ETDK-dolgozat

Biró Norbert

Kelemen Gergő

XXVIII. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Informatika II.: innovatív számítástechnikai termékek, alkalmazások szekció

Kolozsvár, 2025. május 15–18.

VoteHub

Szavazórendszer hivatalos intézmények és közösségi szervezetek számára

Szerzők:

Biró Norbert

III. év, Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Kelemen Gergő

III. év, Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Témavezetők:

dr. Simon Károly, egyetemi adjunktus,

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Kelemen-Fehér Dénes Bálint, szoftverfejlesztő,

Codespring

Rácz-Nagy Katalin, szoftverfejlesztő,

Codespring

CODESPRING

2025. MÁJUS 15-18. KOLOZSVÁR
XXVIII. ERDÉLYI TUDOMÁNYOS DIÁKKÖRI KONFERENCIA
REÁL- ÉS HUMÁNTUDOMÁNYOK
CONFERINȚA ȘTIINȚIFICĂ STUDENȚEASCĂ DIN TRANSILVANIA, EDIȚIA A 28-A
28TH TRANSYLVANIAN STUDENTS' SCIENTIFIC CONFERENCE
ȘTIINȚE REALE ȘI UMANISTE // FORMAL AND EMPIRICAL SCIENCES

VoteHub: Szavazórendszer hivatalos intézmények és közösségi szervezetek számára

Biró Norbert, III. év, Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar
Kelemen Gergő, III. év, Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

dr. Simon Károly, egyetemi adjunktus,
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar
Kelemen-Fehér Dénes Bálint, szoftverfejlesztő,
Codespring
Rácz-Nagy Katalin, szoftverfejlesztő,
Codespring

Az intézmények számára sokszor kihívást jelent hatékonyan és biztonságosan lebonyolítani belső szavazásaikat. Nagy létszám estén a kézfelemeléssel történő szavazás lassú, és nő a hibalehetőségek száma. Az automatizált, digitális megoldások pedig gyakran nem nyújtanak elegendő lehetőséget finomhangolásra, valós idejű visszajelzésre, illetve a szavazások eredményeinek biztonságos és rendszerezett tárolására.

A VoteHub alkalmazás célja, hogy egy megbízható felületet biztosítson intézmények számára, ahol lebonyolíthatják a gyűléseiket, és szavazhatnak azok napirendi pontjairól. Az intézmények az alkalmazáson belül zárt körű fórumokba rendeződnek. A rendszer két szerepkört különböztet meg: a moderátorok menedzselik a tagok listáját, a fórumokat, a gyűléseket és a napirendi pontokat; a fórum tagok részt vehetnek és szavazhatnak a gyűléseken. A rendszer lehetőséget nyújt a gyűlésen készült hangfelvétel alapján átíratok kigenerálására is.

A rendszer három részből tevődik össze: egy webes felület és egy mobilalkalmazás, melyeket egy szerver szolgál ki. A dolgozat bemutatja az alkalmazás funkcionalitásait és felépítését, részletezi a használt mintákat, technológiákat és eszközöket.

Témavezetői igazolás

Alulírott dr. Simon Károly igazolom, hogy *Biró Norbert és Kelemen Gergő* hallgatók a(z) *VoteHub: Szavazórendszer hivatalos intézmények és közösségi szervezetek számára* című dolgozatát az alulírott szakmai irányításával készítették el, és javaslom az említett tudományos munka bemutatását a 2025-ös XXVIII. Reál- és Humántudományi Erdélyi Tudományos Diákköri Konferencián (ETDK).

Mindezek mellett igazolom, hogy a dolgozat megfelel a korszerű tudományos kritériumoknak, nem OTDK-n vagy nemzetközi szintű konferenciákon már bemutatott dolgozat, már megvédett államvizsga vagy magiszteri dolgozat, sem korábban már publikált dolgozat.

dr. Simon Károly, egyetemi adjunktus,
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Kolozsvár, 2025. április 27.

Tartalomjegyzék

Bevezető

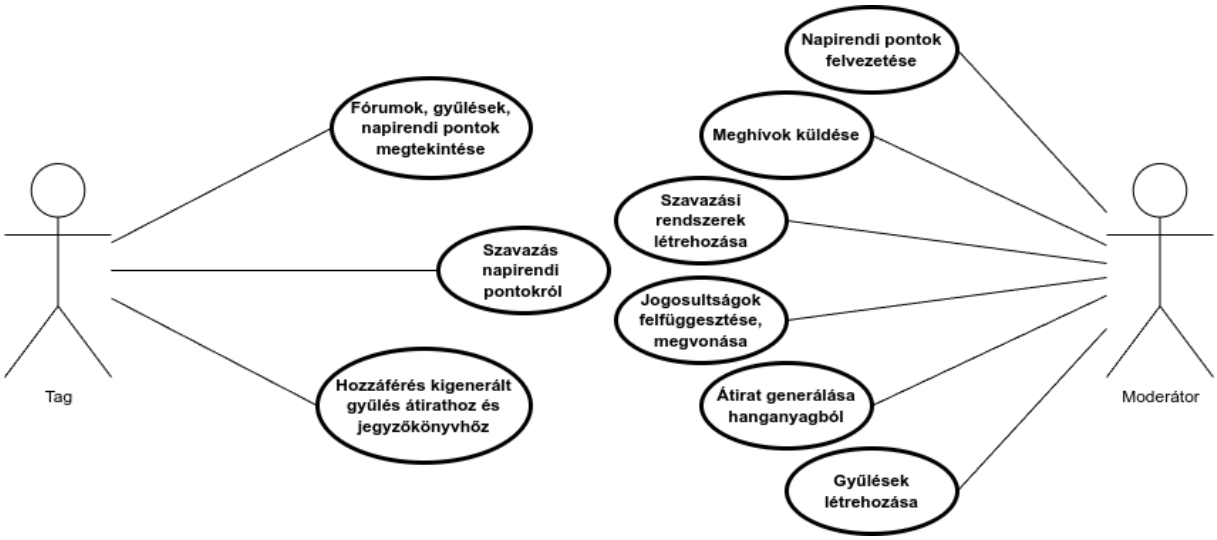
A VoteHub alkalmazás egy olyan felületet biztosít, amely egyszerűvé, kényelmessé és biztonságossá teszi az intézményi szavazások és gyűlések lebonyolítását. Minden intézmény saját szabályzattal rendelkezik a gyűlésein felmerülő szavazásokat illetően. Egyik kézenfekvő megoldás a kézfelemeléssel történő szavazás, viszont nagy létszám esetén ez a folyamat időigényes, hiszen össze kell számolni egyenként a szavazatokat, majd ezek alapján ki kell számítani, hogy mi lett a végső döntés, ha egyáltalán született ilyen, és előfordulhatnak hibák is. Érdemes ezért ezt a folyamatot automatizálni, melyre gyakran használt eszközök a Google Forms, vagy a Microsoft Forms. Ezekben az alkalmazásokban űrlapokat hozhatunk létre melyek megosztásra kerülnek az intézmény tagjai között, ők kitöltik, majd születik egy döntés a szavazataik alapján. Viszont egy hosszabb gyűlésen ez a megközelítés sem kézenfekvő, hiszen hiányzik az élő visszajelzés és nem követhető a szavazás aktuális állapota. A fent említetteken kívül más alkalmazások is használatosak és minden intézménynek megvannak az okai, hogy miért választanak egy felületet a másik helyett. A használt platformok nagy változatossága minden bizonnyal annak tudható be, hogy a már meglévő megoldások túl merevek, nem eléggé testre szabhatóak ahhoz, hogy valamelyik szélesebb körben elterjedjen az intézmények közt.

A VoteHub, a már létező rendszerekkel szemben, egy rugalmasabb megoldást biztosít, ami a szavazások, illetve gyűlések lebonyolítását egyszerűsíti, és ugyanakkor ezt a folyamatot teljesen testre szabhatóvá teszi. A VoteHub zárt körű fórumokba rendszerezi az intézmények gyűléseit, ahová csak meghívóval lehetséges belépni, így biztosítva, hogy csak a megfelelő jogosultsággal rendelkező felhasználók férhessenek hozzá az érzékeny információkhoz. Emellett az alkalmazás lehetőséget nyújt online, vagy akár hibrid jellegű gyűlések lebonyolítására is, ahol a gyűlés állapota élőben követhető, és a felhasználók aktívan rész vehetnek a szavazásban. Illetve, statisztikák, és a gyűlésen elhangzottak alapján készült átiratok kigenerálása is lehetséges.

A dolgozatban bemutatásra kerülnek a VoteHub alkalmazás különböző aspektusai, mint például a rendszer architektúrája, a különböző funkcionalitások, illetve a használt eszközök és technológiák. Az 1. fejezet egy általános képet ad a rendszer felépítéséről és működéséről. A 2. fejezetben a Szerver architektúrája, illetve funkcionalitásai lesznek részletezve. A 3. fejezetben a webes platform működését, illetve az ehhez használt technológiákat tárgyaljuk. A 4. fejezetben a mobil alkalmazás kerül bemutatásra. Az 5. fejezetben az alkalmazáshoz tartozó minőségbiztosító eszközökről, illetve a rendszer teszt környezetbe való kitelepítésesről

lesz szó. A dolgozatot néhány továbbfejlesztési lehetőség felsorolásával zárjuk.

A projekt fejlesztése 2024-ben kezdődött Székelyudvarhelyen, a Codespring mentorprogram, illetve az ITPlus Cluster mentorprogram keretein belül, mint szakmai gyakorlatos projekt. A nyári időszak alatt a dolgozat írói mellé csatlakozott a fejlesztésben Márton Áron, aki az Eötvös Loránd Tudományegyetemen végzi a tanulmányait. Ezt követően az alkalmazás a csoportos projekt nevű egyetemi tantárgy keretein belül lett továbbfejlesztve. Ebben az időszakban Incze Zoltán, László Botond, László Tamás és Sándor László Márk egyetemi hallgatók is hozzájárultak a fejlesztéshez. A szerzők köszönetet szeretnének nyilvánítani mentoraiknak, illetve szakmai koordinátoruknak, akik segítették a projekt létrejöttét.



1. ábra. VoteHub alkalmazás alkalmazási esetei

1. A projektről

A VoteHub projekt három fő részből áll: Web- és Mobil alkalmazás, melyeket egy központi szerver szolgál ki. Az alkalmazás fő célja a gyűlések kényelmes és biztonságos lebonyolítása zárt körű fórumokba szervezve. Az aktuális fejezet a funkcionalitások áttekintését és a rendszerbeli kapcsolatokat hívatott bemutatni.

1.1. Funkcionalitások

Fórum szinten a rendszer két felhasználói szerepkört különböztet meg: tag és moderátor. Az 1. ábrán láthatóak az alkalmazás használati esetei. A két különböző szerepkör az eltérő jogosultságok miatt van bevezetve. Egy regisztrált felhasználó különböző fórumok esetében eltérő szerepköröket tölthet be, de előfordulhat, hogy mindkét szerepkör jogosultságával rendelkezik. A felhasználók jogosultságokat meghívókon keresztül kaphatnak, egy személy mind a két szerepkörhöz kaphat meghívót egy fórum esetében. A felhasználók eldönthetik, hogy elfogadják vagy elutasítják a meghívót. Minden regisztrált felhasználó létrehozhat fórumot, ebben az esetben automatikusan moderátor jogokkal rendelkezik a létrehozott fórumon belül.

