez a gomb mindig elérhető, és segítségével a felhasználó kijelentkezhet az oldalról.

A **képernyő középső részén két kiemelt felület** található, amelyek a következőket tartalmazzák:

Bal oldali felület:

- **Zene feltöltés / Vágás** (☆)**:** A felhasználó ide töltheti fel a zenéjét, a fájl típusának és hosszának követelményeivel. A **feltöltéshez szükséges** (☆) és az **átirányításra szolgáló gomb** (☆) is itt található, és a zenét az adminok később jóváhagyhatják.
- **Szavazatok** (☆)**:** Itt jelennek meg a jelenlegi szavazásban részt vevő zenék. Az első három zene és a rájuk leadott szavazatok száma látható. Ha jelenleg nincs szavazás, akkor egy **„Jelenleg nincsen szavazás”** (☆) üzenet jelenik meg.

Jobb oldali felület:

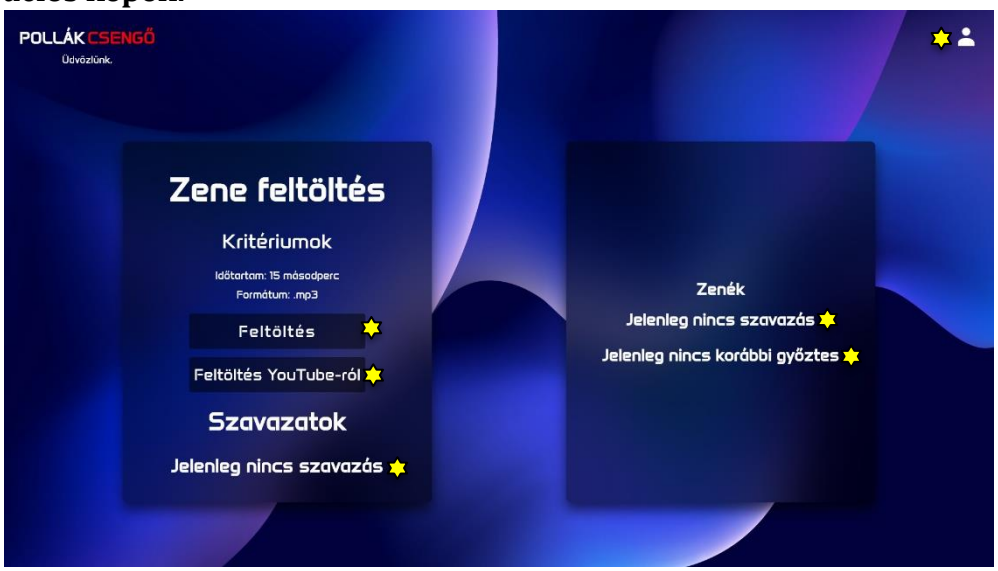
- **Zenék** (☆)**:** A zenék listája, amelyet görgethető formában lehet megtekinteni. Minden zenére egy színnel elválasztott, kerekített téglalap formájú elem van, amely a **következőket tartalmazza:**
    - **Bal oldalon** egy **lejátszás/megállítás ikon** (☆) található, amely a zene lejátszását vagy megállítását szolgálja.

- Középen a zene címe (☆).
- **Jobb oldalon** egy **like ikon** (☆)**,** amely üres vagy teli lehet. Ha üres, a felhasználó még nem szavazott a zenére, ha teli, akkor már leadta a szavazatát. A felhasználó bármennyi zenére szavazhat, és szavazatát visszavonhatja.
- Ha jelenleg nincs szavazás, akkor a **„Jelenleg nincsen szavazás”** (☆) felirat jelenik meg.

- Előző győztes (☆): Itt található az előző szavazás győztesének címe. Ha még nem volt győztes, akkor „Jelenleg nincs korábbi győztes” (☆) felirat jelenik meg.

A felosztás monitoros és telefonos használat során változik. Nagy kijelzőn a kettő fő elem egymás mellett helyezkedik el, azonban kicsi, telefonos kijelzőn egymás alatt az egyszerűbb használat érdekében.

Illusztrációs képek:



Vágó oldal

A Vágó oldal biztosítja a kezelőfelületet a felhasználók számára ahol zenéket vághatnak meg és tölthetnek fel külső eszközök és szoftverek nélkül egy egyszerű YouTube link használatával. Az oldal a következő útvonalon érhető el: **[/snipper]**.

A Vágó oldal megnyitásakor a felhasználó az alábbi elemeket látja:

A **bal felső** sarokban egy **Otthon ikon** (☆) jelenik meg amely megnyomásával a felhasználó vissza navigálhat a kezdőlapra.

