

**Szegedi Tudományegyetem**  
**Informatikai Intézet**

# **SZAKDOLGOZAT**

**Fehér Erik**  
**2023**

**Szegedi Tudományegyetem**  
**Informatikai Intézet**

**Közvetítés segítő és fejlesztő webes alkalmazás  
fejlesztése a Twitch.tv platformhoz**

Szakedolgozat

*Készítette:*

**Fehér Erik**

programtervező informatikus BSc  
szakos hallgató

*Témavezető:*

**Dr. Németh Tamás**

egyetemi docens

Szeged

2023

# Feladatkiírás

A szakdolgozat célja egy olyan alkalmazás tervezése és fejlesztése, amely segíti a Twitch.tv közvetítőit a tevékenységeik végzésében. Az alkalmazás autentikációt végezhajt a Twitch.tv segítségével, biztonságos munkamenetet biztosít, és lehetőséget nyújt a chatbot és a közvetítői összefoglaló felület használatára.

Az alkalmazás elkészítéséhez a következő feladatokat kell elvégezni: Az alkalmazás tervezése és architektúrájának kidolgozása, az alkalmazás funkcióinak megtervezése és implementálása, az alkalmazás felhasználói felületének tervezése és implementálása, az alkalmazás szerver oldali funkcióinak megtervezése és implementálása, beleértve a Firebase adatbázis kezelését, a Twitch eseményekre való feliratkozást, a chatbotot és a felhasználói felülethez szükséges szerver oldali kódrészleteket.

Az alkalmazásnak a következő funkciókat kell tartalmaznia: Authentikáció a Twitch.tv segítségével, biztonságos munkamenet biztosítása a weboldalon, chatbot a közvetítő Twitch üzenőfalán, amely válaszol a felhasználók parancsára, chatbot parancsainak konfigurálása, üzenőfalról és felhasználói felületről egyaránt, Twitch.tv által küldött eseményekre való hallgatás és azok megfelelő feldolgozása az eseménytől függően, egy közvetítői felület, amelyen a közvetítő a közvetítés tevékenységét képes nyomon követni.

Twitch.tv követés, feliratkozás és további eseményekre várakozó felület elkészítése, amely megfelelő felhasználó által feltöltött képet, üzenetet és hangot jelenít meg és beilleszthető megfelelően az Open Broadcaster Software alkalmazásba böngésző forrásként.

A következő technológiák használhatóak az alkalmazás elkészítéséhez: React keretrendszer a felhasználói felület megvalósításához, Node.js szerver oldali alkalmazás megvalósításához. Firebase az adatbázis megvalósításához, Twitch API használata a Twitch események, chatbot, és az autentikáció kezeléséhez, Open Broadcaster Software a böngésző forrás funkció használatához

# Tartalmi összefoglaló

## **A téma megnevezése:**

Közvetítés segítő és fejlesztő webes alkalmazás fejlesztése a Twitch.tv platformhoz

## **A megadott feladat megfogalmazása:**

Egy közvetítés segítő és fejlesztő alkalmazás elkészítése, a Twitch.tv platformhoz a közvetítők részére, akiknek segít a munkájukban és fejleszti az adásaik minőségét.

## **A megoldási mód:**

Egy alkalmazás készítése, ami egy közvetítés segítő és fejlesztő eszközt nyújt a közvetítőknek, ami könnyen használható a felhasználók számára. Az eszköz megoldást nyújt a monoton válaszok adására automatikusan és grafikus elemekkel ruházza fel a közvetítő adását, amit képes személyre szabni. Ezek mellett egy letisztult és informatív nézetet kínál az élő közvetítésekről és azok tartozékairól.

## **Alkalmazott eszközök, módszerek:**

- React keretrendszer
- Node.js keretrendszer
- Tailwind stílus keretrendszer
- Firebase adatbázis és hosting rendszer
- IntelliJ Idea fejlesztői környezet
- Visual Studio fejlesztői környezet
- Postman kérés küldő alkalmazás
- GitHub verziókövető rendszer

- Twitch API

**Elért eredmények:**

Egy alkalmazás amely segítségével a közvetítők egyszerűen tudják vezényelni a közvetítéseket a letisztult felhasználói felület miatt, emellett jobb minőségben, mivel az interakció a közvetítő és a nézők között jelentősen nő az alkalmazás használata közben.

**Kulcsszavak:**

Közvetítés, Twitch.tv, alkalmazás, React, Node.js, Tailwind

# Tartalomjegyzék

Feladatkiírás . . . . .	1
Tartalmi összefoglaló . . . . .	2
Tartalomjegyzék . . . . .	4
<b>1. Bevezetés</b>	<b>7</b>
<b>2. Használt technológiák és eszközök</b>	<b>9</b>
2.1. Fejlesztői környezetek . . . . .	9
2.1.1. Visual Studio Code . . . . .	9
2.1.2. Webstorm . . . . .	10
2.2. Programozási nyelvek és keretrendszerek . . . . .	10
2.2.1. React . . . . .	10
2.2.2. NodeJS . . . . .	11
2.2.3. Tailwind . . . . .	11
2.3. Egyéb szolgáltatások és alkalmazások . . . . .	12
2.3.1. Firebase . . . . .	12
2.3.2. Postman . . . . .	12
2.3.3. GitHub . . . . .	13
2.3.4. Ngrok . . . . .	13
2.3.5. Twitch API . . . . .	13
2.3.6. Twitch CLI . . . . .	14
<b>3. Az alkalmazás felépítése és beüzemelése</b>	<b>15</b>
3.1. Az alkalmazás beüzemelése . . . . .	15
3.1.1. Twitch beüzemelése . . . . .	15

3.1.2.	Firestore beüzemelése . . . . .	16
3.1.3.	Ngrok beüzemelése . . . . .	16
3.1.4.	Alkalmazás elindítása . . . . .	17
3.2.	Az alkalmazás felépítése . . . . .	17
3.2.1.	Adatbázis . . . . .	17
3.2.2.	Szerver oldal . . . . .	18
3.2.3.	Kliens oldal . . . . .	19
<b>4.</b>	<b>Az alkalmazás szerver oldali részletes áttekintése</b>	<b>20</b>
4.1.	Indítás utáni inicializálás szerver oldalon . . . . .	20
4.2.	Az adatbázis és szerver oldal kapcsolata . . . . .	21
4.3.	Twitch chatbot implementáció . . . . .	21
4.4.	A szolgáltatási osztályok . . . . .	22
4.4.1.	AlertboxService szolgáltatás . . . . .	22
4.4.2.	CommandService szolgáltatás . . . . .	22
4.4.3.	TwitchService szolgáltatás . . . . .	23
4.4.4.	UserManagementService szolgáltatás . . . . .	24
4.5.	A vezérlő osztályok . . . . .	26
4.5.1.	AlertboxController vezérlő . . . . .	26
4.5.2.	CommandsController vezérlő . . . . .	27
4.5.3.	TwitchController vezérlő . . . . .	27
4.5.4.	UserManagementController vezérlő . . . . .	28
4.6.	Az AlertboxWebSocket . . . . .	29
<b>5.</b>	<b>Az alkalmazás kliens oldali részletes áttekintése</b>	<b>30</b>
5.1.	Főoldal . . . . .	30
5.1.1.	Bejelentkezés és regisztráció . . . . .	30
5.1.2.	Navigációs sáv . . . . .	31
5.2.	Irányítópult . . . . .	31
5.2.1.	Nyitóoldal . . . . .	32
5.2.2.	Kommand kezelő oldal . . . . .	32
5.2.3.	Eseménykezelő oldal . . . . .	32

5.3. Esemény megjelenítő . . . . .	34
5.4. Privát oldalak . . . . .	34
<b>6. Chatbot funkciói</b>	<b>36</b>
<b>7. További fejlesztési lehetőségek</b>	<b>37</b>
7.1. Adakozás funkció . . . . .	37
7.2. Automatikus üzenetek . . . . .	37
7.3. Személyre szabhatóbb események . . . . .	37
<b>8. Összefoglaló</b>	<b>38</b>
<b>Irodalomjegyzék</b>	<b>39</b>
<b>Nyilatkozat</b>	<b>40</b>



# 1. fejezet

## Bevezetés

Az élő közvetítések története az internet előtti időkre nyúl vissza, amikor a televízió volt az elsődleges médium. A televízióadások korlátozottak voltak a műsorrendjük és az időzítésük tekintetében, és az élő közvetítések különösen kihívást jelentettek a műsorszolgáltatók számára. Az élő közvetítésekhez speciális eszközökre és infrastruktúrára volt szükség, amelyek nagyon drágák voltak, és nehéz volt eljuttatni őket a nagyközönséghez.

Az internet megjelenése és az élő közvetítések online terjesztésének lehetősége új időszakot hozott a média világában. Az interneten keresztül az emberek számára elérhetővé váltak az élő közvetítések, és ez az új lehetőség azonnal nagy népszerűsége tette szert. Az élő közvetítések lehetővé tették a nézők számára, hogy közvetlenül interakcióba lépjenek a közvetítővel, és nagyobb élményt nyújtottak a nézőknek, mint a hagyományos televíziós műsorok.

Az online közvetítések platformjai között a Twitch.tv az egyik legnépszerűbb. A Twitch.tv egy online streaming platform, amely kizárólag a videojáték közvetítéseknek szentelt. Az oldal 2011-ben indult, és azóta a világ egyik legnagyobb élő közvetítési platformjává vált. Az oldal lehetővé teszi a közvetítőknél, hogy élőben közvetítsenek a játékuokról, míg a nézők élőben figyelemmel kísérhetik a közvetítéseket és közvetlenül interakcióba léphetnek a közvetítővel a chat ablakban.