Moderátorok fő funkciói:

- Gyűlések és napirendi pontok létrehozása. Az alkalmazás központi eleme a fórum, az ezen belüli gyűlések, napirendi pontok kezelése a moderátorok egyik fő feladatköre.
- Meghívok küldése más felhasználóknak. A meghívás e-mail cím alapján történik.

A meghívott személy nem kell előzetes regisztrációval rendelkezzen. Felhasználó meghívásakor a szerepkört is meg kell határozni.

- Fórum, lezáratlan gyűlések és napirendi pontok adatainak szerkesztése. Ide tartoznak az entitás nevek, leírások, betervezett időpontok módosítása.
- Szavazási rendszereket hozhatnak létre. A szavazási rendszer definiálja a szavazások menetét, ennek segítségével dönthető el a szavazás eredménye.
- Fórumhoz tartozó más felhasználók jogosultságainak felfüggesztése vagy megvonása. Egy moderátor képes tagok és akár más moderátorok jogait ideiglenesen felfüggeszteni, vagy kizárni teljesen a fórumból. Napirendi pontok szerint is képes megvonni a tagok szavazási lehetőségét (ez esetben nincs fórumból való kizárás).
- Gyűlések lebonyolítása. Egy gyűlést csak egy moderátor indíthat el. A gyűlés megkezdését követően a moderátor állíthat be megszavazandó napirendi pontot. Láthatja a belépett tagok, illetve leadott szavazatok számát. A moderátor felelős a szavazások indításáért és lezárásáért. Lezárás után láthatóvá válik a napirendi pont eredménye, amely tartalmazza, hogy melyik tag melyik opcióra szavazott, illetve melyik opcióra hányan szavaztak százalékban kifejezve.
- Hanganyagból generált jegyzőkönyv létrehozása. A gyűlés alatt rögzített hanganyagból a moderátorok átiratot generálhatnak, amit módosíthatnak és megoszthatnak más felhasználókkal.

Tagok fő funkciói:

- Fórumok, gyűlések, napirendi pontok megtekintése.
- Aktív résztvevői lehetnek gyűléseknek.
- Szavazati joggal rendelkeznek. Ezek a szavazatok határozzák meg a napirendi pont eredményét.
- Hozzáférnek a gyűlés kigenerált átiratához és jegyzőkönyvéhez, amit letölthetnek, megoszthatnak.

1.1.1. Szavazórendszer leírása

A szavazások eredményének meghatározására bevezetésre került a szavazórendszer, amely meghatározza a következő kritériumokat:

1. Kvórum, vagyis a tagok hány százalékának kell jelen lennie a határozatképességhez.

2. Egy szavazási opcióra a szavazatoknak hány százalékának kell érkeznie, ahhoz, hogy az legyen a nyertes opció. Megadható, hogy a jelen levő résztvevők vagy a fórumon belül levő összes tagok számából legyen kiszámítva az opcióra leadott szavazatok százaléka.

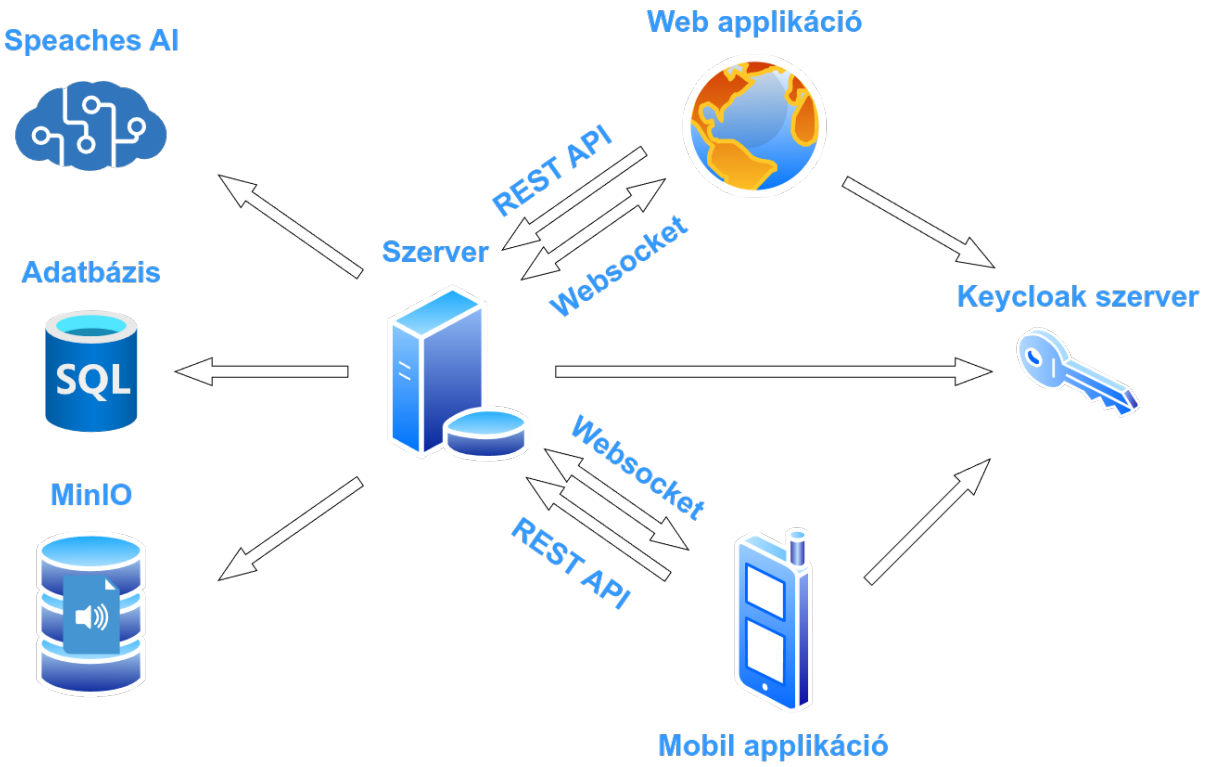
3. Beállítható, hogy a tartózkodás elfogadott vagy sem, mint helyes szavazási eredmény.

A 2. kritérium segítségével megadható, hogy egy teljes közösség létszámából milyen eloszlással szavaztak egyes opciókra, attól függetlenül, hogy minden tag részt vett-e a szavazáson, vagy sem.

A 3. kritérium bevezetésére azért volt szükség, mert vannak esetek, amikor egy tartózkodás nem tekinthető helyes opciónak, ekkor a szavazásnak nincs eredménye.

A moderátor létrehozhat szavazórendszereket sablonokból, a sablonok értékeit módosítva, vagy teljesen új rendszert is meghatározhat. A sablonból létrehozott szavazórendszerek esetében alapértelmezetten a tartózkodás nem lehet nyertes opció és az opciókra a szavazatok százaléka a jelen levő tagok számából lesz kiszámítva. Egyedi nevek segítségével megkülönböztethetők a létrehozott szavazórendszerek. A létrehozásuk után a moderátor hozzárendelheti ezeket a gyűlések napirendi pontjaihoz, egy adott napirendi pont eredménye a hozzárendelt szavazórendszer kritériumai alapján lesz kiszámolva.

Az eredmények kiszámítása a 2. kritérium meghatározásával kezdődik, továbbiakban T_0. Ezt követően minden opcióra kiszámolódik a leadott szavazatok száma (a_i), százalékban is kifejezve, a következő képlet segítségével: $p_i = a_i/T_0$, ahol p_i egy opcióra leadott szavazatok száma százalékban kifejezve és $\sum_{k=1}^{n} p_k = 1$. Ezt követően, ha a tartózkodás nem számít a végeredménybe, akkor a leadott szavazatok számából levonódik a tartózkodások száma, vagyis $T_1 = T_0 - A$. A továbbiakban T_1-el fogunk számolni. Amennyiben lehet tartózkodni, akkor $A = 0$ értéket választva használjuk T_1-t. Ezután következik a fent említett kritériumok ellenőrzése. A szavazás nem tekinthető sikeresnek, amennyiben kevesebb tag volt jelen, mint az 1. kritériumban megadott minimális létszám, vagy $T_1 = 0$, illetve akkor sem, ha több opcióra azonos számú szavazat érkezett. Minden más esetben az az opció tekinthető nyertesnek,



2. ábra. A VoteHub alkalmazás architektúrája

amely esetén p_i értéke nagyobb mint a 2. kritériumban megadott érték. A szavazásnak nincs meghatározott végeredménye, amennyiben nem volt ilyen opció.

A szélesebb körben alkalmazott szavazás típusokhoz a rendszer nyújt sablonokat, ezek a következők:

- Egyszerű többség - $50\% + 1$ szavazat egy opcióra, hogy döntés szülessen ($p_i = 0.5$)
- Kétharmados többség - a tagok $2/3$-a egy opcióra kell szavazzon a határozathoz ($p_i = 0.(66)$)
- Teljes konszenzus - minden tag azonos opcióra kell szavazzon ($p_i = 1$)

1.2. Architektúra

A 2. ábrán látható a VoteHub alkalmazás architektúrája, amely három fő szolgáltatásból áll: egy Web- és Mobil alkalmazás, melyeket egy központi szerver szolgál ki.

- Szerver - A RESTful API-t biztosító szerver Kotlin nyelvben íródott, Spring keretrendszer felhasználásával, amely biztosítja a komponensek közötti hatékony kommunikációt

- Adatbázis - A rendszer adatbázisa egy MySQL adatbázis. Az adatbázis migráció Liquibase segítségével történik.
- Mesterséges intelligencia alapú átirat készítő szolgáltatás - Gyűléseken rögzített hanganyagok átírására a Speaches AI-t használja a rendszer, amely az OpenAI Whisper általános célú beszédfelismerő modelljeire épül.
- Fájl alapú tároló rendszer - A fájlok tárolásáért a MinIO szolgáltatás felelős. Minden átiratra érkező audió fájl itt tárolódik, ezek elérhetőek a felhasználók számára is.
- Web kliens - A web kliens az adminisztratív munkák elvégzésére biztosít felületet a moderátorok számára. Az alkalmazás React-ban íródott Typescript-et felhasználva, az állapotkezelés Redux segítségével valósult meg, a fehasználó felület kinézete pedig Material UI könyvtárat felhasználva, a Material Design elvek szerint van megvalósítva.
- Mobil kliens - A mobil kliens a tagoknak lehetőséget nyújt a szavazásra. Az alkalmazás React-Native-ban íródott, a kód fordítása és csomagolása Expo segítségével történik, a felhasználói felület elemeihez a React-Native Paper könyvtár van felhasználva.
- Keycloak - A felhasználók bejelentkezésére használt szolgáltató a Keycloak szerver. JWT tokenek alapján ad lehetőséget a felhasználók személyazonosságának ellenőrzésére.