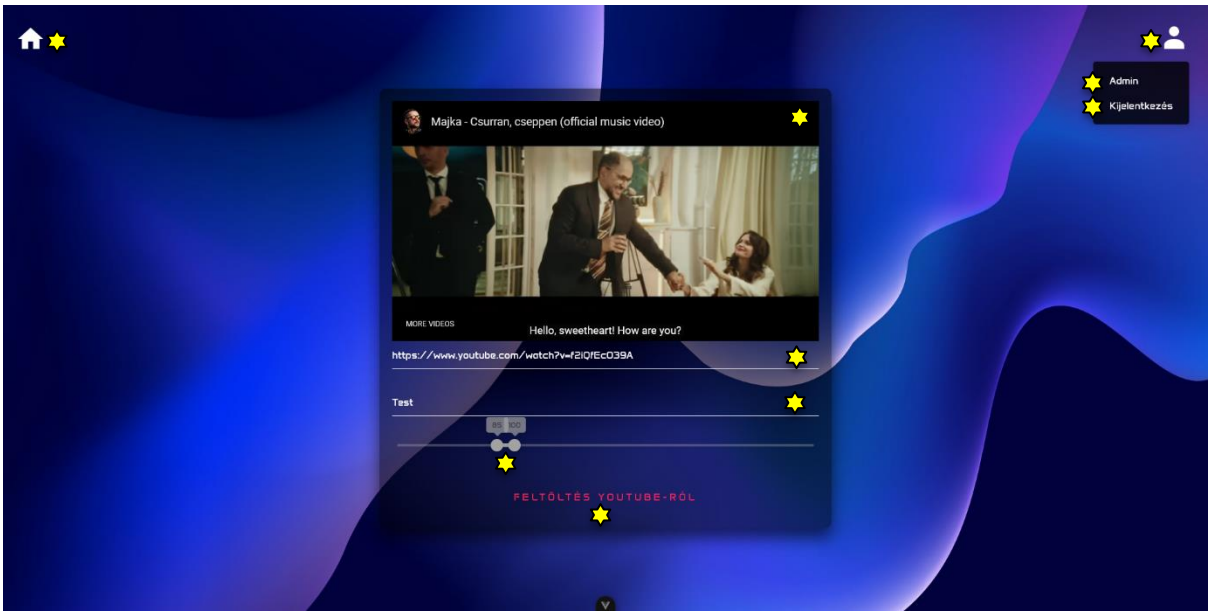
A **jobb felső** sarokban található egy **Felhasználói ikon** (☆). Erre kattintva két lehetőség jelenik meg:

- **Admin** (☆): Ez a gomb csak akkor látható, ha a felhasználó admin ranggal rendelkezik. Rákattintva az admin felületre navigálhat.
- **Kijelentkezés** (☆): Ez a gomb mindig elérhető, és segítségével a felhasználó kijelentkezhet az oldalról.

A **képernyő középső részén egy kiemelt felület** található, amely a következőket tartalmazza:

- **YouTube Lejátszó (☆):** Ez a panel felelős a YouTube-videók lejátszásáért a megadott hivatkozás alapján.
- **Link Beviteli Mező (☆):** Ebben a mezőben adható meg a YouTube-videó hivatkozása. Az itt megadott link alapján jelenik meg és játszódik le a videó a YouTube lejátszóban.
- **Cím Beviteli Mező (☆):** Ez a mező szolgál a feltöltendő zene címének megadására.
- **Időtartomány Csúszka (☆):** A csúszka segítségével egy 5–15 másodperces időintervallum adható meg, amely a videó kiválasztott részletét határozza meg a feltöltéshez.
- **Feltöltés Gomb (☆):** A gomb megnyomásával a kiválasztott videórészlet konvertálása és feltöltése történik az elfogadásra váró zenék közé.

Illusztrációs képek:



TV oldal

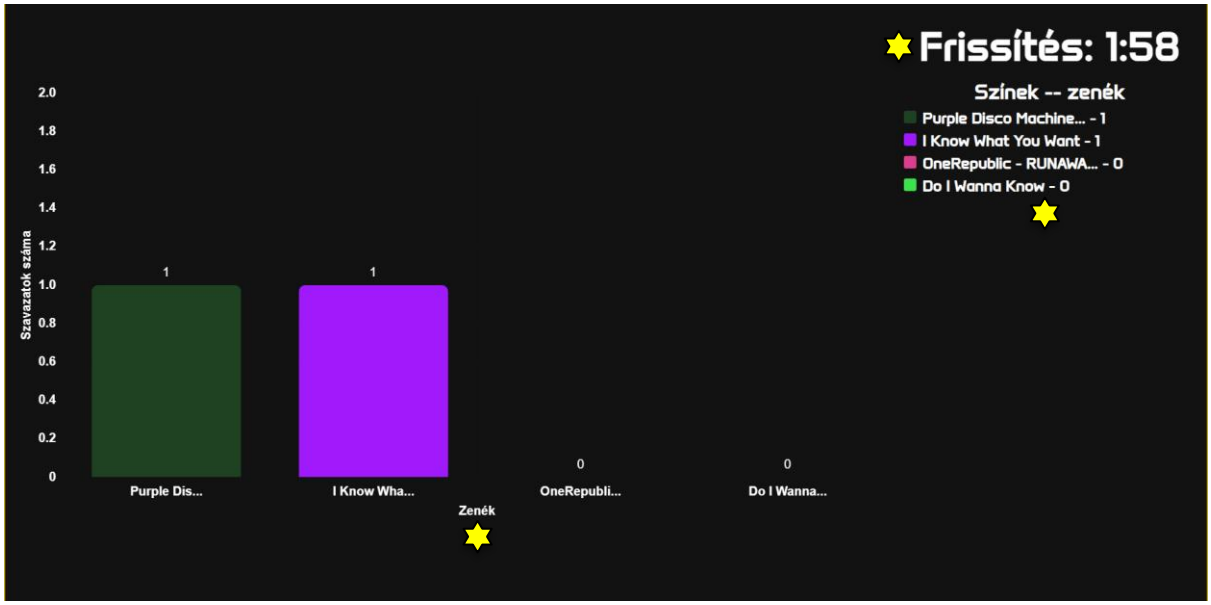
A TV oldal egy felületet biztosít azoknak az iskoláknak, amelyek használják a Pollák Csengő rendszert, lehetőséget adva a szavazás aktuális állásának megjelenítésére. Az oldal csak kijelentkezve és a következő útvonalon érhető el: **[/tv]**.