Az elmúlt években a Twitch.tv platform jelentős változásokon ment keresztül, amelyek nagy hatással voltak az online közvetítések világára. Az oldal kiterjesztette az általa támogatott közvetítések témáját, és ma már lehetővé teszi az élő közvetítéseket a videojátékok mellett az oktatási, az egészségügyi és az életmód témákban is.

Az alkalmazás a Twitch.tv platformhoz készített webes alkalmazásra fókuszál. Az alkalmazás célja, hogy segítse a közvetítőket az interakció javításában és a közönség elkötelezettségének növelésében. Az alkalmazás számos funkcióval rendelkezik, amelyek az interakciót a közvetítők és a közönség között hatékonyabbá és élvezetesebbé teszik.

## 2. fejezet

# Használt technológiák és eszközök

Ebben a fejezetben részletesen bemutatásra kerülnek azok a technológiák, programozási nyelvek, keretrendszerek és eszközök, amelyek nélkülözhetetlenek voltak a szakdolgozatom elkészítéséhez.

### 2.1. Fejlesztői környezetek

A fejlesztői környezetek (IDE-k) olyan szoftverek, amelyek megkönnyítik a fejlesztők munkáját azzal, hogy beépített funkciók segítségével felgyorsítják a fejlesztési folyamatot. Az IDE-k általában tartalmazzák a szintaxis kiemelését, kód javaslatokat, kód navigációt, gyors futtatást és tesztelést, amelyek segítenek a fejlesztőknek hatékonyabbá tenni a munkájukat. Az IDE-k használata különösen fontos, ha a fejlesztőnek nagyobb projektje van, mivel a nagyobb kódolási feladatok nagyobb időt és figyelmet igényelnek. Az IDE-k kényelmes felhasználói felületet és könnyen használható funkciókat biztosítanak, amelyekkel a feladatok gyorsabban és hatékonyabban végezhetők.

#### 2.1.1. Visual Studio Code

A Visual Studio Code egy modern, nyílt forráskódú fejlesztői környezet, amelyet kifejezetten a web- és felhőalapú alkalmazásokhoz terveztek. A Microsoft által készített alkalmazás számos beépített funkcióval rendelkezik, amelyek lehetővé teszik a hatékony fejlesztést, például a szintaxis kiemelését, a kód előzetes megjelenítését, az automatikus

kódjavaslatokat és a verziókezelő rendszerek támogatását. A Visual Studio Code támogatja a különböző programozási nyelveket, keretrendszereket és platformokat, és számos hasznos bővítménnyel bővíthető. Összességében az alkalmazás erőteljes és sokoldalú eszköz a modern web- és felhőalapú alkalmazások fejlesztéséhez. [10]

### **2.1.2. Webstorm**

A Webstorm egy integrált fejlesztői környezet, amelyet a JetBrains fejlesztett ki a Javascript nyelvű alkalmazások fejlesztéséhez. Az alkalmazás beépített funkciói közé tartozik a szintaxis kiemelés, kód navigáció, refaktorálás és kódgenerálás, és támogatja a különböző Javascript keretrendszereket és technológiákat, például a NodeJS-t. A Webstorm kiterjeszthető bővítményekkel, amelyek még hatékonyabbá teszik a fejlesztést. Összességében a Webstorm erőteljes és sokoldalú eszköz a Javascript nyelvű alkalmazások fejlesztéséhez a fejlesztők széles körű támogatása és számos hasznos funkciója révén. [11]

## **2.2. Programozási nyelvek és keretrendszerek**

A programozási nyelvek olyan nyelvek, amelyeket a számítógépek megértik és végrehajtanak. Ezekkel a nyelvekkel írt programok különböző célokra használhatók, például weboldalak készítésére, adatbázisok kezelésére, játékok írására és még sok másra. A keretrendszerek olyan programozási eszközök, amelyek segítségével a programozók hatékonyabban és gyorsabban tudnak dolgozni. Ezek a keretrendszerek általában előre meghatározott szabályokat, eljárásokat és sablonokat tartalmaznak, amelyeket a fejlesztők felhasználhatnak a saját projektjeikben. Ezáltal az időt takarítják meg és csökkentik a hibalehetőségeket a programozás során.

### **2.2.1. React**

A React egy ingyenesen elérhető, JavaScript alapú keretrendszer, amit a Facebook fejlesztett. A React lehetővé teszi, hogy a fejlesztők összetett felhasználói felületeket hozzanak létre a webalkalmazásaikban, amelyek változhatnak az adatok vagy a felhasználói interakciók alapján. A React komponens alapú architektúrát alkalmaz, amelynek köszön-

hetően a fejlesztők újrafelhasználják a kódot és a komponenseket, így gyorsabb és hatékonyabb munkavégzést biztosít. A React leggyakrabban a webalkalmazások fejlesztése során használják, de a React Native keretrendszer segítségével akár mobilalkalmazásokat is lehet készíteni. A React használata más fejlesztői eszközökkel, mint például a Node.js, az NPM és az Yarn is gyakori. A React népszerűségét az egyszerűen használható API-ja, a kiváló teljesítménye és az aktív közössége adja. [6]

### **2.2.2. NodeJS**

A Node.js egy szerveroldali, JavaScript alapú futtatási környezet, amely lehetővé teszi a fejlesztők számára, hogy JavaScript-ben írt alkalmazásokat futtassanak a szerveren. Az NPM-en keresztül számos modul érhető el, amelyek segítségével az alkalmazásfejlesztés egyszerűbbé és hatékonyabbá tehető. A Node.js nagy előnye, hogy aszinkron módon kezeli a kéréseket és válaszokat, így nagy terhelés mellett is hatékonyan működik. A Node.js lehetővé teszi a fejlesztők számára, hogy ugyanazt a JavaScript kódot használják a kliens- és a szerveroldalon, így egyszerűbb és hatékonyabb kód írása válik lehetővé. Az Express.js, a Koa.js és a Meteor.js a Node.js-hez kapcsolódó keretrendszerek, amelyek megkönnyítik az alkalmazásfejlesztést. A Node.js-t nemcsak webalkalmazások fejlesztésére, hanem parancssori alkalmazások, chatbotok és IoT alkalmazások területén is használják. A Node.js egy erőteljes és sokoldalú környezet a JavaScript alapú alkalmazások fejlesztéséhez, amely számos lehetőséget kínál a fejlesztőknek. [4]

### **2.2.3. Tailwind**

A Tailwind CSS egy nyílt forráskódú CSS keretrendszer, amely célja, hogy a weboldalak és webalkalmazások tervezése gyorsabb és hatékonyabb legyen. A fejlesztőknek lehetőséget ad, hogy azonnal és rugalmasan hozzanak létre testre szabott felhasználói felületeket, és közben minimalizálják a CSS kód ismétlődését és az időigényes stíluskód írását. A Tailwind CSS előre definiált osztályokat biztosít, amelyeket a HTML elemekhez lehet hozzárendelni, például a margók, a betűméretek, a színek és a keretek beállításához. A testreszabás lehetősége is adott, így a fejlesztők további osztályokat hozhatnak létre vagy módosíthatják a meglévőket, hogy azok jobban megfeleljenek az adott projekt

igényeinek. A Tailwind CSS használata egyszerű, mivel a változók használatával egyszerűen definiálhatók az általános stílusok, és könnyen módosíthatók a változók, ha szükséges. Összességében a Tailwind CSS hatékony és könnyen használható CSS keretrendszer, amely lehetővé teszi a fejlesztők számára, hogy gyorsan és hatékonyan tervezzenek testre szabott felhasználói felületeket, miközben minimalizálják a CSS kód ismétlődését és az időigényes stíluskód írását. [7]

## **2.3. Egyéb szolgáltatások és alkalmazások**

Az egyéb szolgáltatások és alkalmazások között szerepelnek olyan fejlesztői platformok és eszközök, amelyek segítenek a szoftverfejlesztés folyamatában, mint például a verziókezelő rendszerek, a backend szolgáltatások és az API-fejlesztő eszközök. Emellett vannak olyan online szolgáltatások is, amelyek lehetővé teszik a fejlesztők számára, hogy hozzáférjenek azokhoz az alkalmazásokhoz, amelyeket helyi számítógépen futtatnak, és olyan programozási felületek és eszközök, amelyek segítségével könnyen integrálhatók a Twitch streaming platformmal.