2. Szerver

A VoteHub rendszer központi eleme a szerver, amely az adatokat tárolja, elvégzi az üzleti logika megvalósításához szükséges műveleteket, kiszolgálja a Web- és Mobil klienseket, elmenti a rögzített hanganyagokat és átiratokat készít belőlük.

2.1. Funkcionalitások

Ez a fejezet a VoteHub alkalmazás Szerverének funkcionalitásait mutatja be működés és implementáció szempontból.

2.1.1. RESTful API

A Szerver és Kliensek közti kommunikáció RESTful API-n keresztül történik. Ennek segítségével valósul meg az adatlekérés, adat módosítás és egyéb funkcionalitások. A kommunikáció JSON és Multipart formátumban történik. Alkalmazva van a DTO (Data Transfer Object) tervezési minta, amely lehetővé teszi, hogy a kommunikáció ne függjön az adatbázis sémától és a rendszer modelljeitől. A modell- és DTO osztályok közötti megfeleltetés az org.mapstruct:mapstruct könyvtár segítségével történik.

Az API fő végpontjai:

- `/forums` – fórummal kapcsolatos útvonalak
- `/forums/forum_id/meetings` – gyűlésekkel kapcsolatos útvonalak
- `/forums/forum_id/meetingsmeeting_id/topics` – napirendi pontokkal kapcsolatos útvonalak

2.1.2. Adatbázis hozzáférés és modellek

```
@Repository
interface MeetingAudioRepository: JpaRepository<MeetingAudio, Long> {
    fun getAllByMeetingForumIdAndMeetingId(forumId: Long, meetingId: Long)
    fun deleteByMeetingForumIdAndMeetingId(forumId: Long, meetingId: Long)
}
```

1. kódrészlet. Példa JpaRepository-ra

A Szerver és az adatbázis kapcsolat JDBC [11] (Java Database Connectivity) technológia segítségével épül fel, az org.springframework:spring-jdbc könyvtárat felhasználva, amely blokkoló (szinkron) kérésekkel dolgozik.

Az alkalmazás modelljeinek a létrehozásában és tárolásában a JPA (Java Persistence API) segít. Leképezést biztosít a modell osztályok és adatbázis táblák közt, egyszerűsíti az adatbázis hozzáférést, nincs szükség bonyolult és ismétlődő SQL lekérdezések megírására. A Spring DATA JPA tovább egyszerűsíti az adathozzáférési réteg implementációját. Az általa biztosított JpaRepository-t kiterjesztő interfészek segítségével automatikusan generálja a megfelelő adathozzáférési komponenseket és adatbázis lekérdezéseket [5].

Az 1. kódrészletben látható a JpaRepository használata. A MeetingAudioRepository öröklődik a JpaRepository-ból, amely generikus paramétere a menedzselt MeetingAudio osztály és az entitás elsődleges kulcsának típusa (Long). Kompiláláskor a Spring Data JPA generál egy implementációt, amely alapértelmezett módon tartalmazza a gyakori adathozzáférési műveleteket, mint például a save(), getReferenceById() vagy findAll(). Ezeken kívül saját függvények definiálhatók, ahol a Spring Data JPA a metódus nevéből képes generálni az adatbázis lekérdezést. Az 1. kódrészletben a getAllByMeetingForumIdAndMeetingId() függvény esetében a Spring Data JPA által generált implementáció képes a fórum és a gyűlés elsődleges kulcsa alapján az összes hanganyagot visszatéríteni. Bonyolultabb lekérdezések az interfész metódusain elhelyezett annotációk segítségével határozhatóak meg, JPQL felhasználásával.

Az adatbázis séma kényelmes és biztonságos migrálása a Liquibase könyvtár felhasználásával valósul meg, YAML leíró fájlok alapján [4].

2.1.3. Felhasználói szerepkörök megvalósítása

Rendszer fórum szinten két felhasználói szerepkört biztosít: Moderátor és Tag. Minden regisztrált felhasználónak van egy identitása. Adatbázis szinten a ForumModerator és a ForumMember tábla külső kulccsal hivatkozik az identitás elsődleges kulcsára. A felhasználók a fórumokhoz jogosultságokat meghívókon keresztül kaphatnak, ahol a meghívó tartalmazza a szerepkört. Egy felhasználó egy fórumhoz mind a két szerepkörhöz kaphat meghívót, és mind a két jogosultsággal rendelkezhet. Fórum létrehozásakor automatikusan moderátor joggal rendelkezik a létrehozó felhasználó.

A Szerver gondoskodik a REST végpontok védelméről. Egy adott kérés esetén a JWT

tokenben szereplő e-mail cím alapján a szerver ellenőrzi, hogy a felhasználó rendelkezik a kérés teljesítéséhez szükséges jogosultsággal. A tokenek validálása a com.nimbusds:nimbus-jose-jwt könyvtár segítségével történik [12], amely a JWT token aláírását is ellenőrzi a Keycloak által biztosított publikus kulcs segítségével. Ha a felhasználó nem rendelkezik a megfelelő jogokkal, akkor a Szerver nem teljesíti a kérést és 403 Forbidden (Tiltott) HTTP státusz kóddal válaszol.

A Web- és Mobil alkalmazások esetében az adott felhasználó jogosultsága a válasz DTO-ban szerepel, ez által a kliensek eltérő vizuális felületeket biztosíthatnak.

2.1.4. WebSocket kommunikáció

A Web- és Mobil alkalmazások használata közben előfordul, hogy a rendszerbeli adatok állapotának változásáról a kliensek szeretnének valós idejű értesítést kapni [9], ami REST API-n keresztül nem kivitelezhető. Az adatok változásának a valósidejű közvetítésére és fogadására a Szerver a kliensekkel WebSocketen keresztül kommunikál, az org.springframework:spring-websocket és org.springframework.boot:spring-boot-starter-websocket könyvtárakat felhasználva.

Amikor a tagok egy gyűléshez csatlakoznak, automatikusan feliratkoznak a gyűléshez tartozó csatornára, ahol a gyűlés állapotáról kapnak információt. Egy gyűlésnek több állapota lehet, amit a moderátor állít be. Ezeket a mobil kliens valós időben reagálva megjeleníti a tagoknak, minden állapotváltásnál, ami lehet a gyűlés elindítása, napirendi pont váltása, szavazás leállítása vagy gyűlés befejezése. A Szerver értesítést küld a csatlakozott klienseknek a változásról. Továbbá a tagok csatlakozása, kilépése, szavazat leadása esetén egy másik csatornán közvetít információkat a moderátoroknak. Ezáltal pontosan és valós időben nyomom lehet követni a jelenlevők és a leadott szavazatok számát.

A rögzített hanganyagok feldolgozásának is több állapota lehet (lásd később: Átirat kigenerálása). A moderátorok a hanganyag feltöltése után Websocketen keresztül kapnak információt arról, hogy milyen stádiumban van a feldolgozás.

2.1.5. Átirat kigenerálása és szerkesztése

A moderátorok a gyűlésen rögzített hanganyagokból tudnak átiratokat generálni. Erre a felhasznált technológia a Speaches AI, amely OpenAI Whisper alapú mesterséges intelligencia modelleket használ.

Az átiratokat az adatbázis a meeting_minutes táblában tárolja, amelynek tartalma az átirat

szövege, a feldolgozás állapota, és egy külső kulcsos hivatkozás az adott gyűlésre.

A moderátorok a gyűlések befejezése után egy .mp3 vagy .wav kiterjesztésű hanganyagot tölthetnek fel. Kötelezően meg kell adni a hanganyag nyelvét is a precízebb feldolgozás érdekében. A felhasznált modell által támogatott nyelvek listája elérhető REST API-n keresztül. A hanganyagok feldolgozásáért a MeetingMinutesProcessorService osztály a felelős, amely egy várakozási sorban tárolja a feldolgozandó hanganyagokat. Minden sorba kerülő hanganyag Pending (függő) állapotba kerül, és a rendszer a a (2.1.4)-es fejezetbe tárgyalt módon értesítést küld a moderátornak. Ekkor jön létre egy sor a meeting_minutes táblában Pending státusszal és üres szövegmezővel. Mikor a feldolgozásért felelős szolgáltatás kivesz egy hanganyagot a sorból, akkor Processing (feldolgozás alatt) állapotba kerül és a moderátor értesítve lesz. A feldolgozó a hanganyagot elküldi a Speaches AI szolgáltatásnak, ekkor veszi kezdetét az átirat kigenerálása. A generálást követően a MeetingMinutesProcessorService Finished (Befejezett) állapotba helyezi a generált adatot és lementi az adatbázisba, a moderátor ismét értesítve lesz WebSocket-en keresztül. Hiba esetén az alkalmazás egy Error (Hiba) státuszt állít be.

A moderátorok módosíthatják a generált átiratot, ekkor Edited (módosított) állapotba kerül az átirat.

Lehetőség van törölni a generált átiratot, akkor is, ha hiba lépett fel. Ekkor törlődik a meeting_minutes táblából a megfelelő adat és a mentett hanganyag. Ezt követően a moderátoroknak lehetőségük van egy új hanganyag feldolgozására.

Egy átirat nem csak hanganyag feltöltésével jöhet létre, hanem a moderátor megadhatja a szöveget, ekkor automatikusan Edited státuszban mentődik.

2.1.6. Hangfelvételek tárolása

A (2.1.5)-ös fejezetben már szó volt arról, hogy a moderátoroknak lehetőségük van hanganyagokat feltölteni, hogy átiratokat generáltassanak. Ahhoz, hogy a hanganyagokat a későbbiekben el lehessen érni, szükség van ezek tárolására. Erre a célra egy külön fájltároló szolgáltatás van felhasználva, a MinIO, egy nyílt forráskódú objektumtároló rendszer, amely kompatibilis az Amazon S3 (Simple Storage Service) API-jával. A Szerver és a MinIO közti kommunikáció a MinioClient osztály segítségével történik az io.minio:minio könyvtár felhasználásával. A MinioClient osztály implementációja lehetőséget biztosít az alapműveletek elvégzésére: fájlok lementése, törlése, lekérése, u.n. bucket-ek (tárolóegységek) létrehozása, létezésének ellenőrzése.

A hanganyagok információit a MySQL adatbázisba is lementi a rendszer, a meeting_audios táblába, amely tartalmazza a fájl nevét és nyelvét, hogy későbbi műveletekhez hozzáférést lehessen biztosítani a fájlokhoz.