Az oldal három fő elemet tartalmaz:

- **Grafikon** (☆)**:** Dinamikusan generált vizuális elem, amely a jelenlegi szavazásban részt vevő zenékre leadott szavazatok számát ábrázolja.
- **Lista** (☆)**:** Az adatok egyszerűbb és átláthatóbb megjelenítését biztosítja. Tartalmazza a grafikon oszlopainak megfelelő színeket, a zenék címét, valamint a hozzájuk tartozó szavazatok számát.
- **Visszaszámláló** (☆)**:** A lista felett elhelyezkedő időzítő, amely a beállított időintervallum lejártakor automatikusan frissíti a grafikont és a listát. Ezen időzítő alapállás szerint 2 perc.

Amennyiben nincs folyamatban lévő szavazás, az oldal fekete háttérrel jelenik meg. Ezen a felületen **a Pollák Csengő logója** (☆) és egy **tájékoztató üzenet** (☆) látható**,** amely informálja a diákokat arról, hogy jelenleg nincs aktív szavazás, és a tanári munkaközösség a zenék feltöltésére vár.

Illusztrációs képek:




Jelenleg nincsen aktív szavazás
Töltsetek fel új zenéket

Admin oldal

Az Admin oldal egy webes kezelőfelület, amely lehetőséget biztosít az adminisztrátorok számára a zenék és szavazások kezelésére. Az oldal kizárólag Admin jogosultsággal rendelkező felhasználók számára érhető el a következő útvonalon: **[/admin]**.

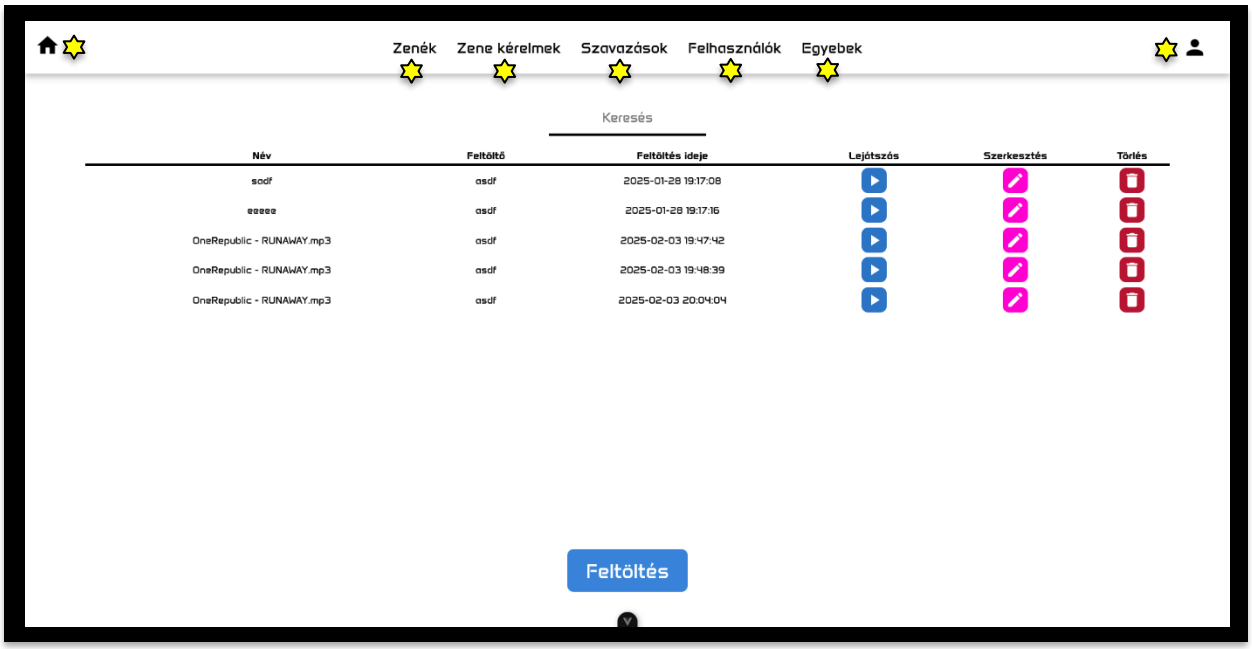
Az oldal az alábbi **öt** fő komponenst tartalmazza:

- **Zenék** (☆)
- **Elfogadásra váró zenék** (☆)
- **Felhasználók** (☆)
- **Szavazások** (☆)
- **Egyebek** (☆)

Ezen komponenseket a navigációs sávban lehet elérni a leírt nevükkel megegyező menüpont alatt.

Ezeken a komponenseken felül található egy **Otthon ikon** (☆) a **navigációs sáv bal szélén** amely a Kezdőlapra navigálja vissza a felhasználót és egy **Felhasználói ikon** (☆) a **jobb szélén** amire rányomva megjelenik az opció a kijelentkezésre.

lllusztrációs képek:



Admin oldal – Zenék

Ez a komponens szolgál az elfogadott zenék megjelenítésére, kezelésére. Itt a felhasználó láthatja az összes zenét amely a weboldalra felvan töltve egy táblában. Egy sor, egy zenét reprezentál. A felhasználó láthatja a **zene nevét** (☆), a **feltöltőjét** (☆) és a **feltöltés időpontját** (☆) és az alábbi három gombot melyek ikonokkal vannak jelezve:

- **Lejátszás / Megállítás (lejátszás/megállítás ikon)** (☆)**:** Ez szolgál a zene elindítására és megállítására amellyel a felhasználó meghalgathatja egyesével a zenéket.
- **Szerkesztés (toll ikon)** (☆)**:** Ez szolgál a zenék átnevezésére. Ezt a gombot megnyonva egy felugró ablak jelenik meg amelyben egy **beviteli mező** (☆) és kettő gomb jelenik meg. A beviteli mezőben kell beírni a zene új címét és az alatta lévő **"Megerősítés" gombbal** (☆) lehet véglegesen megváltoztatni a zene címét. Amennyiben a felhasználó meggondolja magát az átnevezés során, a "Megerősítés" gomb alatt található **"Mégse" gombbra** (☆) kattintva bezárhatja a felugró ablakot, ezzel megszakítva a műveletet.
- **Törlés (kuka ikon)** (☆)**:** Ez a gomb szolgál a feltöltött zene kitörlésére. Ezt a gombot megnyova a zene véglegesen törlődik a szerverről.

Ezenfelül található egy **kereső mező** (☆) a táblázat felett amelybe zene név alapján lehet keresni a feltöltött zenék között és egy **feltöltés gomb** (☆) ami követelményeket ignorálva, direkt módon nyújt lehetőséget zenék feltöltésére.

Illusztrációs képek:

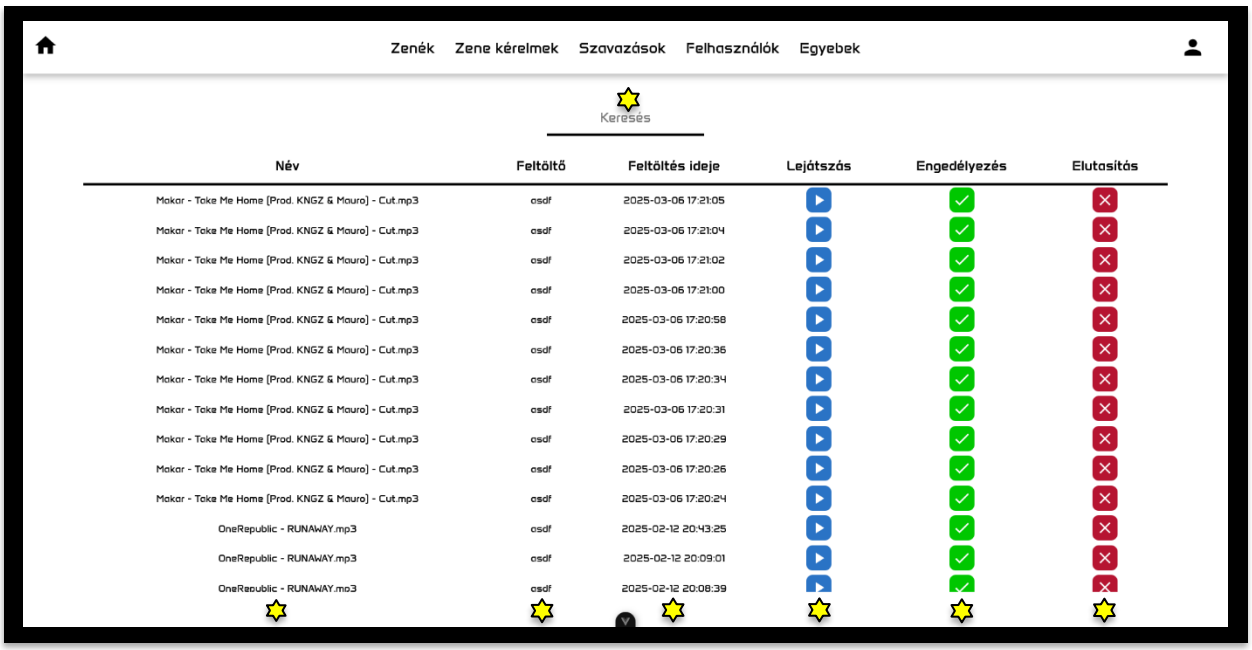
Admin oldal – Zenekérelmek

Ez a komponens szolgál az elfogadásra váró zenék megjelenítésére, kezelésére. Itt a felhasználó láthatja az összes zenét amely a weboldalra felvan töltve és elfogadásra vár, egy táblában. Egy sor, egy zenét reprezentál. A felhasználó Láthatja a **zene nevét** (☆), a **feltöltőjét** (☆) és a **feltöltés időpontját** (☆) és az alábbi három gombot melyek ikonokkal vannak jelezve:

- **Lejátszás / Megállítás (lejátszás/megállítás ikon)** (☆)**:** Ez szolgál a zene elindítására és megállítására amellyel a felhasználó meghallgathatja egyesével a zenéket.
- **Elfogadás (pipa ikon)** (☆)**:** Ez a gomb szolgál a zene elfogadására. Ezt a gombot megnyomva a zene elfogadásra kerül és áthelyezésre kerül a sima zenék listájába.
- **Elutasítás (X ikon)** (☆)**:** Ez a gomb szolgál a zene elutasítására. Ezt a gombot megnyova a zene elutasításra kerül és törlődik a listában és a szerveren lévő fájlok közül is véglegesen.

Ezenfelül található egy **kereső mező** (☆) a táblázat felett amelybe a zene neve alapján lehet keresni az elfogadásra váró zenék között.