### **2.3.1. Firebase**

A Google által kínált Firebase egy fejlesztői platform és backend szolgáltatás, amely számos funkciót biztosít a fejlesztőknek, beleértve a felhasználókezelést, az adatbáziskezelést, az értesítéseket, az autentikációt, az analitikát, a fiókkövetést és a felhőtárhelyet. A Firebase célja, hogy megkönnyítse a mobilalkalmazások és webalkalmazások fejlesztését és üzemeltetését. Az API-k használatával a fejlesztők kommunikálhatnak a Firebase szervereivel, így csökkenthetik az infrastruktúra kiépítésére és a backend kódolására fordított időt. [1]

### **2.3.2. Postman**

A Postman egy olyan eszköz, amely lehetővé teszi az API-k tesztelését, dokumentálását és megosztását a fejlesztők számára. Az API-k tesztelése egyszerűvé válik a Postman segítségével, mivel a felhasználók könnyen megérthetik, hogy az API-k hogyan működnek,

és hogyan lehet velük kommunikálni. A Postman segítségével lehetőség van az API-k dokumentálására is, ami megkönnyíti más fejlesztők számára, hogy megértsék, hogyan kell használni az API-kat. Továbbá, a Postman lehetővé teszi az API-k megosztását is, így a fejlesztők egyszerűen tudják megosztani azokat a fejlesztői közösségben. [5]

### **2.3.3. GitHub**

A GitHub egy webes alapú verziókezelő és kollaborációs platform, amely lehetővé teszi a fejlesztők számára, hogy együttműködjenek szoftverprojekteken. A GitHubon a fejlesztők tárolhatják az alkalmazásaik és projektek verzióit, és egyszerűen kezelhetik ezeket. A projekt tagjai hozzáférhetnek az adott projekthez, megoszthatják a munkájukat, és figyelemmel kísérhetik, hogy milyen változásokat hajtanak végre a többi tag. A GitHub továbbá számos kiegészítő eszközt és funkciót is kínál, mint például projektmenedzsment eszközöket, kódkönyvtárat, ügyféloldali alkalmazásokat és webhookokat. [2]

### **2.3.4. Ngrok**

Az ngrok egy olyan szolgáltatás, amely lehetővé teszi a fejlesztők számára, hogy a helyi fejlesztői környezetüket publikus címmé alakítsák át, így az interneten keresztül elérhetővé teszik a szerverüket. Az ngrok támogatja a TCP és az HTTP protokollokat, és egy könnyen használható felülettel rendelkezik, amely lehetővé teszi a szerverek könnyű konfigurálását és kezelését. Az ngrok különösen hasznos lehet a webalkalmazások és az API-k fejlesztése során, amikor a fejlesztőknek tesztelniük kell az alkalmazást vagy az API-t, és szeretnék azt más helyekről is elérhetővé tenni. [3]

### **2.3.5. Twitch API**

A Twitch API egy olyan programozási interfész, amely lehetővé teszi a fejlesztők számára, hogy hozzáférjenek és kezeljék a Twitch platformjának adatátviteli lehetőségeit. Az API-n keresztül a fejlesztők képesek lekérdezni a Twitch szervereit, így információkat szerezhetnek az élő adásokról, a csatornákról, a felhasználókról, a videókról, a játékokról és még sok másról. Az API-t felhasználva a fejlesztők képesek Twitch alkalmazásokat és szolgáltatásokat fejleszteni, amelyek az élő adásokkal, az interakciókkal

és a közösségi funkciókkal kapcsolatosak. A Twitch API lehetőséget biztosít az integrációkra, amelyek más alkalmazásokkal és szolgáltatásokkal kapcsolódnak, mint például a játékstream-ekkel kapcsolatos alkalmazások és weboldalak. Az API lehetővé teszi a felhasználók számára, hogy létrehozzanak saját Twitch integrációkat és alkalmazásokat, amelyek széles körű lehetőségeket kínálnak az élő adásokkal és a Twitch közösségével való interakcióra. [8]

### **2.3.6. Twitch CLI**

A Twitch CLI egy parancssori interfész, amely lehetővé teszi a fejlesztők számára, hogy az adatokat a Twitch API-ból kérjék le, és parancssorból kezeljék a Twitch fiókjaikat. A CLI lehetővé teszi a felhasználóknak, hogy stream-eket indítsanak és állítsanak le, megtekintsék a nézettségi adatokat, kezeljék a Twitch-fiókjaikat és még sok mást. A Twitch CLI egy nyílt forráskódú projekt, amely lehetővé teszi a fejlesztők számára, hogy testreszabják a parancssori interfészt a saját igényeiknek megfelelően. A CLI nagyon hasznos lehet a Twitch streamereknek és fejlesztőknek, akik szeretnék könnyen és gyorsan kezelni Twitch fiókjaikat a parancssorban. [9]



## 3. fejezet

# Az alkalmazás felépítése és beüzemelése

### 3.1. Az alkalmazás beüzemelése

Ebben a fejezetben részletes leírások találhatóak az alkalmazás külső szolgáltatásainak és programjainak beüzemeléséhez, mint például a Twitch, Firebase, ngrok és a saját program konfigurációja.

#### 3.1.1. Twitch beüzemelése

Az alkalmazás beüzemelése számos lépést igényel, amelyek között először is regisztráció vagy bejelentkezés szükséges a Twitch.tv oldalon. Miután sikeresen bejelentkeztünk, továbblépünk a Twitch fejlesztői oldalára, ahol ismét be kell jelentkezünk a Twitch fiókkal. Ezt követően a saját konzolra kell navigálni, ahol megtaláljuk az alkalmazás regisztrációjának lehetőségét, ahol számos információt kell megadnunk.

Itt lehetőségünk van egyedi név kiválasztására az alkalmazás számára, amely segít az azonosításban és a kommunikációban a Twitch rendszerében. Az alkalmazás kategóriájának kiválasztása is fontos lépés, például ha egy Chat Bot funkciót szeretnénk implementálni, ezt itt kell jelezni. Az alkalmazás átirányítási címének megadása is szükséges, ez az a weboldal címe, ahová a felhasználók át lesznek irányítva a Twitch autentikáció során.

Miután az alkalmazást regisztráltuk, kapunk egyedi titkos kulcsot és kliens kulcsot. Ezek a kulcsok rendkívül fontosak, mivel ezekkel azonosítjuk az alkalmazásunkat a Twitch rendszerében, és lehetővé teszik az alkalmazás és a Twitch közötti biztonságos

kommunikációt. Fontos megjegyezni, hogy ezeket a kulcsokat bizalmasan kell tárolni, és nem szabad másokkal megosztani, mivel ha illetéktelen személyek hozzáférnek ezekhez, akkor azok a nevünkben bármilyen tevékenységet végezhetnek a Twitch rendszerében. Az API hozzáférési token kéréséhez ezeket a kulcsokat kell használni, amit egy HTTP kérés segítségével lehet elvégezni. Az ehhez szükséges leírás megtalálható a Twitch API dokumentációjában.

Mivel Chat Bot-ot is tartalmaz az alkalmazás annak is szüksége lesz egy külön felhasználói fiókra a Twitch.tv-n, ennek a felhasználónak szüksége lesz az OAuth azonosítójára, amit ezen a weboldalon lehet kérni `“https://twitchapps.com/tmi/”`.

### **3.1.2. Firebase beüzemelése**

A Firebase használatához szükséges első lépések a Firebase weboldalán való bejelentkezés és egy új projekt létrehozása. A projektnek egyedi nevet kell adni, és el kell fogadni a Google Analytics használatára vonatkozó feltételeket a projektben. Miután a projekt létrejött, ki kell választani, hogy egy webalkalmazást szeretnél hozzáadni a projekthez. Az új webalkalmazásnak ismételten egy nevet kell adni. Ezzel az alkalmazás elkészült a Firebase rendszerében, és ezt követően használhatod az elérhető szolgáltatásokat, például a Firebase adatbázist és tárolót.

### **3.1.3. Ngrok beüzemelése**

Az első lépés az ngrok szolgáltatásra való regisztráció. Ezután, az ngrok kliens alkalmazás telepítése szükséges a fejlesztői környezetbe, ami elérhető az ngrok weboldaláról. Telepítés után az ngrok parancsot hozzá kell adni a rendszerhez, hogy elérhető legyen a parancs rendszer szintű környezeti változóiban. Az autentikációhoz szükséges kulcsot megkapod a weboldalon való bejelentkezést követően, amit a ngrok config add-authtoken token parancs segítségével kell hozzáadni. Ezzel az autentikáció sikeresen megtörtént, és használható a szolgáltatás.

### **3.1.4. Alkalmazás elindítása**

Az alkalmazás elindításához a fentebb említett szolgáltatások és az alkalmazás összehangolása szükséges. A Twitch és az alkalmazás összehangolása egyszerű, csupán a szerver oldali alkalmazás környezeti változói közé kell beilleszteni az egyedi titkos kulcsot, a kliens kulcsot, a hozzáférési tokent, a chatbot-hoz szükséges OAuth azonosítót és az ehhez tartozó felhasználói fiók nevét.

Az alkalmazás és a Firebase szinkronizálása nem bonyolult, mivel az alkalmazás a Firebase Admin SDK-t használja. Ehhez szükség van egy privát kulcsra, amelyet a projekt beállításában találhatsz meg, amikor egy új kulcs generálása történik. Ezután letöltésre kerül egy JSON fájl, amit `serviceAccountKey.json` néven kell elmenteni, majd be kell másolni az alkalmazás config mappájába.

Az ngrok és az alkalmazás összehangolása kissé bonyolultabb lehet, ugyanis minden port amin hallgat az alkalmazásunk közvetíteni kell a világháló felé, hogy elérhető legyen a nagyközönségnek. Ehhez az ngrok szolgáltatást kell használni, és a következő parancsot kell meghívni a 8080, 8000 és 3000 portokhoz: `ngrok http port` A portok megnyitása után kapunk minden porthoz egy hivatkozási címet és ezeket a címeket be kell illeszteni a szerver oldali és kliens oldali kód környezeti változói közé.

Ezek után az alkalmazás készen áll az indításra.

## **3.2. Az alkalmazás felépítése**

### **3.2.1. Adatbázis**

Az alkalmazás adatbázisnak a Firebase szolgáltatást használja a felhasználóbarátsága miatt. Az adatbázis JSON formátumot használ és a következő kollekciókat tartalmazza: `user`, `session`, `command`, `alertbox`.

A `user` kollekció a felhasználók adatait tárolja, amelyek magukban foglalják a dokumentum azonosítóját (`document ID`), a felhasználó Twitch azonosítóját (`Twitch ID`), valamint az email címet (felhasználók email címe), a felhasználó azonosítóját (`userID`), a felhasználó nevét (Twitchről átvett felhasználói név), valamint az `access token`-t, amely lehetővé teszi a hozzáférést a Twitch felhasználói fiókjához. Egy felhasználóhoz egy da-

rab dokumentum tartozik a user kollekcióból.