A Szerveren belül a MinIO-val való kommunikációért a MinioService felelős, míg a hanganyagok kezeléséért a MeetingAudiosService. A moderátorok a Szervernek a hanganyagot multipart/form-data formában tudják elküldeni. Ekkor a MeetingAudiosService továbbküldi a fájlt a MinioService-nek, amely lementi azt a megfelelő bucket-be. Minden fórumnak létrejön egy saját tárolója forum-forum_id néven, ezen belül minden gyűlésnek megvan a saját meeting/meeting_id/audio nevű almappája, melyben minden hanganyag random_generált_név.eredeti_kiterjesztés néven van elmentve. Ez lehetővé teszi egy gyűlés esetében több hanganyag elmentését is. Ezután a véletlenszerűen generált névvel és megadott nyelvvel a MeetingAudioService a meeting_audios táblába beszúr egy sort. Ezt követően a mentett hanganyag bármikor elérhető/letölthető. Amikor a moderátor egy átiratot töröl, akkor az adatbázisból a megfelelő sor törlésre kerül és a MinioService segítségével a mentett hanganyag törlődik a gyűlés almappájából.

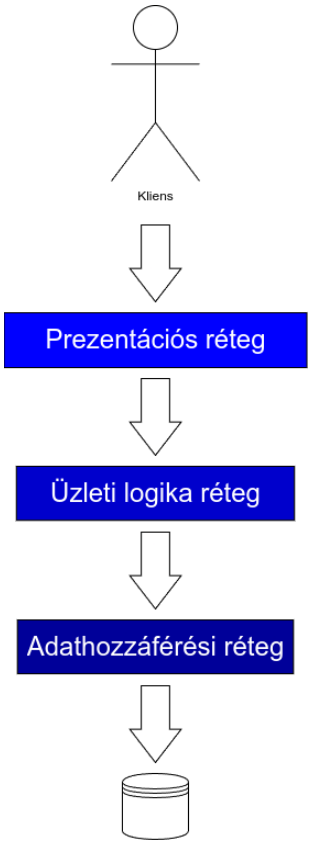
A MinIO-nak valamint az adatbázisnak köszönhetően a Szerver újraindulása esetén a már említett várakozási sor újra felépíthető és folytatható a feldolgozás.

2.2. Szerver architektúra

A 3.ábrán látható a Szerver architektúrája, amely három rétegből áll: Prezentációs-, Üzleti logika- és Adathozzáférési réteg. A funkcionalitások rétegekbe való rendezése lehetővé teszi a komponensek függetlenségét, modularitást biztosítva [10]. Egy réteg implementációja lecserélhető a többi réteg módosítása nélkül. Minden réteg jól meghatározott feladatkörrel rendelkezik. A rétegek közti kommunikáció mindig a Prezentációs rétegtől a Üzleti

```
interface IdentityService {
    fun existsWithEmail(email: String): Boolean
    fun save(identity: Identity): Identity
    fun saveAll(identities: List<Identity>): List<Identity>
    fun getById(id: Long): Identity
    fun getByEmailAddress(email: String): Identity
    fun deleteById(id: Long)
}
```

2. kódrészlet. Példa az Üzleti logika réteg interfészeire



3. ábra. Szerver háromrétegű architektúrája

logika rétegen keresztül az Adathozzáférési rétegig történik. Egy réteg szolgáltatásainak meghatározása Kotlin interfészeken keresztül történik.

A Prezentációs réteg ad lehetőséget a Web- és Mobil klienseknek, hogy kéréseket tudjanak intézni a Szervernek. A VoteHub alkalmazás szervere egy RESTful API-t biztosít, amivel a kliensek HTTP kérésekkel tudnak kommunikálni. Ezeknek az osztályoknak a definíciójánál a @RestController annotáció határozza meg, hogy az adott osztály képes HTTP kéréseket fogadni. A 3. kódrészletben látható a fórummal kapcsolatok kérések kiszolgálására definiált kontroller. A (2.1.4)-es fejezetben már említésre került, hogy a Szerver képes Websocketen keresztül is kommunikálni, valós időben biztosítva bizonyos információkat a kliensek számára.

```
@RestController
class ForumController(
    private val forumService: ForumService,
    private val userRoleService: UserRoleService,
    private val forumMapper: ForumMapper,
    private val votingSystemMapper: VotingSystemMapper,
)
```

3. kódrészlet. Példa RestController létrehozására és függőség injektálás

A Üzleti logika réteg határozza meg a Szerver teljes funkcionalitását. A RestControllerek a kérés fogadását követően a kérést továbbítják a megfelelő Service osztálynak. A 2.kódrészletben látható egy Üzleti logika rétegben levő interfész. Az osztályok, amelyek megvalósítják ezeket az interfészeket a @Service annotációval rendelkeznek.

Az Adathozzáférési rétegben elhelyezkedő osztályok a MySQL adatbázissal létesítenek kapcsolatot. A (2.1.2)-es fejezetben már tárgyalt JpaRepository interfészt öröklő interfészek a @Repository annotációval vannak ellátva.

2.3. Használt technológiák

A VoteHub alkalmazás szervere Kotlin-ban írodott a Spring keretrendszer felhasználásával. A Kotlin forráskód a Java nyelvhez hasonlóan JVM bájtkódra fordítódik. Ennek köszönhető az átjárhatóság a Kotlin és a Java nyelvek között, így a Java nyelvben fejlesztett függvénykönyvtárak és keretrendszerek felhasználhatóak.

Az adatbázis kapcsolat létrehozása a Spring Boot Starter JDBC segítségével történik. Az adatbázis táblák és a modell osztályok közötti leképezést a Spring Data JPA valósítja meg. A Spring Boot Starter Web segítségével van konfigurálva Spring Web modulja, így lesz a szerver képes HTTP kéréseket fogadni és válaszolni ezekre. Az adatok helyességének ellenőrzése a Bean Validation technológia segítségével történik, a Spring Boot Starter Validation konfigurálja a szükséges függőségeket. Az erőforrások biztonságáért a Spring Security, Spring Security OAuth2 Resource Server és Spring Security OAuth2 Client könyvtárak felelnek. A kliensektől érkező JWT tokenek validálására a Spring Security OAuth2 Jose könytárban levő Nimbus JWT dekóder felelős. A szerver által küldött HTTP kérések az OkHttp3 könyvtár segítségével történnek.

A kód fordítása és futtatása gradle segítségével történt [3]. A forráskód egységességének és olvashatóságának biztosítása érdekében a Ktlint statikus kódelemző eszköz került alkalmazásra.

3. Web alkalmazás

A web alkalmazás elsődleges célja, hogy adminisztrációs felületet biztosítson a moderátorok részére, illetve lehetővé tegye a fórumok és a hozzájuk tartozó tartalmak megjelenítését a fórum tagok számára. A fejezetben részletezve lesznek a webfelület főbb funkcionalitásai, a platform architektúrája, illetve a fejlesztés során felhasznált technológiák. A fejezet végén ismertetve lesz a felhasználói felület elkészítésének folyamata.

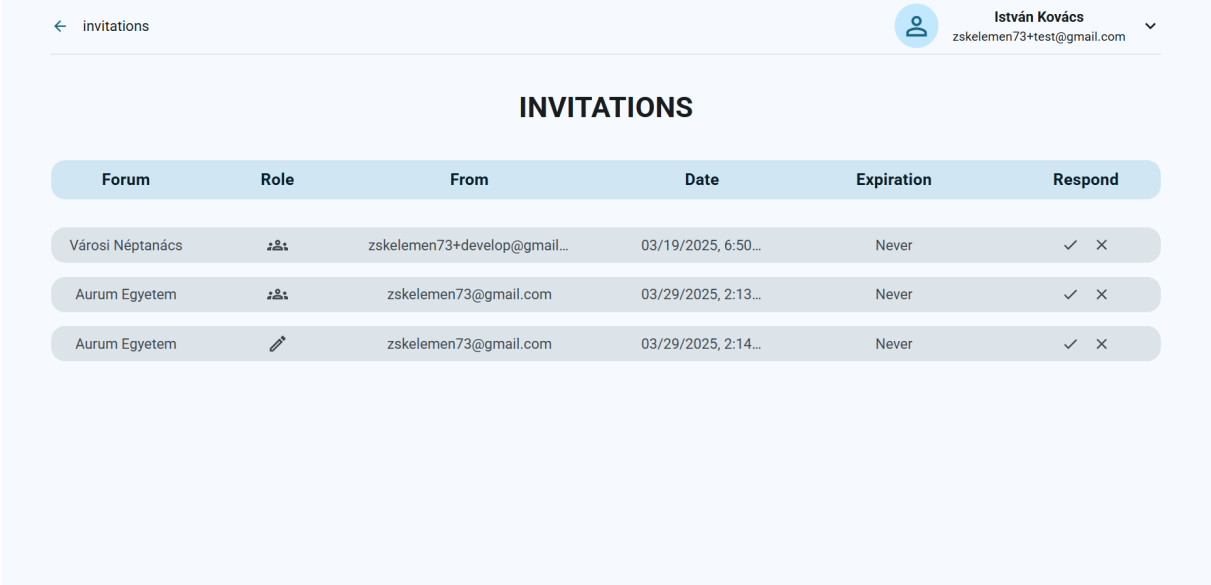
3.1. Funkcionalitások

Mivel a web alkalmazás főként a moderátorok számára készült, ezért az itt bemutatott funkcionalitások egy része csak számukra érhető el, nem hozzáférhető a fórum tagok számára.

3.1.1. Identitás és hozzáférés kezelés

A weboldal első megnyitásakor a felhasználó, egy ismertető oldalt lát, ahol a Bejelentkezés gombra kattintva, az alkalmazás átirányítja őt a bejelentkezési oldalra, ahol lehetősége van regisztrálásra is. Regisztráláskor a felhasználónak igazolnia kell az e-mail címét. Ezeket a folyamatokat a Keycloak szerver végzi. Sikeres bejelentkezés esetén a Keycloak szerver generál egy token-t, majd a felhasználót visszairányítja a VoteHub oldalára.

A Keycloak szerverrel történő OIDC protokollt követő kommunikáció a React OIDC Context könyvtár segítségével lett megvalósítva. Annak érdekében, hogy a könyvtár által nyújtott funkcionalitások elérhetővé váljanak, az alkalmazás komponensei be kell legyenek ágyazva az AuthProvider nevű komponensbe. Ez által elérhetővé válik a signinRedirect függvény, melynek meghívásával elindítható a fent leírt autentikációs folyamat. Bejelentkezés után a felhasználó, illetve a Keycloak által visszaadott JWT tokenek (accessToken, refreshToken), a böngésző lokális tárhelyén lesznek eltárolva. Amíg ezek érvényesek, a felhasználónak nem kell minden alkalommal újra bejelentkeznie. Amennyiben a felhasználó valamilyen védett erőforráshoz szeretne hozzáférni, az alkalmazás a szerverhez intézett kérés fejlécéhez automatikusan hozzáfűzi az accessToken-t, mint Bearer token. Ezt a szerver ellenőrzi, és csak akkor küldi el a megfelelő erőforrást, ha a token érvényes.