Illusztrációs képek:



Admin oldal – Szavazások

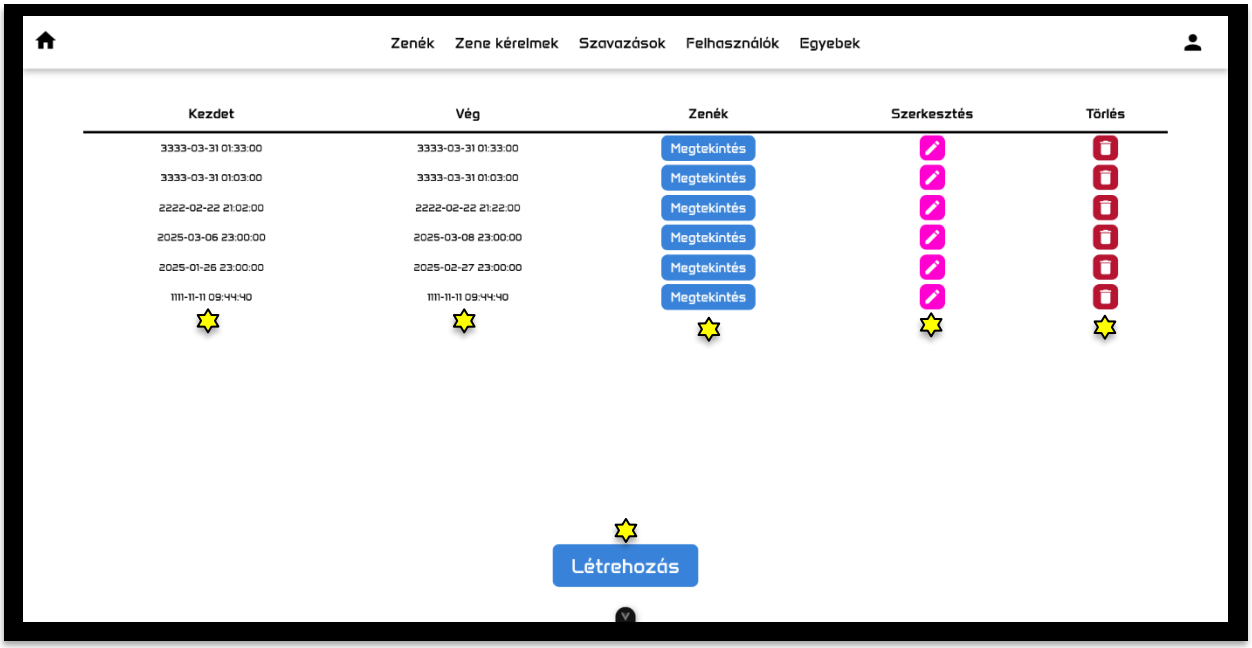
Ez a komponens szolgál a szavazati időszakok megjelenítésére, kezelésére. Itt a felhasználó láthatja az összes szavazást ami már volt, éppen van vagy lesz. Ezenfelül a felasználó létrehozhat, szerkeszthet és törölhet is szavazásokat. Egy sor, egy szavazást reprezentál. A felhasználó láthatja a **szavazás kezdetének időpontját** (☆), a **szavazás végének időpontját** (☆) és az alábbi három gombot melyekből kettő ikonokkal van jelezve:

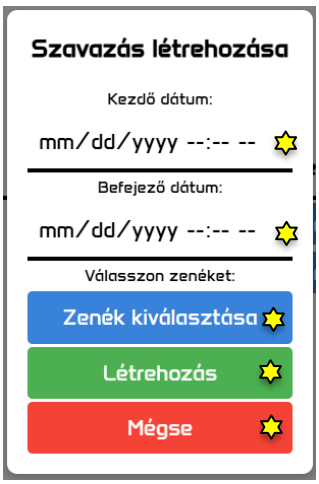
- **Megtekintés** (☆)**:** Ez a gomb szolgál a szavazásban lévő zenék megtekintésére. Ezt a gombot megnyomva felugrik egy ablak amelyben listába vannak szedve a szavazásban lévő zenéknek a címei. Az ablak alján található egy "Bezár" gomb amely ennek a felugró ablaknak a bezárására szolgál.
- **Szerkesztés (toll ikon)** (☆)**:** Ez a gomb szolgál a kiválasztott szavazásnak a szerkesztésére. Ezt a gombot megnyomva egy felugró ablak jelenik meg kettő dátum típusú beviteli mezővel és egy lenyíló menüvel, alatta egy **"Módosítás"** (☆) és **"Mégse"** (☆) gombbal. A **felső dátum mezőbe** (☆) kell megadni a szavazás kezdetének az időpontját és az alatta lévő, **második dátum mezőben** (☆) pedig a szavazás végének az időpontját. Ezen kettő mező alatt található egy **"Zene kiválasztása"** (☆) névvel ellátott gomb ami egy **lenyíló menüt** (☆) hoz fel benne az összes elfogadott zenével. Az itt kiválaszott zenék kerülnek bele a szavazásba mint megszavazható zenék. Amennyiben a felhasználó sikeresen beállította ezeket, a "Módosítás" gomb megnyomásával véglegesítheti a módosításait. Ha a felhasználó mégsem szeretne változtatás végrehajtani, a "Mégse" gomb megnyomásával bezárhatja az ablakot és elvetheti a módosításokat.
- **Törlés (kuka ikon)** (☆)**:** Ez a gomb szolgál a szavazások törlésére. Ezt a gombott megnyomva a kiválaszott szavazást véglegesen törölheti a felhasználó.