A command kollekció tartalmazza a chatbot parancsait, amelyekhez az automatikusan generált dokumentum azonosító (document ID) tartozik, és a dokumentumon belül az adatok a következők: a parancs neve (command néven), a visszatérő érték (üzenet result néven) és a felhasználó azonosítója (userID), ami azt jelzi, hogy melyik felhasználóhoz tartozik a parancs. Egy felhasználóhoz több parancs is tartozhat, ezért a command kollekcióban egy felhasználóhoz több dokumentum is tartozhat.

Az alertbox kollekció a Twitch eseményekhez kapcsolódó megjelenítendő adatokat és beállításokat tartalmazza. Egy felhasználóhoz annyi dokumentum tartozhat ebben a kollekcióban, amennyi fajta eseményt kezelünk az alkalmazásban. A kollekcióban tárolt adatok a következők: a felhasználó azonosítója (userID) ami megadja, hogy kihez tartozik az esemény, az esemény típusa (type) ami meghatározza az esemény jellegét, az üzenet (message) ami megjelenik az esemény érkezésekor a felhasználónak, az időtartam (duration) ami meghatározza a megjelenített idő hosszát, a szöveg felolvasó szolgáltatás (tts) hangerő (tts volume) ami null értékkel bírva kikapcsolja a felolvasást, különben a felolvasás hangerejét állítja be, a zene hangerő (volume) ami a lejátszandó zene hangerejét határozza meg, valamint az audio fájl neve (audio file name) és az kép fájl neve (image file name) ami az feltöltött fájlok eredeti neveit tartalmazzák.

A session kollekció az oldalon található biztonságos böngészésért felelős, és tartalmazza a frissítő kulcsot (refresh key), ami a bejelentkezett felhasználóhoz kapcsolódik. Minden felhasználóhoz egy dokumentum tartozhat, ahol az azonosítója (userID) szolgál dokumentum azonosítóként.

### **3.2.2. Szerver oldal**

A szerver oldal az alkalmazás számítási és feldolgozási része. Ez a komponens kezeli az adatbázisból érkező lekérdezéseket, valamint az üzleti logikát és az alkalmazás működését irányítja. A szerver oldal felelős az adatok feldolgozásáért, az üzleti logika végrehajtásáért, az autentikáció és az engedélyezés kezeléséért, valamint a válaszok elküldéséért a kliens oldal felé.

### **3.2.3. Kliens oldal**

A kliens oldal az alkalmazás felhasználói felülete, amellyel a felhasználók az alkalmazással kommunikálnak. A kliens oldal felelős a felhasználói interakciók kezeléséért, az adatok megjelenítéséért, az adatok felhasználói bemenetekre történő validálásáért és az adatok továbbításáért a szerver oldal felé.

## 4. fejezet

# Az alkalmazás szerver oldali részletes áttekintése

### 4.1. Indítás utáni inicializálás szerver oldalon

A szerver oldal elindítása után a program beállítja és meghívja a szükséges funkciókat az alkalmazás működéséhez. Első sorban az Index osztály konstruktora meghívja az initExpress, initFirebase, start és listen metódust.

Az initExpress metódus beállítja az Express alkalmazást, beállítja a CORS-t (Cross-Origin Resource Sharing) a megengedett eredeti címekkel, és engedélyezi a fájlfeltöltést a fileUpload modul segítségével. Az initFirebase metódus inicializálja a Firebase Admin SDK-t a megadott hitelesítési adatokkal (serviceAccount), és beállítja a Firebase Storage tárolási egységet.

A start metódus lekéri az adatbázisból az összes felhasználónevet egy DAO objektum segítségével, majd ezeket az adatokat továbbítja a Twitch objektumnak, amely a ChatBot működéséért felelős. Ezután inicializálja az alkalmazás vezérlő osztályait (UserController, TwitchController, AlertboxController, CommandsController) és beállítja az AlertWebSocketet.

A listen metódus elindítja az Express alkalmazást, és meghallgatja a megadott portot (amelyet a this.port változó tartalmaz), majd kiírja a konzolra az alkalmazás elérhető URL-jét.

## **4.2. Az adatbázis és szerver oldal kapcsolata**

Az alkalmazás inicializálása során a Firebase szolgáltatással való adatbázis-szerver oldali kapcsolat kiépítése után a kommunikáció a DAO (Data Access Object) osztályon keresztül történik. A DAO osztály felelős az adatok eléréséért és kezeléséért a Firebase szolgáltatáson keresztül, lehetővé téve az alkalmazás számára az adatbázis műveletek végrehajtását, például adatok lekérdezését, módosítását, törlését.

## **4.3. Twitch chatbot implementáció**

A Twitch osztály egy Twitch chatbotot valósít meg, amely a Twitch chaten keresztül kapcsolódik az IRC (Internet Relay Chat) szerverhez és kezeli az üzeneteket, parancsokat, eseményeket és visszaküldi a válaszokat az üzenetekre és parancsokra a Twitch chaten keresztül.

A `connectionHandler` metódus csatlakozik a Twitch IRC szerverhez, beállítja az eseménykezelőket (event handlers) az üzenetek, kapcsolódási hibák, és kapcsolat bezárása eseményekre és a `users` paraméterben megadott felhasználókhoz csatlakozik. A `connectionMessage` metódus kezeli az érkező üzeneteket a Twitch IRC szerverről, kiolvassa az üzenet típusát és továbbítja az üzeneteket a `typeHandler` függvénynek. A `connectionFail` metódus kezeli a kapcsolódási hibákat, amíg a `connectionClose` metódus kezeli a kapcsolat bezárását. A `joinChannel` metódus belép egy Twitch csatornára a megadott felhasználók listájával. A `getMessageType` metódus kiolvassa az üzenetek típusát az üzenetekből, amelyeket a Twitch IRC szerverről kapott. A `typeHandler` metódus kezeli az üzeneteket típusuktól függően. Például `PRIVMSG` típusú üzenetekre válaszol, bejövő kommandókat észlel és kezeli, stb. A `messageHandler` metódus kezeli az üzeneteket, amelyeket a Twitch IRC szerverről kapott, kiszedi az üzenet küldőjének adatait, az üzenet tartalmát és a csatorna nevét. A `sendMessage` metódus elküld egy üzenetet a megadott csatornára. A `commandHandler` metódus kezeli a bejövő üzenetekben található parancsokat, és választ ad rájuk. A `basicCommands` metódus kezeli az alapvető parancsokat amik a felhasználókhoz tartoznak. Az `adminCommands` metódus kezeli az adminisztrátori parancsokat, amelyekhez különleges jogosultság szükséges.

## **4.4. A szolgáltatási osztályok**

A szolgáltatási osztályok felelnek a logika felépítéséért az alkalmazásban.

### **4.4.1. AlertboxService szolgáltatás**

Ez a szolgáltatási osztály felelős az Alertbox entitások kezeléséért. Az osztály tartalmazza az Alertbox entitások mentésére és frissítésére szolgáló metódusokat, valamint az ahhoz tartozó fájlok (képek és hangok) mentésére szolgáló metódust.

A saveAlertBox metódus felelős az Alertbox entitás mentéséért vagy frissítéséért az adatbázisban. Megkapja az Alertbox entitáshoz szükséges adatokat paraméterként (id, type, image, audio, message, volume, ttsvolume, duration), ellenőrzi a paramétereket (pl. üzenet hossza, fájlnevek hossza), és végrehajtja a szükséges mentési vagy frissítési műveleteket az adatbázisban a DAO (Data Access Object) osztály segítségével.

A saveChannelPoint metódus felelős a Channel Point, vagyis egy különleges Alertbox mentéséért vagy frissítéséért az adatbázisban. Megkapja a szükséges adatokat paraméterként (id, type, ttsvolume), ellenőrzi a paramétereket, és végrehajtja a szükséges mentési vagy frissítési műveleteket az adatbázisban a DAO osztály segítségével.

A saveFile metódus felelős fájlok (képek és hangok) mentéséért. Megkapja a fájlt, az Alertbox entitás azonosítóját (id), az Alertbox típusát (alertType), valamint a fájl típusát (fileType) paraméterként. Ellenőrzi a fájl típusát és méretét, majd menti a fájlt az adatbázisba az egyedi nevével a DAO osztály segítségével, majd visszatér a mentés sikerességével (true vagy false).

### **4.4.2. CommandService szolgáltatás**

A CommandService osztály a parancsokkal foglalkozik, amit a chatbot használ a felhasználók üzenőfalán.

A getCommands függvény az adott felhasználóhoz tartozó összes parancsot lekéri az adatbázisból a DAO osztály segítségével, azaz visszaadja az összes parancsot az adott felhasználó azonosítójával megegyezően.

A modifyCommand függvény lehetővé teszi egy meglévő parancs módosítását vagy új parancs létrehozását az adatbázisban. Ellenőrzi, hogy a parancs és az üzenet nem üres,



valamint hogy a parancs hossza nem haladja-e meg a maximális megengedett értéket (100 karakter), és az üzenet hossza nem haladja-e meg a maximális megengedett értéket (500 karakter). Ha a parancs már létezik az adatbázisban, akkor az üzenetét frissíti, különben pedig létrehoz egy új parancsot.

A `deleteCommand` függvény lehetővé teszi egy meglévő parancs törlését az adatbázisból. Az adott felhasználó azonosítójával és a parancs nevével azonosítja a törlendő parancsot, majd törli azt az adatbázisból.

A `createCommand` függvény egy új parancs létrehozását végzi az adatbázisban a DAO osztály segítségével.

A `updateCommand` függvény egy meglévő parancs frissítését végzi az adatbázisban a DAO osztály segítségével.

#### **4.4.3. TwitchService szolgáltatás**

A `TwitchService` osztály egy Twitch eventekkel kapcsolatos szolgáltatást valósít meg.