4. ábra. A meghívókat listázó oldal

3.1.2. Meghívórendszer

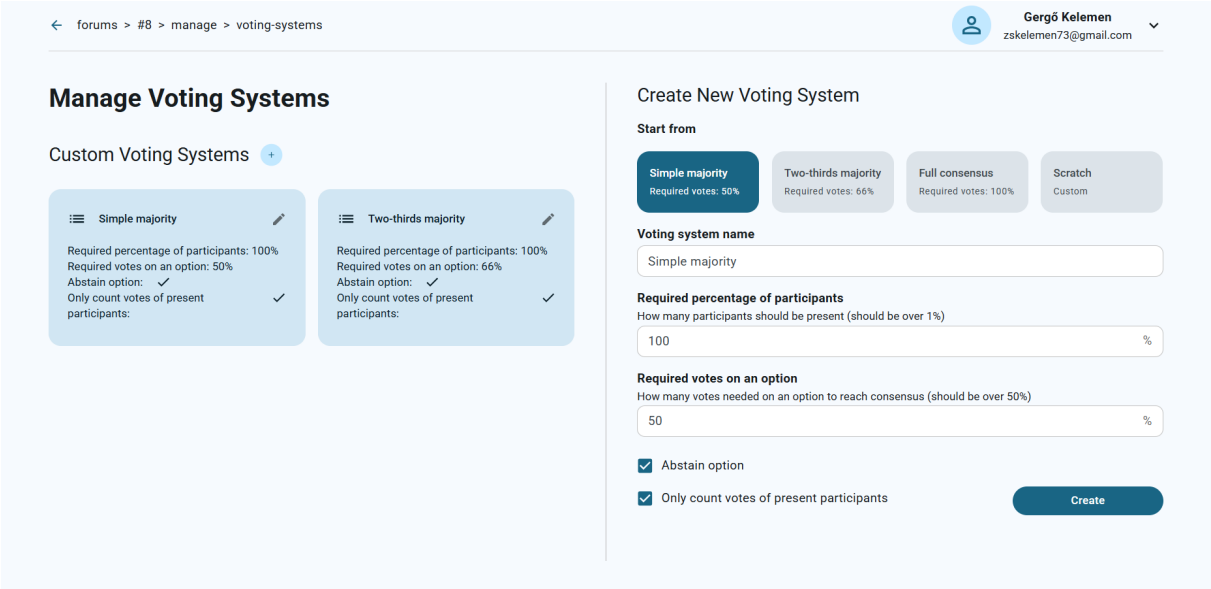
Ha egy moderátor meg szeretne hívni valakit egy fórumba, akkor azt az adott fórum oldalán teheti meg. Az oldalon kiválaszthatja, hogy moderátorként, vagy tagként szeretne meghívni valakit, majd beírva az email címét az illetőnek, elküldheti a meghívót. Ezt követően a meghívott személy neve megjelenik a tagok listájában, egy ikonnal jelezve, hogy még el kell fogadnia a meghívót. Ha a meghívott személy még nem regisztrált az oldalon, akkor a tagok listájában, a felhasználó neve helyett csak az e-mail címe látható.

A bejelentkezett felhasználó a meghívóit a jobb felső sarokban található menüponton keresztül érheti el. Itt, ahogyan azt a 4. ábra is mutatja, táblázat formájában jelenik meg, hogy melyik fórumba hívták meg, milyen jogosultságokat kap, ki hívta meg, illetve, hogy mikor kapta a meghívót. A táblázat utolsó oszlopában a megfelelő gombra kattintva tudja elfogadni, vagy elutasítani a meghívót.

3.1.3. Fórumon belüli jogosultságok kezelése

A moderátoroknak lehetősége van, az egyes fórum tagok szavazati jogának a felfüggesztésére. Ez a megvonás két szinten történhet:

- Fórum szinten – Az adott felhasználó a fórumon belül zajló gyűlések egyikén sem szavazhat amíg vissza nem kapja a szavazati jogát. Ezt a moderátor az adott fórum oldalán állíthatja be, a tagok listájában, a felhasználó neve mellett levő kalapács ikonra kattintva.



5. ábra. A szavazati rendszereket kezelő oldal

- Napirendi pont szinten – Az adott felhasználó, egy gyűlésen a megjelölt napirendi pontokról nem szavazhat, de a többi napirendi pontról igen. Ezt a moderátor az adott napirendi pont oldalán állíthatja be, a résztvevők listájában a megfelelő felhasználó nevére kattintva.

Azok a felhasználók, akiknek a szavazati jogukat megvonták, részt vehetnek a gyűlésen, de nem jelenik meg számukra a lehetőség, hogy szavazzanak és nem számolódnak bele a jelenlévők listájába eredményszámításkor.

3.1.4. Szavazati rendszer testre szabása

Az alkalmazásban a moderátorok testre szabhatják, hogy az egyes napirendi pontokon a szavazás milyen formában történjen, azaz hogyan számolódjon a végeredmény. Ezt saját szavazati rendszerek létrehozásával tehetik meg, amelyre egy külön oldal áll rendelkezésükre a webfelületen. A szavazati rendszer létrehozásakor kiválaszthatják, hogy milyen sablon (egyszerű többség, minősített többség, teljes konszenzus) alapján szeretnék létrehozni a rendszert, vagy választhatnak teljesen egyedi beállításokat is. Egyedi szavazati rendszer létrehozásakor a következő adatokat kell megadják: a szavazati rendszer nevét, egy százalékot, ami azt mutatja, hogy a fórum teljes tagságából hányan kell jelen legyenek, hogy érvényes legyen a szavazás, illetve, hogy egy adott választási lehetőségre hány százaléka kell a jelen levőknek szavazzon ahhoz, hogy az nyerje meg a szavazatot. Emellett az is kiválasztható, hogy az adott szavazási rendszer tartalmaz-e tartózkodom opciót.

A VoteHub testreszabhatóságot biztosít a szavazásokon megadható döntési lehetőségek szintjén is, ugyanis elképzelhető, hogy egy adott napirendi ponton nem csak Igen-el vagy Nem-el lehet szavazni. A moderátorok fórum szinten definiálhatnak szavazási lehetőségeket, melyek közül majd kiválasztják, hogy az egyes napirendi pontokhoz melyik lehetőségek tartoznak.

Amikor egy moderátor létrehoz egy gyűlést, meg kell adnia a napirendi pontokat, melyekről a fórum tagjai szavaznak a gyűlés során. Ezek után, az egyes napirendi pontoknál, ki kell választania, hogy az adott napirendi pont esetében milyen szavazati rendszer alapján számolódik a végeredmény, illetve, hogy milyen választási lehetőségekre lehet szavazni majd. Amennyiben a kiválasztott szavazati rendszer megköveteli, hogy legyen lehetőség tartózkodni, a moderátornak azt is ki kell választani, hogy a napirendi ponthoz tartozó választási lehetőségek közül melyik jelenti a tartózkodást, ezt a rendszer majd semleges választásnak tekinti az eredmények számításakor.

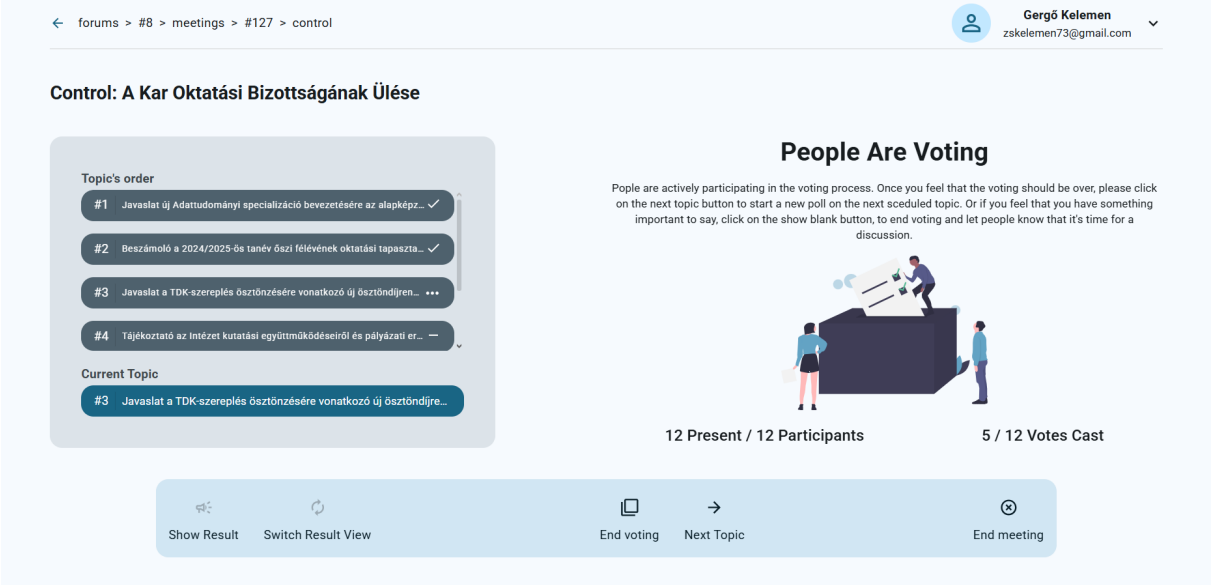
Amennyiben a fent említett beállítások közül akár egy is hiányzik egy gyűlés valamelyik napirendi pontján, a gyűlés nem indítható el, és ezt a rendszer az adott napirendi pontnál jelzi, mint konfigurációs hiba. A rendszer azon esetben is konfigurációs hibát mutat, amennyiben valamely napirendi pont szavazati rendszere megköveteli, hogy egy bizonyos százaléka a fórum tagoknak jelen legyen szavazáskor, viszont ez a százalék elérhetetlen. Az alkalmazás ebben az esetben is jelzi a moderátorok számára, hogy a gyűlésben beállítási hibák vannak, és amíg nem oldódnak meg, a gyűlés nem lesz elindítható. Ez az eset akkor következhet be, ha a moderátorok túl sok fórum tagtól megvonták a szavazati jogot, és így nem maradt elegendő szavzatra képes fórum tag ahhoz, hogy a szavazati rendszer által előírt százalék elérhető legyen, még teljes létszám esetén sem.

3.1.5. Gyűlések lebonyolítása

A gyűlések levezetésére egy külön felület áll a moderátorok rendelkezésére. A gyűlést egyetlen moderátor tudja levezetni, ami annak elindításától, a lezárásáig tart. Amíg a gyűlés nincs elindítva a moderátor ezen az oldalon csak egy szöveget lát, amely tudtára adja, hogy a gyűlés még nem indult el. Ezalatt a szöveg alatt látható a start gomb, amivel a gyűlés elindítható.

A start gombra kattintva a gyűlés elindul, és megjelenik a gyűlés vezérlő felülete, amely a következő három részből áll:

- A képernyő bal oldalán egy lista, amelyben a gyűlés napirendi pontjai vannak kilistázva,



6. ábra. A gyűlés oldala

ezalatt pedig az éppen tárgyalás alatt álló napirendi pont, amennyiben van ilyen. A listában az is fel van tüntetve, hogy melyek azok a napirendi pontok, melyeket már megszavaztak, illetve melyek azok, amelyek még megszavazandóak.

- A képernyő alján, található maga a vezérlő felület, ahol a különböző gombokra kattintva, a moderátor elindíthatja, leállíthatja a szavazást, vagy megtekintheti egy szavazás eredményeit.
- A képernyő jobb oldalán pedig különböző információk láthatóak a gyűlésről az alapján, hogy a vezérlő felületen milyen műveletet hajtott végre a moderátor. Például, itt jelennek meg a szavazatok eredményei is.