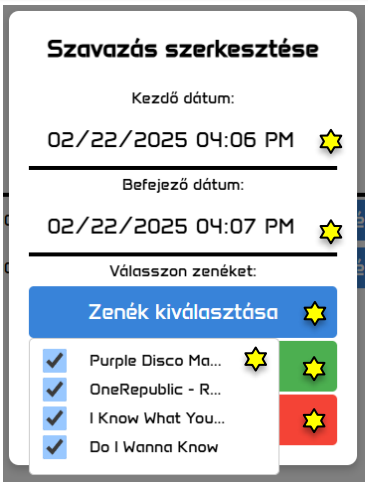
A **létrehozás gomb** (☆) a táblázat alatt található, amelynek megnyomásával egy felugró ablak jelenik meg kettő dátum típusú beviteli mezővel és egy lenyíló menüvel, alatta egy **"Létrehozás"** (☆) és **"Mégse"** (☆) gombbal. A **felső dátum mezőbe** (☆) kell megadni a szavazás kezdetének az időpontját és az alatta lévő, **második dátum mezőben** (☆) pedig a szavazás végének az időpontját. Ezen kettő mező alatt található egy **"Zene**

kiválasztása" (☆) névvel ellátott gomb ami egy **lenyíló menüt** (☆) hoz fel benne az összes elfogadott zenével. Az itt kiválaszott zenék kerülnek bele a szavazásba mint megszavazható zenék. Amennyiben a felhasználó sikeresen beállította ezeket, a "Létrehozás" gomb megnyomásával létrehozhatja a szavazást. Ha a felhasználó mégsem szeretne szavazást létrehozni, a "Mégse" gomb megnyomásával bezárhatja az ablakot elvetve a műveletet.

Illusztrációs képek:







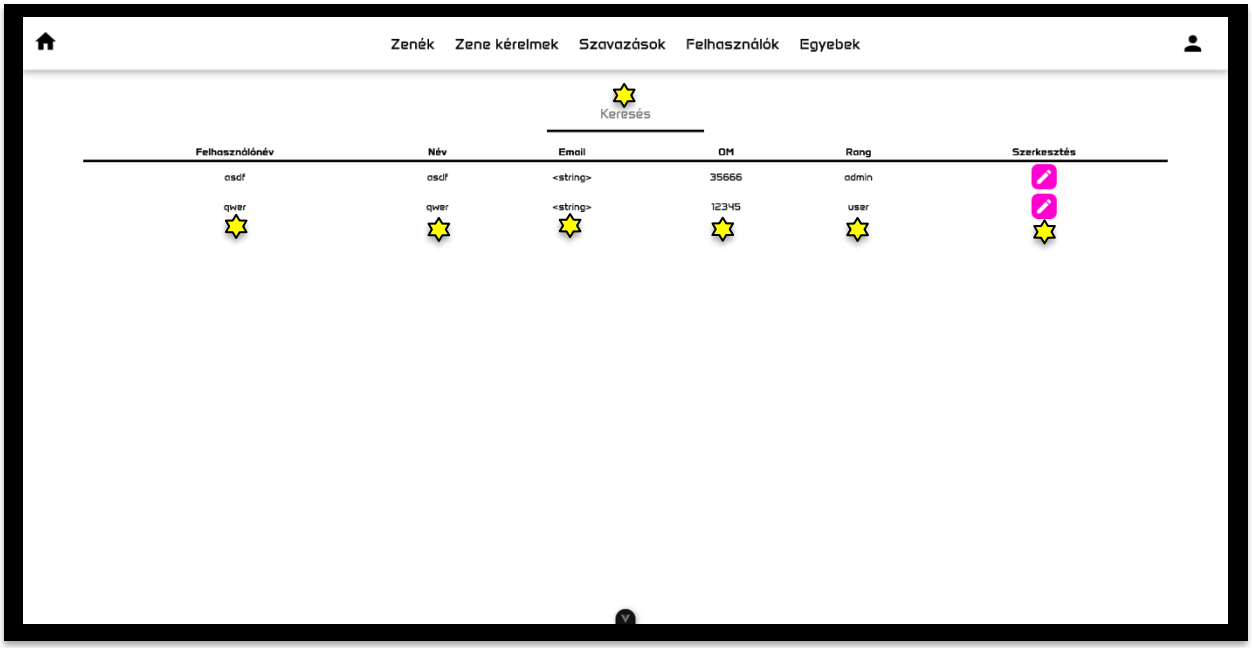
Admin oldal – Felhasználók

Ez a komponens szolgál a rendszerbe regisztrált felhasználói fiókok megjelenítésére, kezelésére. Itt a felhasználó láthatja az összes fiókot egy táblázatban. Egy sor, egy fiókot reprezentál. A felhasználó láthatja a többi fehasználó **felhasználónevét** (☆), a krétába jegyzett **valós nevét** (☆), az **email címét** (☆), az **OM azonosítóját** (☆) és **rangját** (☆). Ezenfelül láthat egy **szerkesztés gombot toll ikonnal jelölve** (☆) amely használatával a felhasználó szerkesztheti más felhasználók rangját és jelszavát.

Erre a **szerkesztés gombra** kattintva egy felugró ablak jelenik meg egy **beviteli mezővel** (☆), egy **lenyíló mezővel** (☆), egy **"Szereksztés"** (☆) és egy **"Mégse"** (☆) **gombbal**. A beviteli mezőbe a felhasználó új jelszavát lehet beírni, amely megfelel a regisztrációnál is feljegyzett kritériumoknak, míg a lenyíló menüben a rangot lehet kiválasztani. A szerkesztéshez nem muszály mind a kettőt változtatni, lehet külön-külön is. Amennyiben a felhasználó mindent beírt / kiválasztott amit szeretett volna, a **"Szerkesztés" gombra** kattintva véglegesítheti a változásokat. Amennyiben a felhasználó mégsem szeretne változtatni, a **"Mégse" gombra** kattinva bezárhatja az ablakot, ezzel megszakítva a szerkesztés.