A `getSecret` függvény visszaadja a titkot (secret), amit az eseményekre való feliratkozásnál használnak. A jelenlegi implementációban a titkot a `SECRET HMAC` környezeti változóból olvassa ki.

A `getHmacMessage` függvény összeállítja az HMAC (hash-alapú üzenet hitelesítő kód) generálásához szükséges üzenetet a Twitch értesítési kérés fejléceiből és testéből.

A `getHmac` függvény generálja az HMAC kódot a kapott titok és üzenet alapján. A HMAC algoritmus a `crypto` modul segítségével kerül használatra.

A `verifyMessage` függvény összehasonlítja az előzőleg generált HMAC kódot a Twitch értesítés fejlécében kapott ellenőrző aláírással. Az összehasonlítás biztonságosan történik a `crypto.timingSafeEqual()` függvénnyel.

A `subscribeToRequiredEvents` függvény feliratkozik a szükséges Twitch eseményekre egy adott felhasználó kontextusában.

A `subscribeToTwitchEvent` függvény elküld egy HTTP kérést az eseményekre való feliratkozásra a megadott felhasználói azonosítóval, hozzáférési tokennel, és esemény típusal. A kérés törzsében meghatározza az esemény típusát, a callback URL-t, és az HMAC titkot. A kérés válaszát konzolra logolja, vagy hibát dob, ha a feliratkozás sikertelen.

A `decideEventType` függvény az értesítések alapján eldönti az esemény típusát, és az esemény típusának megfelelő eseménykezelő függvényt hívja meg.

A `deleteAllEventListener` függvény az aktuális Twitch Eventsub eseményfigyelők törlését végzi el. Először elküld egy GET kérést a Twitch API-hoz, hogy lekérje az aktuális eseményfigyelők listáját, majd az eredmény alapján végigiterál és mindegyik eseményfigyelőt törli a `deleteEventListenerAfterRevoked` függvény segítségével.

A `deleteEventListenerAfterRevoked` a függvény az adott id-vel azonosított Twitch Eventsub eseményfigyelő törlését végzi el. Elküld egy DELETE kérést a Twitch API-hoz az adott id-vel, hogy törölje az eseményfigyelőt a feliratkozások közül. Ha a törlés sikeres, akkor kiírja a konzolra, hogy a feliratkozás sikeresen törölve lett, ha pedig hiba történik, akkor a hibát is kiírja a konzolra.

A `getUserBearerToken` függvény az OAuth 2.0 hitelesítési kód alapján lekéri a felhasználóhoz tartozó hozzáférési token-t a Twitch API-tól. Elküld egy POST kérést a Twitch OAuth 2.0 token végpontjára, és a kérés paramétereként átadja a szükséges adatokat, mint például a kliens azonosítót, a kliens titkot, a hitelesítési kódot, a grant típust és a visszairányítási URI-t. Az eredményként visszakapott hozzáférési token-t adja vissza.

A `getUserInfo` függvény az adott hozzáférési token-t használva lekéri a felhasználóhoz tartozó információkat a Twitch API-tól. Elküld egy GET kérést a Twitch OAuth 2.0 userinfo végpontjára, és a kérés fejlécében átadja az autorizációs adatokat, beleértve az access token-t is. Az eredményként visszakapott felhasználói adatokat adja vissza.

Az osztály további metódusai, mint pl. `followEventHandler`, `subscribeEventHandler`, `reSubscriptionEventHandler` stb., amelyek az egyes esemény típusok eseménykezelőit valósítják meg. Ezek az eseménykezelők logolják az eseményeket, majd továbbítják azokat egy globális WebSocket csatornán a kliensnek.

#### **4.4.4. UserManagementService szolgáltatás**

A `UserManagementService` osztály a felhasználókkal foglalkozik, azon belül is a létrehozásával és azokhoz kapcsolódó eljárásokkal, módosításokkal és a kliens oldalnak nyújt biztonságos felhasználói folyamatot.

A `manageUser` metódus az adott felhasználó kezelését végzi el. Meghívja a `createUser` függvényt a felhasználó létrehozásához vagy frissítéséhez, majd feliratkoztatja a felhasználót.

nálót az előírt eseményekre a TwitchService segítségével. Végül csatlakozik a felhasználó Twitch csatornájához.

A `createUser` metódus az adott felhasználó létrehozását végzi el az adatbázisban a DAO segítségével. Ha a felhasználó már létezik az adatbázisban, akkor frissíti a felhasználó adatait a `updateUser` függvénnyel. Ha valamilyen hiba történik a felhasználó létrehozása vagy frissítése közben, hibát dob.

A `updateUser` metódus az adott felhasználó adatainak frissítését végzi el az adatbázisban a DAO segítségével. Ha valamilyen hiba történik a felhasználó frissítése közben, hibát dob.

Az `addRefreshTokenToUser` metódus az adott felhasználóhoz létrehoz és hozzáad egy refresh token-t az adatbázisban a DAO segítségével. Ha a felhasználónak már létezik refresh token-je az adatbázisban, akkor azt adja vissza. Ha valamilyen hiba történik a refresh token hozzáadása közben, hibát dob.

A `createAccessToken` metódus egy access token-t hoz létre a kapott refresh token alapján, amelyet a `jwt` (`jsonwebtoken`) modul segítségével aláír. Az access token-t adja vissza, és beállítja az érvényességi időt 600 másodpercre.

A `getCookie` metódus az adott sütit keresi meg a kapott süti neve alapján a kapott stringben. Ha megtalálja, akkor visszaadja annak értékét. Ha a süti nem található, `false` értéket ad vissza.

A `validateSession` függvény bemeneti paraméterei az `accessToken` és a `refreshToken`, amelyek az autentikációs folyamat során kapott tokenek. A függvény az `accessToken` validálását végzi el a `validateAccessToken` függvény segítségével. Ha az `accessToken` érvényes (nem járt le, valós aláírással rendelkezik stb.), akkor az eredeti `accessToken`-t adja vissza. Ha az `accessToken` nem érvényes, azaz validálási hiba történik, akkor a függvény az `accessToken` helyreállításához az eredeti `refreshToken`-t használja fel. Először megpróbálja lekérni a `refreshToken`-t az adatbázisból az új DAO (`Data Access Object`) objektum használatával. Ha a `refreshToken` megtalálható az adatbázisban, akkor az `accessToken` helyreállításához az eredeti `refreshToken`-t használja fel a `createAccessToken` függvény segítségével, majd visszaadja az újonnan generált `accessToken`-t. Ha a `refreshToken` nem található az adatbázisban, akkor a függvény 401-es HTTP hibakóddal tér vissza, ami azt jelzi, hogy az autentikáció sikertelen volt.

A `validateAccessToken`, az access token validálását végzi el a `jwt.verify` metódus segítségével, amely az access token érvényességét ellenőrzi az előre definiált `ACCESS TOKEN SECRET` titkos kulccsal. Ha az access token érvénytelen, azaz nem érvényes aláírással rendelkezik vagy lejárt, akkor hibát dob, amelyet az eredeti függvény továbbít a hívó félnek.

## **4.5. A vezérlő osztályok**

A vezérlő osztályok endpointok sokaságát foglalják magukba és ezek által tud kommunikálni a szerver oldali alkalmazás más alkalmazásokkal, ha HTTP kéréseket küldenek az alkalmazás felé.

### **4.5.1. AlertboxController vezérlő**

Az osztály fő feladata az Alertbox beállítások kezelése, többek között a hangüzenetek (TTS - Text-to-Speech) generálása és a beállítások mentése a AlertboxService szolgáltatással való kommunikáció révén.

Az `alertBoxFollowChange` metódus beállít egy útvonalat (endpoint-ot) a POST kérésre (`/editFollow`). Amikor ez az útvonalra érkezik kérés, a metódus a kérés testéből kiolvassza bizonyos adatokat, például a típust (`type`), az azonosítót (`id`), a hangos üzenet hangerőt (`ttsvolume`), képet (`image`), hangfájlt (`audio`), üzenetet (`message`), hangerőt (`volume`) és időtartamot (`duration`). Azután ezeket az adatokat továbbítja az `alertboxService.saveChannelPoint()` vagy az `alertboxService.saveAlertBox()` metódusoknak attól függően, hogy a típus `channelPoints` vagy más értéket kapott. Az eredménytől függően választ küld a kérőnek (status és message adattagokkal).

Az `alertBoxTTS` metódus beállít egy útvonalat (endpoint-ot) a GET kérésre (`/alertBoxTts`). Amikor ez az útvonalra érkezik kérés, a metódus kiolvassa a kérés URL paraméteréből a szöveget (`text`), majd létrehoz egy `gtts` objektumot a `gTTS` modul segítségével, és streamelni kezdi a generált hangfájlt. A generált hangfájl bufferét küldi vissza a kérőnek a megfelelő tartalmi típussal és hosszúsággal beállított válasszal.

### **4.5.2. CommandsController vezérlő**

Az osztály feladata parancsok kezelése, lekérdezése, törlése és módosítása az alkalmazás szerveroldali részén.

A `getCommands` metódus egy POST HTTP kérésre válaszol a `/commands` végponton. Az átvett HTTP kérés `req` objektumból kinyeri az adatot, amelyet JSON formátumban vár. Az adatban a `userid` és a `commandService` használatával lekérdezi az adott felhasználóhoz tartozó parancsokat. Az eredményt a válasz objektumon keresztül küldi vissza JSON formátumban, amely tartalmazza a `status` és a `data` mezőket.

A `deleteCommand` metódus egy POST HTTP kérésre válaszol a `/deleteCommand` végponton. Az átvett HTTP kérés `req` objektumból kinyeri az adatot, amelyet JSON formátumban vár. Az adatban található `userid` és `command` paramétereket felhasználva meghívja a `commandService` `deleteCommand` metódusát a parancs törlésére. A választ a `res` objektumon keresztül küldi vissza, ami egy egyszerű JSON objektum a `status` mezővel.