A gyűlés ideje alatt a webalkalmazás a REST API kommunikáción kívül egy STOMP [1] protokollt követő WebSocket alapú kapcsolatot is fenntart a szerverrel. Ennek köszönhetően a moderátor követni tudja, hogy jelen pillanatban hányan vannak jelen a gyűlésen a teljes tagságból. A szavazás elindítása után az is láthatóvá válik számára, hogy az adott pillanatig hányan adták le a szavazatukat. Ezeknek a valósidejű információknak az alapján a moderátor indíthat vagy lezárhat szavazásokat. A websocket kommunikáció megvalósításához a STOMP.js JavaScript könyvtárat használja a projekt, mely egy kliens osztályt biztosít a STOMP alapú kommunikációhoz. Az alkalmazás megfelelő hibaüzenettel jelzi ha megszakadt a WebSocket

[1] Simple Text Oriented Message Protocol

kapcsolat a kliens és a szerver között. Ebben az estben a kliens másodpercenként megpróbál újra csatlakozni, és ha sikeres, ismét jelez a felhasználónak.

Az alkalmazás két különböző nézetet biztosít a szavazás eredményének a megjelenítésére. Listázható egyenként, hogy a szavazók közül ki mire szavazott, vagy megtekinthető az eredmény összesítve egy kördiagramon. A kördiagram egy a Material UI könyvtárból származó PieChart nevű komponens, melynek az adatai dinamikusan állíthatóak.

3.1.6. Átirat feltöltése és kigenerálása

Egy lezárt gyűléshez a moderátorok társíthatnak egy átiratot a gyűlésen elhangzottakról az erre kijelölt oldalon. Az átiratot kétféleképpen vezethetik be a rendszerbe: vagy bemásolják az átirat szövegét az erre szánt mezőbe és lementik, vagy feltöltenek egy hang fájlt-t és a rendszer AI technológia segítségével automatikusan létrehozza az átiratot.

A fájl feltöltése egy drag and drop komponens használatával lett megvalósítva, amely csak mp3 vagy wav kiterjsztésű file-okat fogad el. A hang file-t egy olyan HTTP kérés formájában küldi el a web alkalmazás a szervernek, melynek a teste multipart/form-data típusú. Amennyiben a szerver sikeresen fogadta a hang file-t, egy websocket kapcsolat létesül a webalkalmazás és a szerver közt, amely szintén a STOMP protokollt követi. A WebSocket kapcsolat azért szükséges, hogy a moderátorok folyamatos visszajelzést kapjanak a feldolgozásban levő fájl jelenlegi állapotáról a webfelületen.

Miután a moderátor sikeresen bevezette a rendszerbe az átiratot, lehetősége van módosítani azt (ez kifejezetten az AI által generált átiratoknál fontos, mivel ezek esetében könnyebben előfordulhatnak hibák).

Az átiratok HTML formátumban vannak eltárolva, így lehetőség van a szöveg stilizálására is, címek, alcímek, listák beszúrásával. Ennek érdekében, amikor egy moderátor kézzel írja be az átirat szövegét vagy módosítja azt, egy rich text szerkesztőben dolgozik. Az alkalmazás a TipTap nevű rich text szerkesztőt használja.

Az átiratoknak HTML formátumban való tárolása azért is hasznos, mivel így megjeleníthető az átirat tartalma a weboldalon. React-ban lehetőség van külső forrásból származó HTML-t beszúrni, mint egy komponens gyermeke a dangerouslySetInnerHTML property segítségével. Viszont ez az eljárás önmagában nem biztonságos mivel rést hagy a cross-site-scripting (XSS) támadások számára, hiszen a beszúrt HTML tartalmazhat JavaScript kódot is, amelyet a böngésző lefuttathat. Ennek elkerülése érdekében a beérkező HTML-t meg kell “tisztítani”.

Erre a célra lett használva a DOMPurify könyvtár, amely jól karbantartott, és egy biztonságos megoldást nyújt a külső forrásból érkező HTML dokumentumok megtisztítására [6].

3.2. Architektúra

A webfelület felépítése a React által előírt komponens alapú architektúrát követi, amely a komponenseket úgy határozza meg, mint a rendszer funkcionalitásainak jól elhatárolt és jól meghatározott részei. Ennek megfelelően a React minden vizuális elemet egy komponensként kezel, amelyek egymásba ágyazhatóak. Tehát valójában az egész alkalmazás egy komponens, melynek alkomponensei a különböző oldalak a webfelületen, amelyek szintén tovább bonthatóak kisebb részkomponensekre, mint például a listák, form-ok, és egyéb, a funkcionalitás szempontjából jól elhatárolható elemek. Egyik előnye a komponens alapú architektúrának a komponensek újrahasználhatóságában rejlik, így a meglévő funkcionalitások könnyen bővíthetőek, és változtathatóak.

Az alkalmazás egy single page webalkalmazás (SPA), vagyis miután a React kiépítette az alkalmazást, a weboldal egyetlen HTML file-ból áll. Annak érdekében, hogy a felhasználó ezt ne érzékelje, az alkalmazás a React Router könyvtárat használja, ami virtuális oldalakat hoz létre az alkalmazáson belül és megteremti a köztük levő navigációt. A könyvtár nyújtotta funkcionalitások használatának érdekében, az alkalmazás oldalait egy router nevű objektumban kell definiálnunk, amely alapján a RouterProvider nevű komponens képes a gyerek komponenseit olyan formában kirajzolni, hogy több oldal látszatát keltse. Ennek köszönhetően egyetlen helyen van meghatározva az alkalmazás oldalainak a szerkezete.

A főbb entitások, mint például a fórumok, gyűlések, JavaScript objektumokként vannak reprezentálva. Ezeknek az objektumoknak a típusa TypeScript típusok segítségével van megadva, amelyre példát a 4. kódrészletben láthatunk. A szervernek küldött, és a szervertől érkező adatok is ilyen formában vannak reprezentálva. Ezek a típus deklarációk a rendszer

```
type ForumT = {
    id: number;
    name: string;
    description: string;
    active: boolean;
};
```

4. kódrészlet. Példa TypeScipt típus meghatározásra

modelljeiként viselkednek, amelyek megadják, hogy a főbb entitások milyen attribútumokkal kell rendelkezzenek.

React alkalmazásokban fontos problémát jelent a komponensek közös állapotának a kezelése, azaz hogyan osztjuk meg az olyan információkat, amely több komponenst is érint. Erre a célra lett kifejlesztve a Redux Toolkit [7] könyvtár, amelyet a webalklamazás is használ.

A Redux megközelítése alapján, az alkalmazás megosztott állapota egy úgynevezett store-ban van eltárolva. Azok a komponensek, amelyek el akarják érni ezt az információt egy Provider nevű komponens gyerekei kell legyenek. A store adatait úgynevezett reducer függvényekkel módosíthatjuk.

A Redux előírja, hogy egy alkalmazáson belül csak egy store lehet, viszont általában a komponensek a benne levő összes információnak csak jól elkülöníthető részein osztoznak egyszerre, azaz nincs olyan komponens, amelyet a store teljes tartalma érdekelne. Emiatt a Redux Toolkit a megosztott állapotot módosító reducer függvényeket úgynevezett szeletekbe (slice) rendezi. Más szóval egy szeleten belül olyan logika található, ami a megosztott állapot egy jól elkülöníthető részét módosítja. Erre példát láthatunk az 5. kódrészletben, ahol az alkalmazáson belüli felugró ablakokra vonatkozó információkat módosító szeletet láthatjuk. Ez az információ tulajdonképpen nem más, mint egy igaz-hamis érték, amely azt jelzi, hogy az adott ablak nyitva van-e vagy sem.

Egy komponens egy speciális kiválasztó függvény segítségével képes hozzáférni a store-ban tárolt adatokhoz. A függvény segítségével a komponens megadja, hogy a megosztott állapot

```
const popupSlice = createSlice({
    name: "popups",
    initialState,
    reducers: {
        modifyCreateForumPopup: (state, action: PayloadAction<boolean>) => {
            state.createForumPopupOpen = action.payload;
        },
        modifyCreateMeetingPopup: (state, action: PayloadAction<boolean>) => {
            state.createMeetingPopupOpen = action.payload;
        },
        modifyCreateTopicPopup: (state, action: PayloadAction<boolean>) => {
            state.createTopicPopupOpen = action.payload;
        },
    },
});
```

5. kódrészlet. Példa egy szelet működésére

mely része érdekli őt, és amennyiben az megváltozik, a Redux gondoskodik arról, hogy a komponens újra rajzolódjon. Ez a függvény automatikusan elérhetővé válik minden olyan komponensben, amely a fent említett Provider komponens gyermeke.

Gyakori példája a komponensek közt megosztott információnak az olyan információ, amely egy külső szervertől érkezik. Az ilyen jellegű adatok kezelése általában bonyolultabb, mivel a kommunikáció aszinkron módon történik. Erre a célra alkalmazható az RTK Query (Redux Toolkit Query) eszköz, amely alapértelmezetten része a Redux Toolkit könyvtárnak. Segítségével egy sajátos szeletet hozhatunk létre, amelyben egy külső API végpontjaira érkező kéréseket írhatunk. Kétféle kérést különít el az RTK Query: lekérdezés és mutáció. A lekérdezés típusú kérések az adatok lekérésére használtak, és általában egy GET HTTP kérést kezdeményeznek. A mutáció típusú kérések adatmódosítóak, és általában POST, PUT, DELETE vagy akár más HTTP kéréseket kezdeményeznek. A 6. kódrészletben láthatunk példát arra hogyan kell megadni a végpontokat az RTK query számára. A példában látható végpontok a meghívókat kezelik.

A külső szervertől érkező adatokat más módon érik el a komponensek, mint ahogyan az idáig volt bemutatva. Az RTK Query minden általunk definiált kérésnek saját React hook-okat hoz létre, amelyeket a komponensek csak meghívnak, hogy lefuttassák a kérést. Ezek a hook-ok azért hasznosak, mert nem csak a kérést indítják el, hanem visszaadják a kérés eredményét is, illetve egyéb változókat is nyújtanak, melyek a kérés állapotáról

```
const invitationsApiSlice = apiSlice.injectEndpoints({
    endpoints: (builder) => ({
        getInvitations: builder.query<InvitationT[], void({
            query: () => `/users/me/invitations`,
            providesTags: ["Invitations"],
        }),
        acceptInvitation: builder.mutation<void, { id: number }>({
            query: ({ id }) => ({
                url: `/users/me/invitations/${id}`,
                method: "PUT",
                body: { status: invitationConstants.userStatus.ACCEPTED },
            }),
            invalidatesTags: ["Invitations"],
        }),
    }),
});
```

6. kódrészlet. Példa egy külső szerverrel kommunikáló szeletre

adnak információt, mint például isFetching, isSuccess, isError. Ezek alapján a komponensek dinamikusan változtathatóak annak függvényében, hogy a kérés milyen állapotban van.