Ezenfelül található egy **kereső mező** (☆) a táblázat felett amelybe a felhasználó valós, krétába feljegyzett neve alapján lehet keresni a felhasználói fiókok között.

Illusztrációs képek:



Admin oldal – Egyebek

Ez az oldal szolgál az egyéb, egyszerűsítési funkciók megjelenítésére. Ezen az oldalon található az alábbi kettő gomb középre igazítva egymás alatt:

- **Szavazásban lévő zenék letöltése** (☆)**:** Ez a gomb szolgál a szavazásban lévő zenék letöltésére .zip formátumban szükséglet esetén. Ezt a gombot megnyomva felugrik egy fájlkezelő ablak ahol a felhasználó kiválaszthatja, hogy hova szeretné letölteni a zenéket.
- **Nyertes zene letöltése** (☆)**:** Ez a gomb szolgál az utolsó szavazásban lévő nyertes zene letöltésére. Ezt a gombot megnyomva felugrik egy fájlkezelő ablak ahol a felhasználó kiválaszthatja, hogy hova szeretné letölteni a zenét. Ez a gomb létfontosságú a szoftver hasznosságának érdekében, ugyanis így egy kattintásból letölthető a zene a megfelelő helyre, ahonnan már egyből működőképesen szólhat az új szám az iskolacsengőbe.

Illusztrációs képek:

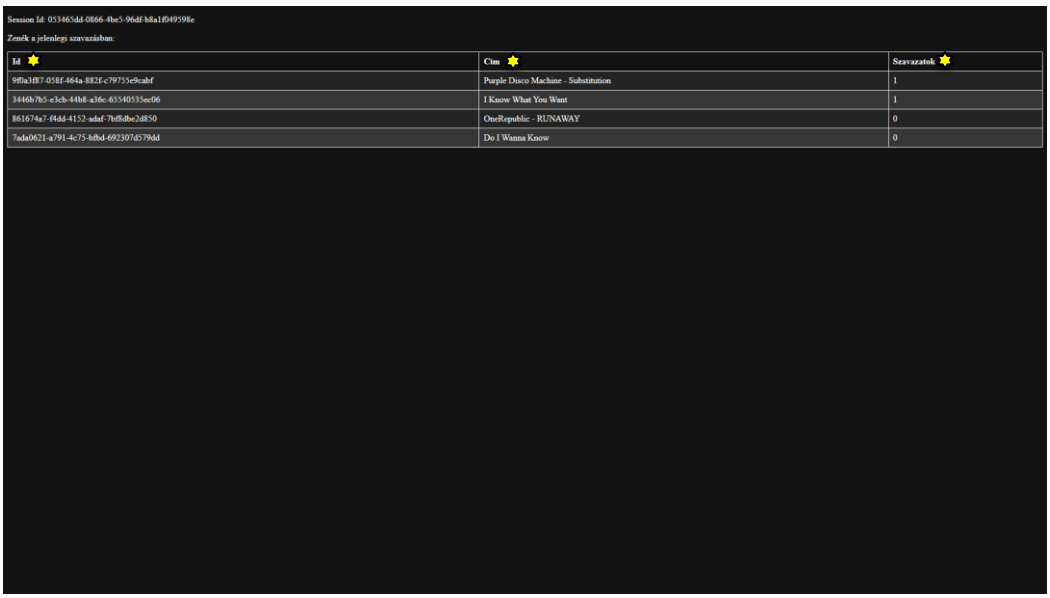
Tv (Szerver Oldali)

Ezen a szerveroldali komponensen a felhasználók egy letisztult felületen, kliens futtatása nélkül tekinthetik meg a jelenlegi szavazás eredményeit. Az adatok egy táblázatban jelennek meg, amelyben látható a **zeneszám azonosítója** (☆), **címe** (☆), valamint a rá leadott **szavazatok száma** (☆).

Az oldal az alábbi útvonalon érhető el: **[http://localhost:3300/view/tv]**

Amennyiben nincs folyamatban lévő szavazás, az oldal fekete háttérrel jelenik meg. Ezen a felületen egy **tájékoztató üzenet** (☆) látható**,** amely informálja a felhasználót arról, hogy jelenleg nincs aktív szavazás.

Illusztrációs képek:



Elfogadásra váró zenék (Szerver Oldali)

Ezen a szerveroldali komponensen a felhasználók egy letisztult felületen, kliens futtatása nélkül tekinthetik meg a jelenleg elfogadásra váró zenéket. Az adatok egy táblázatban jelennek meg, amely tartalmazza a **zeneszám azonosítóját** (☆), **címét** (☆), **feltöltésének** (☆) és **frissítésének** (☆) idejét, a **zenekezelő tábla azonosítóját**, valamint a **feltöltő azonosítóját** (☆).

Az oldal az alábbi útvonalon érhető el: **[http://localhost:3300/view/pending-songs]**

Amennyiben nincs elfogadásra váró zene, az oldal fekete háttérrel jelenik meg. Ezen a felületen egy **tájékoztató üzenet** (☆) látható**,** amely informálja a felhasználót arról, hogy jelenleg nincs elfogadásra váró zene.