A `modifyCommand` metódus egy POST HTTP kérésre válaszol a `/modifyCommand` végponton. Az átvett HTTP kérés `req` objektumból kinyeri az adatot, amelyet JSON formátumban vár. Az adatban található `userid`, `command` és `message` paramétereket felhasználva meghívja a `commandService` `modifyCommand` metódusát a parancs módosítására. Ha a módosítás sikeres, akkor a válaszban egy egyszerű JSON objektumot küld vissza a `status` mezővel, ha hiba történik, akkor pedig a hibaüzenetet is tartalmazza a válasz.

### **4.5.3. TwitchController vezérlő**

Az osztály egy Twitch API események kezeléséért felelős kontroller osztályt implementál.

A `postTwitchEventSub` metódus az `/eventsub` útvonalon keresztül várja az érkező Twitch API EventSub eseményeket HTTP POST kérések formájában. Az `eventsub` végpontra érkező eseményeket ellenőrzi a Twitch API által használt HMAC algoritmus segítségével a `twitchService.getSecret`, `twitchService.getHmacMessage`, és `twitchService.verifyMessage` metódusokat használva. Az érvényesítés az események fejlécében található aláírással (`req.headers[constants.TWITCHMESSAGE_SIGNATURE]`) történik. Ha az események aláírása érvényes, akkor az osztály különböző típusú Twitch API Even-

tSub üzeneteket kezel, például értesítő (`constants.MESSAGETYPENOTIFICATION`), ellenőrzési (`constants.MESSAGETYPEVERIFICATION`), vagy visszavonási (`constants.MESSAGETYPEREVOCATION`) üzeneteket. Az értesítő üzenetek esetén az osztály feldolgozza az esemény adatait, és további műveleteket hajt végre a notification objektum alapján. Az ellenőrzési üzenetek esetén az osztály 200-as státuszkóddal és a `notification.challenge` értékével válaszol, ami egy ellenőrző kihívás a Twitch API-tól. A visszavonási üzenetek esetén az osztály feldolgozza az esemény adatait, és további műveleteket hajt végre a notification objektum alapján. Az ismeretlen típusú üzenetek esetén az osztály 204-es státuszkóddal válaszol, és hibaüzenetet ír ki a konzolra. Ha az események aláírása nem érvényes, akkor az osztály 403-as státuszkóddal válaszol, és hibaüzenetet ír ki a konzolra.

#### **4.5.4. UserManagementController vezérlő**

Ez az osztály felelős a felhasználókkal és a felhasználói munkamenettel kapcsolatos HTTP kérések kezeléséért.

A `userAuth` metódus egy HTTP POST kérést kezel, amely az `/auth` végponton érkezik. A kérés törzsében JSON formátumban várja a felhasználói adatokat, amelyeket a Twitch API-val való kommunikációhoz használ fel. Az access token és refresh token értékeket kinyeri a válaszból, majd létrehoz egy User objektumot az adatok alapján. Ezután meghívja a `userManagementService.manageUser` metódust az új felhasználó kezelésére, majd létrehoz egy session refresh token-t, majd létrehoz egy session access token-t. Végül HTTP sütit (cookie-t) küld a válaszban, amely tartalmazza a session refresh token-t, és elküldi a felhasználó adatait és az access token-t a válaszban.

A `deleteCookie` metódus egy HTTP POST kérést kezel, amely a `/deleteSession` végponton érkezik. Az endpoint válaszában törli a refresh token HTTP sütit, ezzel megszünteti a munkamenetet.

A `validateSession` metódus egy HTTP POST kérést kezel, amely a `/validateSession` végponton érkezik. A kérés törzsében JSON formátumban várja az access token-t, majd a kérés fejlécében található süti (cookie) alapján kinyeri a refresh token értékét. Ezután meghívja a `userManagementService.validateSession` metódust az access token és a refresh token ellenőrzésére. Ha az ellenőrzés sikeres, akkor elküldi az új access token-t a

válaszban.

## **4.6. Az AlertboxWebSocket**

Ez az osztály egy WebSocket szerver implementációt valósít meg egy Twitch Alert rendszer számára. Az osztály tartalmaz egy konstruktort, ami inicializálja a WebSocket szerver a 8000-es porton, és beállítja a kapcsolódott felhasználók listáját.

Az `initServer` metódus hozzáad eseménykezelőket a kapcsolatfelvétel, üzenet érkezés, és kapcsolat bontás eseményekre. Az osztály tartalmaz továbbá két aszinkron metódust: `sendChannelPointsEvent` és `sendEvent`, amelyek az értesítések küldését valósítják meg a kapcsolódott felhasználókhoz.

A `sendChannelPointsEvent` metódus fogadja az esemény típusát és az esemény objektumot, majd lekérdezi az adatbázisból az értesítési dobozhoz tartozó adatokat a DAO (Data Access Object) osztály használatával. Ha az adatok nem érhetők el az adatbázisból, akkor az alapértelmezett értesítési dobozhoz tartozó adatokat használja. Ezután összeállítja a küldendő adatokat az értesítési doboz és az esemény objektum alapján, majd az összeállított adatot JSON formátumban elküldi az összes olyan felhasználónak, akinek a broadcaster user id-je megegyezik az esemény objektumban található broadcaster user id-vel.

A `sendEvent` metódus hasonlóan működik, mint a `sendChannelPointsEvent` metódus, azonban az esemény típusától függően más adatokat kér le az adatbázisból, és más kulcsokat állít össze az összeállított adatban. Az elküldendő adatokat JSON formátumban elküldi az összes olyan felhasználónak, akinek a broadcaster user id-je megegyezik az esemény objektumban található broadcaster user id vagy to broadcaster user id értékkel. Az osztály az `AlertWebSocket` nevet viseli, és az `module.exports` segítségével exportálódik, hogy más modulok is importálhassák és használhassák azt.

## 5. fejezet

# Az alkalmazás kliens oldali részletes áttekintése

Az alkalmazás egy főoldalból, irányítópultból, kommand kezelő oldalból, esemény beállító oldalból és esemény megjelenítő oldalból áll. Ezek az oldalak különböző függvényeket, api hívásokat végeznek a megfelelő működésért.

### 5.1. Főoldal

A főoldalon lehetőség van bejelentkezni és regisztrálni az alkalmazásra a Get Started gomb lenyomásával.

#### 5.1.1. Bejelentkezés és regisztráció

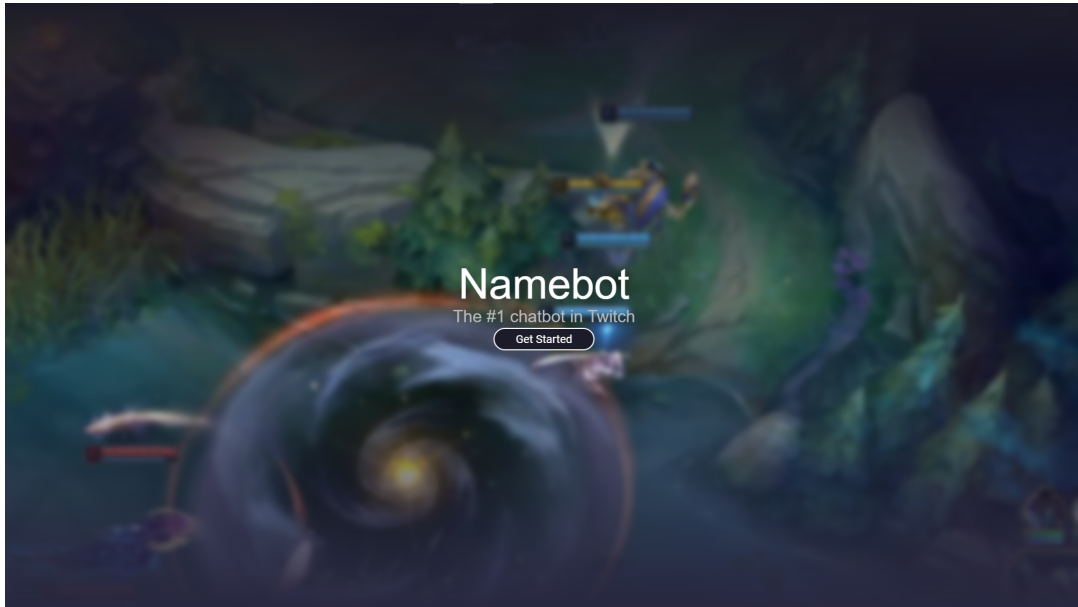
A bejelentkezés és regisztráció egy folyamat, ami Twitch autentikációval van megoldva, ahol a felhasználónak csak fel kell hatalmazni az alkalmazást, hogy használja az adatait és hozzáférjen funkciókhoz a fiókjából. Miután ez megtörtént visszairányít a /auth oldalra és elvégzi az alkalmazás a bejelentkezést az login függvény segítségével.

Ez a fájl annyit tesz, hogy meghívja az AuthenticationService.login függvényét majd visszairányít a főoldalra. A függvény egy HTTP POST kérést indít a megfelelő szerveroldali címre (/auth) a code értékkel, amit a Twitch nyújt, mint token a felhasználóhoz. Ha a kérés sikeresen lefut, akkor az access token, username, picture és id kulcsokat a válaszból



kiolvasott adatokkal elmenti a localStorage-ba, amely egy olyan tartós adattároló, amely a webböngészőben elérhető és a későbbi látogatások során is elérhető marad.

Bejelentkezés után a Get Started gomb átváltozik Go to Dashboard gombra.