Az RTK Query emellett cache-elési technikákat is alkalmaz, amely jelentősen gyorsítja az alkalmazást. A cache úgynevezett címkék (tag) segítségével van megvalósítva. A lekérdezés típusú kéréseknek megadhatjuk, hogy milyen címkékre figyeljenek, a mutációknak pedig, hogy milyen címkéket kell érvényteleníteniük. Egy mutáció típusú kérés meghívásakor, a mutáció címkéi érvénytelenek lesznek. Ha egy címke érvénytelen, akkor az összes, arra a címkére figyelő lekérdezés újra lefut, mivel ez azt jelenti, hogy a lekérdezés által szolgáltatott adatok már elavultak. Amíg egy lekérdezés adatai érvényesek a Redux a lementett adatokat használja, és így elkerüli a fölösleges API hívásokat. A címkék megadása szintén a 6. kódrészletben látható.

3.3. Felhasznált technológiák

A VoteHub webalkalmazás TypeScript-ben íródott, amely egy kibővítése a JavaScript nyelvnek. A TypeSrcipt típusokkal egészíti ki a JavaScript-et egy erősen típusos nyelvé, így lehetőségünk adódik minden változónak, objektumnak és függvénynek megadni a típusát, amelyek lehetnek akár saját vagy összetett típusok is. A TypeScript fordító egy tsconfig fájl segítségével konfigurálható.

Ahogyan azt a korábbiakban említettük, az alkalmazás React segítségével lett megírva, és több, a React ökoszisztémához tartozó könyvtár is használva volt a fejlesztés során, mint a React Router, Redux Toolkit és RTK Query.

A felhasználói felület vizuális elemeit a Material UI komponens könyvtár szolgáltatta, amely elsősorban React alkalmazások fejlesztésére volt kitalálva.

A fejlesztés során statikus kódelemző eszközök biztosították a kód egységességét. Az egyik ilyen felhasznált eszköz az ESLint, amely főként a gyakori JavaScript és TypeScript hibákat érzékeli és jelzi a fejlesztő számára. A legtöbb feljesztői környezettel könnyen integrálható, így az ESLint által generált jelzések valós időben vizuálisan is megjelennek a kódszerkesztőben. Szintén használva volt a Prettier, amely főként a kód olvashatóságát, megfelelő szerkezetét ellenőrzi, és hasonló módon integrálható a legtöbb fejlesztői környezetbe, mint az ESLint.

Az alkalmazás kiépítésére a Vite eszköz volt használva. A Vite fejlesztés alatt egy lokális szervert is futtat, amelyen élő újtatöltéssel lehetséges a fejlesztés.

A fent említett technológiák mellett, további JavaScript könyvtárak is használva voltak, ezek közül néhányat már ismertettünk a (3.1)-es fejezetben. Ezek a könyvtárak az npm csomagkezelő

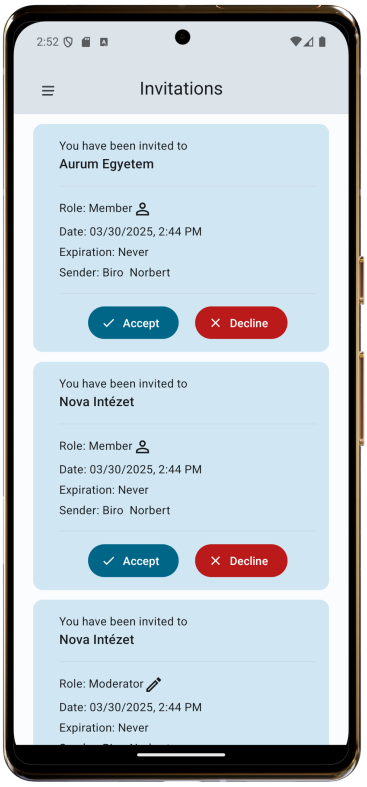
segítségével lettek bevezetve az alkalmazásba.

3.4. A vizuális elemek megtervezése

A webfelület oldalainak többsége először Figma-ban került megtervezésre és csak ezt követően került lefejlesztésre. A felület a Google Material Design 3 elvei alapján lett megtervezve. A látványterv elkészítésekor a Material 3 Design Kit Figma könyvtár szolgáltatta a különböző komponenseket.

Az alkalmazás fejlesztése során a Material UI komponens könyvtár volt használva, amely a Material Design 2 irányelveit követi, viszont a könyvtár kiemelkedő testreszabhatósága lehetővé tette, hogy egyetlen konfigurációs file segítségével az alkalmazás összes komponense a Material Design 3 irányelveinek feleljen meg.

Az alkalmazásban látható statikus svg-k az unDraw weboldalról származnak. Az alkalmazás logóját pedig a dolgozat írói készítették, a Gimp nevű alkalmazásban.



7. ábra. Meghívók mobil alkalmazáson belül

4. Mobil alkalmazás

A mobil alkalmazás elsődleges célja, hogy a tagok számára biztosít egy felületet szavazások lebonyolítására. A fejezet célja a mobil alkalmazás főbb funkcionalitásainak, architektúrájának és a fejlesztés során használt technológiáknak a bemutatása.

4.1. Funkcionalitások

A mobil alkalmazás elsősorban a tagok igényeit hívatott kiszolgálni, de a funkcionalitások nagy része a moderátorok számára is elérhető.

4.1.1. Meghívó rendszer

A web alkalmazáshoz hasonlóan a tagoknak és a moderátoroknak egyaránt lehetőségük van a fórum meghívók elutasítására vagy elfogadására. Az alkalmazás egy különálló oldalt biztosít a meghívók kezelésére, mely az oldalsó menüben a Meghívók szekcióban elérhető, ezt kiválasztva az alkalmazás átirányít a dedikált oldalra. A 7. ábrán a meghívók kezelésére szolgáló felület látható. A mobil alkalmazás a felhasználó meghívóit a szervertől REST API-n keresztül kéri le. A meghívók megjelenítésére az alkalmazás a React-Native-Paper által biztosított

8. ábra. Szavazás egy napirendi pontra

Card komponenst használja. Minden kártya tartalmazza a (3.1.2)-es fejezetben már tárgyalt információkat. A kártya alján található két gomb segítségével lehet elfogadni vagy elutasítani a meghívót.

4.1.2. Napi gyűlések

A VoteHub alkalmazás a gyűléseket fórumokba szervezi, ezek az oldalsó menüben érhetőek el. A felhasználók egy fórumot kiválasztva megtekinthetik a gyűlések listáját, amely elkülöníti a befejezett gyűléseket a még el nem kezdett vagy elkezdett gyűlésektől. Külön dedikált oldal listázza a napi gyűléseket, mivel az esetek többségében egy felhasználó aktuális gyűlésen szeretne részt venni az alkalmazás használatakor, ezért a bejelentkezést követően az alkalmazás automatikusan átirányítja erre az oldalra. Az oldalsó menüből is elérhető ez a nézet.

Kiválasztva egy elemet a napi gyűlések oldalán megjelenő listából, a felhasználó a gyűlés aktuális állapotáról kap információ. Az oldal a megfelelő üzenettel jelzi amennyiben nincs az adott napon elkezdett vagy az adott napra tervezett gyűlés.

4.1.3. Szavazás lebonyolítása

A napirendi pontokra való szavazás a Mobil alkalmazás egyetlen olyan funkcionalitása, ami csak a tagok számára elérhető. Abban az esetben, ha egy moderátor megpróbál belépni, a rendszer egy megfelelő hibaüzenettel válaszol.

Miután egy tag belépett egy gyűlésre, a mobil alkalmazás REST API-n keresztül lekérdezi a szervertől a gyűlés aktuális állapotát és rácsatlakozik a gyűlés számára fenntartott WebSocket kapcsolatra, hogy a további állapotváltozásokról valós időben értesüljön a kliens a (3.1.5)-ös fejezetben tárgyalt protokollt alkalmazva.

Egy gyűlésnek négy állapota lehet:

1. Ütemezett gyűlés - a moderátor még nem kezdte el a gyűlést
2. Elkezdett gyűlés - megszavazandó napirendi pont nélkül
3. Elkezdett gyűlés - megszavazandó napirendi ponttal
4. Befejezett gyűlés

A mobil alkalmazás a fent említett állapotokat a megfelelő üzenet megjelenítésével jelzi a felhasználóknak. A 8. ábrán látható egy napirendi pont szavazása. Amikor a moderátor elindítja a szavazást egy napirendi pontra, akkor a gyűlés Websocket csatornáján értesítve lesznek a felhasználók. A megjelenített szavazókártya tartalmazza a napirendi pont címét, leírását, a választható opciók listáját és egy küldés gombot, amivel a felhasználók jóvá tudják hagyni a szavazatukat. Alapértelmezetten nincs kiválasztva egy opció sem és a küldés gomb is le van tiltva. Az opciók listájából a szavazó csak egyetlen lehetőséget tud kiválasztani. Miután a választás megtörtént a küldés gomb aktívvá válik, és a választott opcióra le lehet adni a szavazatot. A küldés gombot megnyomva a mobil kliens a szavazatot REST API-n keresztül elküldi a szervernek és WebSocket-en keresztül értesíti a moderátort, hogy a felhasználó szavazott. Ezt követően a küldés gomb ismét inaktívvá válik és egy jóváhagyó üzenet jelenik meg a szavazókártya alatt. Egy felhasználó csak egyszer szavazhat egy napirendi pontra.

4.2. Architektúra

Mivel a mobil alkalmazás React Native keretrendszerben íródott, ezért a webes alkalmazáshoz hasonlóan, komponens alapú architektúrát követ. A főbb entitások is JavaScript objektumokként vannak reprezentálva, melyeknek típusa TypeScript segítségével van megadva.

Architektúra szempontjából lényeges eltérés a webes alkalmazáshoz képest az oldalak elrendezése és a köztük történő navigáció. Ennek a megvalósítására az Expo Router keretrendszer van használva, amely egy fájl alapú megközelítést nyújt a mobil alkalmazás oldalainak elrendezésére. Ennek alapján a különböző oldalaknak megfelelő komponenseket külön mappákba kell elhelyezni. Minden ilyen mappának tartalmaznia kell egy _layout.ts nevű fájlt, amelyben magának az oldalnak megfelelő főkomponens található. Természetesen, ez a komponens állhat több részkomponensből, melyek nem feltétlenül kell közös mappában legyenek az oldalnak megfelelő komponenssel. Az oldalak közt történő navigáció, a megfelelő oldalt tartalmazó mappa nevének segítségével történik, amely alapján a keretrendszer megállapítja, hogy melyik oldalt kell betöltenie. Átirányításkor paraméterek is megadhatóak, hogy a betöltött oldal, dinamikusan változtathassa adatait.