Illusztrációs képek:

Összefoglalás, Köszönetnyilvánítás

Önértékelés és a projekt hasznosulása

A **Pollak Csengő** fejlesztése során csapatként dolgoztunk együtt, és rengeteget tanultunk nemcsak a szoftverfejlesztési folyamatokról, hanem a különböző technológiák hatékony alkalmazásáról is. A projekt során használt programozási nyelvek és keretrendszerek mélyebb megértésére tettünk szert, miközben megtapasztaltuk a csapatmunka kihívásait és előnyeit.

Az elkészült munkánkat az alábbi szempontok szerint értékeljük:

- **Működés és stabilitás:** A csapat közös erőfeszítéseinek köszönhetően a szoftver megbízhatóan és az elvárásoknak megfelelően működik. Teszteléseink során sikerült az esetleges hibákat időben javítani, így egy stabil rendszert hoztunk létre.
- **Kódminőség és fejleszthetőség:** A projekt során nagy hangsúlyt fektettünk a tiszta és átlátható kódírásra, hogy a későbbiekben könnyen bővíthető és karbantartható maradjon a rendszer.
- **Csapatmunka és tanulás:** A fejlesztés során sokat tanultunk a verziókezelésről, a különböző programozási nyelvek és keretrendszerek használatáról, valamint az egyes komponensek hatékony összekapcsolásáról. Ezen kívül fejlesztettük kommunikációs és problémamegoldó képességeinket is.
- **Felhasználói élmény:** A szoftver kialakításánál kiemelten fontos volt számunkra, hogy intuitív és könnyen használható felületet hozzunk létre, amely egyszerűvé és kényelmessé teszi a mindennapi használatot.

A program hasznosulása a továbbiakban

A Pollak Csengő nemcsak a vizsga teljesítésére készült, hanem hosszú távon is hasznosítható megoldást nyújt. Az alábbi területeken látunk lehetőséget a program jövőbeli alkalmazására:

- **Intézményi felhasználás:** A szoftver könnyedén integrálható különböző oktatási intézményekben vagy munkahelyeken, ahol szükség van egy megbízható, automatizált csengőrendszerre.
- **Folyamatos fejlesztés:** A projektet úgy terveztük, hogy a jövőben továbbfejleszthető legyen, akár új funkciókkal, adminisztrációs felülettel vagy testreszabható beállításokkal.
- **Tapasztalatok hasznosítása:** Az itt megszerzett tudást nemcsak ennél a projektnél, hanem későbbi fejlesztéseink során is kamatoztathatjuk. Megtanultuk, hogyan lehet hatékonyan együttműködni egy szoftverfejlesztési projektben, és milyen módszerekkel érhetjük el a legjobb eredményt.

A fejlesztési folyamat során nemcsak egy működőképes rendszert hoztunk létre, hanem értékes tapasztalatokkal is gazdagodtunk, amelyeket a jövőben is hasznosítani tudunk.

Köszönetnyilvánítás

Szeretnénk köszönetet mondani mindazoknak, akik támogatták a Pollak Csengő fejlesztését és hozzájárultak a projekt sikeréhez.

Külön köszönjük:

- **Csapattagjainknak**, akik kitartó munkájukkal és elkötelezettségükkel hozzájárultak a szoftver létrehozásához. A közös munka során rengeteget tanultunk egymástól, és értékes tapasztalatokat szereztünk.
- **Tanárainknak és mentorainknak**, akik iránymutatásaikkal és szakmai tanácsaikkal segítették a fejlesztés folyamatát.

Ez a projekt nemcsak egy szoftver létrehozásáról szólt, hanem egy fejlődési folyamatról is, amely során új készségeket sajátítottunk el, és valódi csapatként dolgoztunk együtt. Köszönjük mindenkinek, aki bármilyen módon hozzájárult ennek a munkának a sikeréhez!

Irodalomjegyzék

- **Bella-Iskolacsengő - Konkurencia**
  - Elérhető: https://www.bella-iskolacsengo.hu/
- **Gipen Iskolacsengő - Konkurencia**
  - Elérhető: https://iskolacsengo.gipen.hu/
- **Stackoverflow – A szoftverfejlesztési problémák szent grálja**
  - Elérhető: https://stackoverflow.com/questions
- **PgAdmin – Az adatbázis kezelő dokumentációja**
  - Elérhető: https://www.pgadmin.org/docs/
- **Postgres 16.6 – Az adatbázis dokumentációja**
  - Elérhető: https://www.postgresql.org/docs/16/index.html
- **Vue – Frontedhez használt könyvtár dokumentációja**
  - Elérhető: https://vuejs.org/guide/introduction.html
- **NestJs – A backendhez használt könyvtár dokumentációja**
  - Elérhető: https://docs.nestjs.com/
- **Playwright – A teszteléshez használt könyvtár dokumentációja**
  - Elérhető: https://playwright.dev/docs/intro
- **Infojegyzet – A záródolgozathoz használt szempontok**
  - Elérhető**:** https://infojegyzet.hu/webszerkesztes/dokumentacio/

A három könyv, amelyek segítettek a fejlesztés során, különösen Vue, TypeScript és backend témakörben:

- **Heitor Ramon Ribeiro: "Vue.js 3 Cookbook":** Részletes példákkal mutatja be a Vue.js 3 fejlesztési lehetőségeit, és gyakorlati megoldá sokat kínál a frontend fejlesztéshez.
- **Boris Cherny: "Programming TypeScript":** Mélyrehatóan tárgyalja a TypeScript nyelvet, annak előnyeit és alkalmazási lehetőségeit nagyobb méretű projekteknél.
- **Kamil Myśliwiec: "NestJS: A Progressive Node.js Framework":** A NestJS alapítója által írt könyv, amely bemutatja a modern backend fejlesztést TypeScript alapokon.