5.1. ábra. Az alkalmazás főoldala

### **5.1.2. Navigációs sáv**

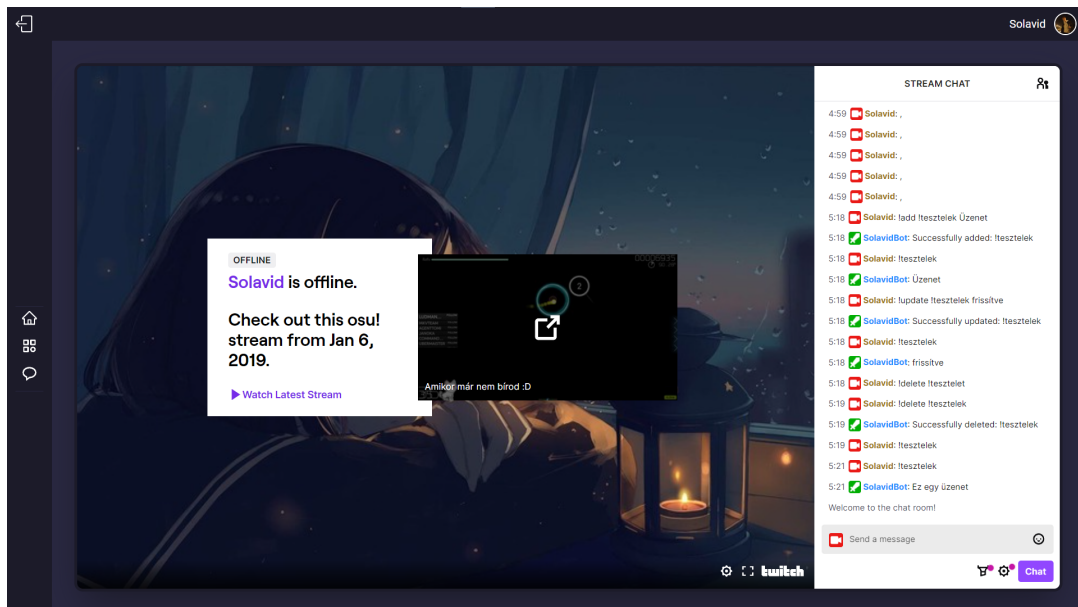
A bejelentkezés után megjelenik egy navigációs sáv, ezen a navigációs sávon megjelenik a Twitch-től elkért profil kép és egy kijelentkezés gomb. Ez a gomb meghív egy függvényt ami egy HTTP POST kérést hajt végre a szerver oldal "/deleteSession" URL-re. A kérés elküldi a sütiket a szervernek és törli a refresh token-t közülük, ezután az access token, username, picture és id nevű elemeket eltávolítja a localStorage-ból, ami a felhasználói adatok törléséhez szükséges. Ha ezek megtörténtek a program visszairányít a főoldalra, eltűnik a navigációs sáv és kijelentkeztet.

## **5.2. Irányítópult**

Az irányítópulton 3 darab oldalt találunk ezek pedig az irányítópult alap oldala más néven nyitóoldal, kommand kezelő oldal és az esemény beállító oldal. Emellett egy oldalsó sáv is látható, ami a navigációt teszi lehetővé az oldalak között.

### 5.2.1. Nyitóoldal

A felhasználó ezen az oldalon az élő adását tudja nyomon követni és az üzenőfalát. Ezt a két komponenst a Twitch biztosítja, ezáltal csak kettő iframe-re van szükség a beillesztéshez.



5.2. ábra. Az alkalmazás irányítópultja

### 5.2.2. Kommand kezelő oldal

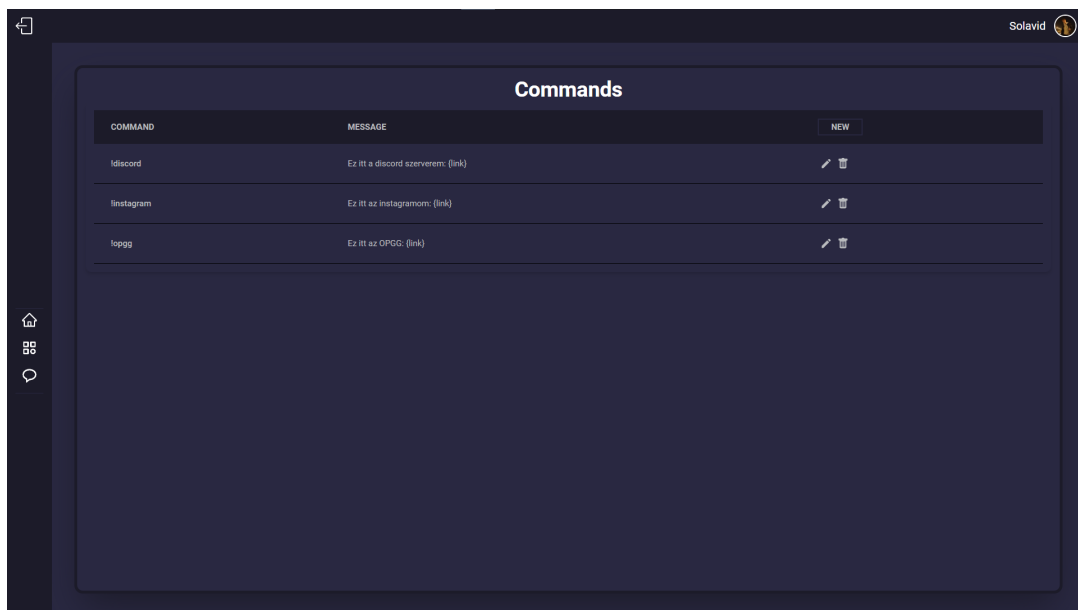
A felhasználó ezen az oldalon tudja kezelni, vagyis létrehozni, módosítani és törölni a parancsait, amit a Twitch üzenőfalán fog feldolgozni a chatbot a Twitch felhasználók meghívására. Egy táblázatban találhatóak a létező parancsok és a fentebb említett funkciók, minden parancsra szétosztva és ezek a következő képpen működnek.

A megfelelő gombra vagy ikonra kattintva a szerver oldal felé a megadott adatokkal egy HTTP POST kérés lesz küldve, ami ha sikeres elvégzi a megadott feladatot.

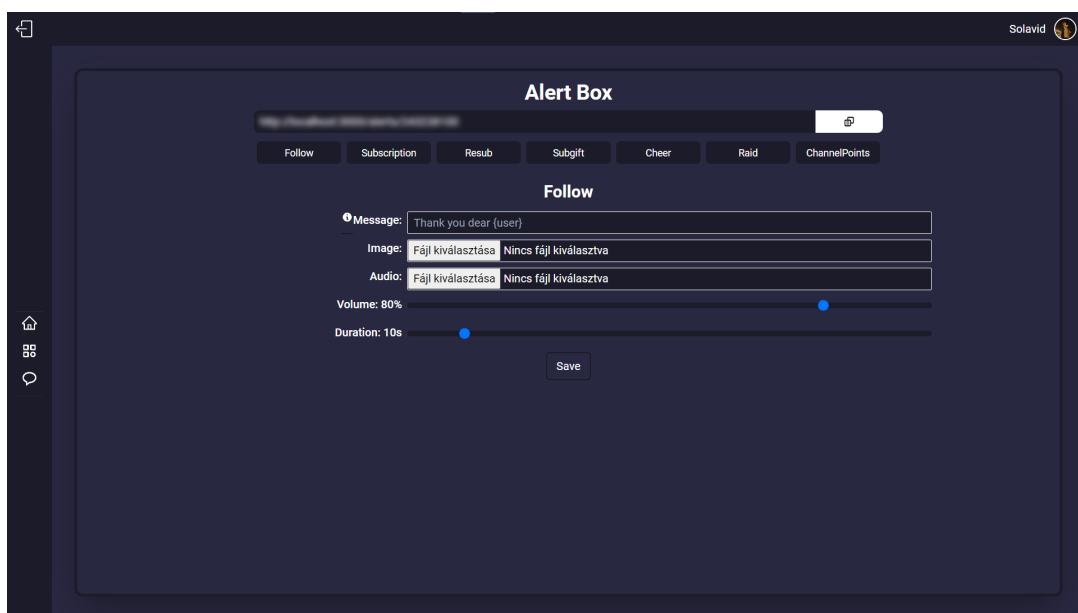
### 5.2.3. Eseménykezelő oldal

A felhasználó ezen az oldalon tudja beállítani azt, hogy az adott Twitch események érkezésekor, milyen hangfájl, képfájl és szöveg jelenik meg a képernyőn és még azt, hogy mennyi ideig lesznek megjelenítve és milyen hangosan. Ezeket minden típusra külön be

lehet állítani. A mentés a Save gombra kattintva működik és az adatokat továbbítja a szerver oldalnak az egy HTTP POST kéréssel.