A mobil alkalmazáson belül a komponensek osztott állapotának a kezelésére a React Context API van használva a Redux könyvtár helyett. A Context API kevesebb konfigurációt igényel, és emiatt kisebb méretű projektek esetén gyakran használt. A Context API segítségével saját, úgynevezett kontextusokat hozhatunk létre, amelyek tartalmazhatnak különböző adatokat, illetve függvényeket amelyek segítségével ezek az adatok módosíthatóak. Ezeket az információk és függvények egy speciális useContext React hook segítségével érhetőek el az egyes komponensekben. Ilyen kontextusokban helyezhetőek el a komponensek megosztott adatai.

4.3. Felhasznált Technológiák

A mobil alkalmazás a React Native keretrendszerben íródott, amely cross-platform mobil applikációk fejlesztésére ad lehetőséget. A React Natív egy JavaScriptre épülő keretrendszer, és támogatja a TypeScript használatát is, ezért a webes felülethez hasonlóan a mobil alkalmazás is TypeScriptben lett megírva.

Az alkalmazás felépítése Expo segítségével történik, ami egyéb olyan eszközöket is magába foglal, amelyek React Native fejlesztéshez szükségesek. Ezek közül a (4.2)-es fejezetben említett Expo Router is használva volt.

A vizuális felület elemeit a React Native Paper könyvtár szolgáltatta. A webes alkalmazáshoz hasonlóan, az ESLint [13] és a Prettier statikus kódelemző eszközök biztosították a kód egységességét, a külső függőségek kezelésére pedig az npm csomagkezelő rendszer volt használva.

5. DevOps

Az alkalmazás fejlesztése során prioritás volt az alkalmazás folytonos integrációja és kitelepítése. Az automatizált alkalmazás kitelepítés lehetővé teszi az ismétlődő és automatikus alkalmazás frissítést. A projekt szervere és web alkalmazása a Codespring által biztosított szerverre van kitelepítve Docker konténerek segítségével. A szoftver verziókövetésére Git van használva és GitLab a kódbázis központi kezelésére és a fejlesztési folyamatok automatizálására.

5.1. GitLab CI/CD

A GitLab a verziókövetés mellett, CI/CD lehetőségeket is nyújt a GitLab pipeline-ok segítségével [2]. Ezekben a pipeline-okban olyan feladatokat határozhatunk meg, amelyek automatikusan lefutnak, amennyiben új verziók kerülnek a GitLab tárolóba. A Szerver, Web és Mobil alkalmazások pipeline-jai a következő két feladatot tartalmazzák:

- Lint feladat - amely a statikus kódelemzőket futtatja.
- Build feladat - amely az alkalmazások felépíthetőségét ellenőrzi a megfelelő build eszközök futtatásával.

Ezek mellett a szerver és web alkalmazás pipeline-jai tartalmaznak teszteléshez és kitelepítéshez tartozó feladatokat is.

A tesztelési folyamatok minden új verzió esetén lefutnak. Célja, hogy ellenőrizze az alkalmazások működésének helyességét és stabilitását. Szerver tesztelése egy memóriában futó H2 adatbázissal történik, amely egy izolált környezetet biztosít, elkerülve a külső adatbázissal való kommunikációt. Az egységtesztelés során a rendszer komponensei a Mockito és MockK könyvtárak segítségével vannak szimulálva. A web alkalmazás teszteléséhez a Vitest keretrendszer, és a React Testing Library [14] nyújtja a megfelelő függvényeket, amelyek segítségével a komponensek valódi DOM [2] elemek ként tesztelhetőek. A tesztek futtatása során a szervert egy mock API helyettesíti, amely az MSW (Mock Service Worker) könyvtár segítségével lett megírva.

A kitelepítési folyamatok csak a megfelelő ágakra érkező commit-ok esetén futnak le (main vagy develop ág). Ezek a feladatok a megfelelő projektek gyökereiben található Dockerfile-ok

[2]Document Object Model - a JavaScript így reprezentálja a weboldal HTML tartalmát

alapján létrehoznak egy Docker image-et és azt feltöltik a GitLab Container registry-be. Az image kiépítése Kaniko segítségével történik, amely egy olyan eszköz, ami lehetővé teszi egy Dockerfile felépítését egy Docker container-en belül, és biztonságosabbnak, illetve gyorsabbnak bizonyul a gyakran használt Docker in Docker alternatívával szemben [1].

5.2. Docker és kitelepítés

A VoteHub alkalmazás Docker konténerizációt használ a szerver és web alkalmazások esetében. Ennek köszönhetően a futó szoftver környezetének a kialakítása a fejlesztő felelősége és a hoszt rendszergazdának nem kell ismernie a rendszer függőségeit [8]. Az (5.1)-es fejezetben említett kitelepítési folyamat keretén belül a létrejött Docker image a GitLab Container Registry-ből elérhető. Egy Docker image az alkalmazás egy standardizált csomagja, amely tartalmazza egy adott projekt összes állományát, binárisait és konfigurációt a futtatáshoz. A projekt kell tartalmazzon egy Dockerfile-t, ahhoz, hogy az alkalmazásból Docker image készülhessen. A Dockerfile az image készítéshez szükséges parancsokat tartalmazza. A 7. ábrán látható a webes alkalmazáshoz tartozó Dockerfile tartalma, a végső image a felépített alkalmazást és a futtatásához szükséges eszközöket tartalmazza. A szerver oldali image építés is hasonlóan történik. A létrejött image-ek futtathatóak is, ezeket nevezzük Docker konténereknek.

A létrejött image-ek kényelmes kezelésére a Docker Compose eszköz lett felhasználva, amely egyszerűsíti az alkalmazáson belüli szolgáltatások irányítását, megkönnyíti a kommunikációt, hálózati és adathozzáférési kapcsolatok létrehozását. Konfigurálása egy YAML fájlon keresztül történik, amely segítségével létrehozhatóak a projekten belüli szolgáltatások. A Docker Compose lehetőséget ad a szolgáltatások létrehozására, a konténerek elindítására és

```
# build stage
FROM node:22.10.0-alpine3.20 AS build
WORKDIR /app
COPY . .
RUN npm install
RUN npm run build

# runtime stage
FROM nginx:alpine3.19-slim AS runtime
COPY --from=build /app/config/nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=build /app/dist /usr/share/nginx/html
```

7. kódrészlet. Web alkalmazás Dockerfile-ja

leállítására.

Különálló git tárolóban található a kitelepítésért felelős konfiguráció. Ez a deploy tároló felelős a környezeti változók beállításáért, hozzáfér a Test szerverhez és konténereket hoz létre a Szerver és Web alkalmazások image-eiből. A Docker Compose YAML fájl öt szolgáltatás futtatásáért felelős:

1. Szerver
2. Web alkalmazás
3. MySQL adatbázis
4. Speaches átirat generáló mesterséges intelligencia
5. MinIO fájlalapú tárolórendszer

A deploy tároló pipeline-ja három manuálisan futtatható job-ot biztosít: szolgáltatások létrehozása és futtatása, naplózási előzmények megtekintése, szolgáltatások leállítása és törlése. A VoteHub alkalmazás a Codespring által biztosított teszt környezetbe van kitelepítve.

Következtetések és továbbfejlesztési lehetőségek

A VoteHub projekt keretein belül sikeresen megvalósult egy olyan, nagy mértékben testre szabható rendszer, amelynek segítségével az intézmények biztonságosan és hatékonyan bonyolíthatják le szavazásaikat.

Az alkalmazás számos más funkcionalitással bővíthető annak érdekében, hogy széles körben használható legyen. Ezek közül a következő továbbfejlesztési lehetőségek vannak tervben:

- A fórum meghívók kiküldése a felhasználó email címére.
- A gyűlésekről készült jegyzőkönyv kigenerálása pdf vagy egyéb dokumentum formátumban, amely részletes információt tartalmaz a gyűlések résztvevőiről, a gyűlés napirendi pontjairól, a szavazatok eredményeiről, és amelyhez hozzáilleszthető a gyűlésről készült átirat.
- Lehetőség, hogy a moderátorok fórum szinten sablonokat hozzanak létre, az előző alpontban említett jegyzőkönyvek számára.
- Lehetőség a moderátorok számára, hogy a gyűléshez szükséges iratokat feltöltsék a rendszerbe, amelyek elérhetővé válnak a tagok számára.
- Nyílt gyűlések létrehozása, amelyeket követhetnek fórumon kívüli személyek is. A zárt gyűlésekhez hasonlóan, ebben az esetben is csak a fórum tagok szavazhatnak, viszont külső személyek is követhetnék a szavazás állását. Ez például újságírók számára lehetne hasznos, hogy a rendszeren keresztül hozzáférhessenek az éppen aktív gyűlésekhez, melyekről esetleg írni szeretnének.
- Értesítések a mobil alkalmazásban a fórum tagok számára, a fórumon belül zajló tevékenységekről.
- Chat felület bevezetése egy fórumon belül.

Hivatkozások

[1] Ashish Choudhary. „Exploring Docker Alternatives”. *When Docker Meets Java: A Practical Guide to Docker for Java and Spring Boot Applications*. Springer, 2025, 147–172. oldal.

[2] Christopher Cowell, Nicholas Lotz és Chris Timberlake. *Automating DevOps with GitLab CI/CD Pipelines: Build efficient CI/CD pipelines to verify, secure, and deploy your code using real-life examples*. Packt Publishing Ltd, 2023.

[3] Adam L Davis és Adam L Davis. „Gradle”. *Learning Groovy 3: Java-Based Dynamic Scripting* (2019), 105–114. oldal.

[4] Radoslaw Dziadosz. „Liquibase: Version Control for Database Schema”. (2017).

[5] Oliver Gierke, Thomas Darimont és Christoph Strobl. „Spring Data JPA-Reference Documentation”. *URL https://docs. spring. io/spring-data/jpa/docs/current/reference/html/.[utoljára megtekintve: 2017. 04. 21.]* (2012).

[6] Mario Heiderich, Christopher Späth és Jörg Schwenk. „Dompurify: Client-side protection against xss and markup injection”. *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*. Springer. 2017, 116–134. oldal.

[7] Patil Krutika. „Redux State Management System-A Comprehensive Review”. *International Journal of Trend in Scientific Research and Development* 6.7 (2022), 1021–1027. oldal.

[8] Ian Miell és Aidan Sayers. *Docker in practice*. Simon és Schuster, 2019.

[9] Victoria Pimentel és Bradford G Nickerson. „Communicating and displaying real-time data with websocket”. *IEEE Internet Computing* 16.4 (2012), 45–53. oldal.

[10] Klaus Renzel és Wolfgang Keller. „Three layer architecture”. *Software Architectures and Design Patterns in Business Applications* 10 (1997).

[11] Edward Sciore és Edward Sciore. „JDBC”. *Database Design and Implementation: Second Edition* (2020), 15–47. oldal.

[12] Pankaj Singh és Vikas Gupta. „JWT BASED AUTHENTICATION”. ().

[13] Kristín Fjóla Tómasdóttir, Mauricio Aniche és Arie Van Deursen. „The adoption of javascript linters in practice: A case study on eslint”. *IEEE Transactions on Software Engineering* 46.8 (2018), 863–891. oldal.

[14] Mai Tran. „Testing React Applications Using React Testing Library”. (2023).