5.3. ábra. Az alkalmazás parancs kezelő oldala



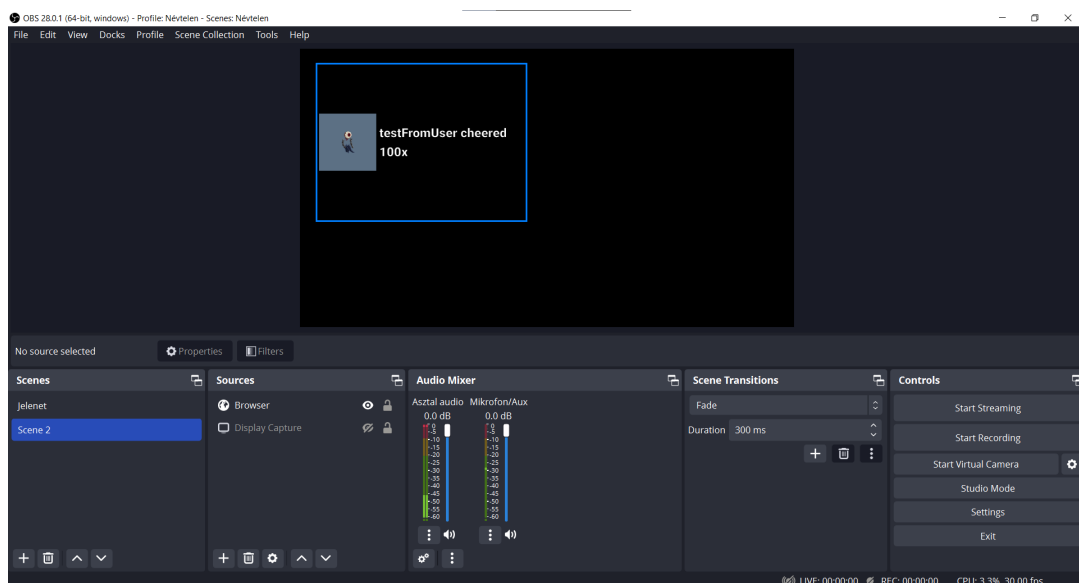
5.4. ábra. Az alkalmazás esemény kezelő oldala

### 5.3. Esemény megjelenítő

Az esemény megjelenítő a /alerts/userid címen érhető el, aminek a feladata, hogy minden felhasználónak megjelenítse az értesítést a Twitch-től érkező eseményekről és azokat feldolgozza a megfelelő módon. A komponens a következő függvényeket használja.

A connectToWebSocket függvény kapcsolódik a WebSocket szerverhez, és kezeli az onopen és onmessage eseményeket. A decideEventType függvény az érkező üzenet típusa alapján dönti el, hogy milyen feldolgozást hajtson végre. A sayTtsForChannelPoints függvény hangot játszik le a kapott üzenet alapján. A showAlertBox függvény megjelenít egy figyelmeztető ablakot a kapott adatok alapján. Az editText függvény az üzenetben található helyettesítendő változók cseréjét csinálja az aktuális esemény adataira. A getSoundData függvény hang objektumot és ennek az objektumnak a betöltését és végét figyeli.

A csatlakozásért a useEffect felelős, ami az inicializálási fázisban meghívja a connectToWebSocket függvényt ezáltal képes működni a figyelmeztetés.



5.5. ábra. Az alkalmazás esemény megjelenítője az Open Broadcaster Softwareben

### 5.4. Privát oldalak

Az említett oldalak között az irányítópulton lévő oldalak nem publikusak, ezért privát útvonalakat használ az alkalmazás. Ezek az útvonalak a szerver oldal fele küldenek egy

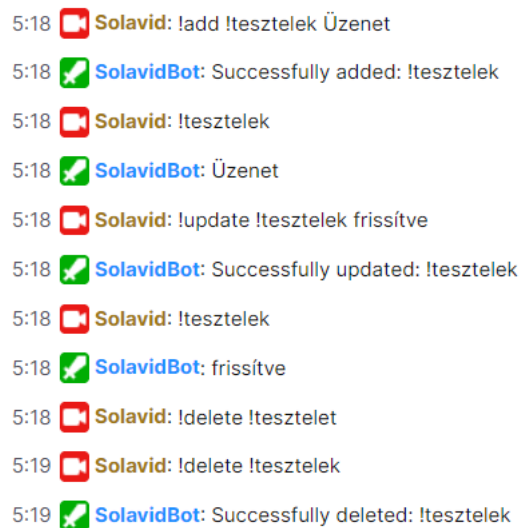
HTTP POST kérést a /validateSession útvonalra minden megnyitáskor és ha elfogadja a szerver oldal a jelentkezést, akkor folytathatja tovább a felhasználó a böngészést a web-oldalon.

## 6. fejezet












### Chatbot funkciói

A chatbot alapvető parancsokkal van felruházva, amiket az adott üzenőfali moderátorok vagy közvetítő tud használni, ilyen a parancs létrehozása, a parancs módosítása és a parancs törlése. Ezeket a parancsokat a következőképpen lehet meghívni Az új parancs létrehozása: `!add parancs név parancs üzenet`, a parancs módosítása: `!update parancs név parancs üzenet` és a parancs törlése: `!delete parancs név`.

A chatbot ezen kívül a tartalmazza a közvetítő vagy moderátorok által létrehozott parancsokat, ezek a parancsok arra az üzenőfalra szólnak amiken létre lettek hozva.



The screenshot displays a chat log with the following messages:

- 5:18  **Solavid:** !add Itesztelek Üzenet
- 5:18  **SolavidBot:** Successfully added: Itesztelek
- 5:18  **Solavid:** Itesztelek
- 5:18  **SolavidBot:** Üzenet
- 5:18  **Solavid:** !update Itesztelek frissítve
- 5:18  **SolavidBot:** Successfully updated: Itesztelek
- 5:18  **Solavid:** Itesztelek
- 5:18  **SolavidBot:** frissítve
- 5:18  **Solavid:** !delete Itesztelek
- 5:19  **Solavid:** !delete Itesztelek
- 5:19  **SolavidBot:** Successfully deleted: Itesztelek

6.1. ábra. A chatbot parancsai

## **7. fejezet**

# **További fejlesztési lehetőségek**

### **7.1. Adakozás funkció**

Az adakozás funkció nézőktől a közvetítő felé, ez növelheti az interakciót és a nézők kifejezhetik tetszésüket vagy akár megélhetést biztosíthatnak a közvetítő számára. Ezt a funkciót az eseményekkel össze lehet kötni és itt is üzenetet hangot lehet lejátszani.

### **7.2. Automatikus üzenetek**

Az automatikus üzenetek a chatbot által az üzenőfalakra amelyek időzítve jelenítenek meg üzeneteket, ezzel automatikus hirdetésre is lehet használni, ezáltal több monoton feladattól lehet megszabadítani a közvetítőt.

### **7.3. Személyre szabhatóbb események**

A személyre szabás már így is nagy mértékben lehetséges az alkalmazásban, de ezt még lehet fokozni. A szövegek színeinek változtatása eseményenként, a szöveg és kép pozíciójának szabadkézi beállítása és több fajta animáció hozzáadása.

## 8. fejezet

# Összefoglaló

A szakdolgozat fő célja egy közvetítés segítő és fejlesztő alkalmazás kifejlesztése a Twitch.tv platformhoz. Az alkalmazás olyan funkciókat tartalmaz, amelyek szükségesek egy közvetítés segítő és fejlesztő alkalmazáshoz. Az alkalmazás rendelkezik egy felhasználóbarát felhasználói felülettel, amely lehetővé teszi a felhasználók számára, hogy személyre szabják a Twitch által küldött eseményekre való figyelmeztetéseket, megjelenítsék ezeket a figyelmeztetéseket, nyomon kövessék az üzenőfalukat és az élő adásukat, valamint könnyen létrehozzanak, töröljenek és módosítsanak parancsokat a chatbot-hoz.

Az alkalmazás emellett tartalmaz egy szerver oldali alkalmazást, amely felelős a logika, a biztonság és az adatok kinyeréséért az adatbázisból a kliens oldal és a chatbot számára. Az adatbázis megfelelően struktúrált tárolási formát biztosít az adatoknak, hogy könnyen kezelhetők legyenek. A chatbot az alkalmazás egyik kulcsfontosságú eleme, amely megszünteti a monotonitást és könnyen kezelhető és személyre szabható válaszadási formát kínál a közvetítőknél.

Bár nem könnyen reprodukálható az alkalmazás, valamint felépítése komplex, mégis jó iránymutatást nyújt, ha valaki hasonló alkalmazást szeretne készíteni, mivel kevés anyag található róla az interneten vagy könyvekben. A szakdolgozat kidolgozza az alkalmazás működését, valamint bemutatja az alkalmazás funkcionalitását és használatát a felhasználók számára. Emellett a szakdolgozat javaslatokat tesz a további fejlesztési lehetőségekre és az alkalmazás jövőbeli fejlesztési irányaira. Az alkalmazás sikeres megvalósítása és a szakdolgozat részletes leírása segíthet más fejlesztőknek hasonló alkalmazások készítésében, és hozzájárulhat a közvetítések fejlődéséhez.



# Irodalomjegyzék

- [1] Firebase. <https://firebase.google.com/docs>. Utolsó megtekintés: 2023.03.26.
- [2] Github. <https://docs.github.com/en>. Utolsó megtekintés: 2022.11.14.
- [3] Ngrok. <https://ngrok.com/docs/>. Utolsó megtekintés: 2023.01.03.
- [4] Nodejs. <https://nodejs.org/en/docs>. Utolsó megtekintés: 2023.01.04.
- [5] Postman. <https://www.postman.com/api-documentation-tool/>. Utolsó megtekintés: 2023.03.11.
- [6] React. <https://legacy.reactjs.org/docs/getting-started.html>. Utolsó megtekintés: 2023.01.27.
- [7] Tailwind. <https://v2.tailwindcss.com/docs>. Utolsó megtekintés: 2023.04.10.
- [8] Twitch api. <https://dev.twitch.tv/docs/api/>. Utolsó megtekintés: 2023.03.23.
- [9] Twitch cli. <https://dev.twitch.tv/docs/cli/>. Utolsó megtekintés: 2023.01.29.
- [10] Visual studio code. <https://code.visualstudio.com/docs>. Utolsó megtekintés: 2022.10.19.
- [11] Webstorm. <https://www.jetbrains.com/help/webstorm/meet-webstorm.html>. Utolsó megtekintés: 2022.10.15.

# Nyilatkozat

Alulírott Fehér Erik programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2023. május 9.

.....Fehér Erik.....  
aláírás