Önálló laboratórium beszámoló

Távközlési és Mesterséges Intelligencia Tanszék

Készítette:	**Buga Péter**
Neptun-kód:	**G50RDF**
Ágazat:	**Intelligens Hálózatok**
E-mail cím:	**buga.peti@gmail.com**
Konzulens(ek):	**Dr. Maliosz Markosz, Dr. Simon Csaba**
E-mail címe(ik):	**maliosz@tmit.bme.hu, simon@tmit.bme.hu**

Live streaming szolgáltatás kialakítása konténeralapú környezetben

Feladat

A feladat célja egy élő videó közvetítő (streaming) szolgáltatás fejlesztése konténerizált mikroszolgáltatás-architektúrában, Kubernetes környezetben. A hallgatónak ki kell alakítania a rendszer fő komponenseit (pl. stream kezelés, felhasználókezelés, nézőszámlálás), és biztosítania kell azok együttműködését konténerekben.

2024/2025. 2. félév

1. A laboratóriumi munka környezetének ismertetése, a munka előzményei és kiindulási állapota

1.1 Bevezető

Napjainkban a streaming szolgáltatások szinte mindenhol jelen vannak az életünkben. Gondoljunk csak a szórakozásra, mint a Netflix vagy a YouTube, az oktatásra (online kurzusok, előadások közvetítése) vagy akár az ipari felhasználásra (távoli megbeszélések, folyamatok monitorozása). Az, hogy videó- és audiotartalmakat valós időben tudunk továbbítani az interneten keresztül, alapvetővé vált.

Ezeknek a komplex rendszereknek a működtetése komoly informatikai hátteret igényel. Itt jönnek képbe a modern IT megoldások, mint a felhőalapú rendszerek és a konténerizáció. A felhő lehetővé teszi, hogy rugalmasan, igény szerint használjunk szervereket és erőforrásokat, anélkül, hogy saját drága gépeket kellene vennünk és üzemeltetnünk. A konténerizáció pedig olyan, mint egy digitális "dobozolás": az alkalmazásainkat minden szükséges összetevővel együtt csomagoljuk be, így azok könnyen és megbízhatóan futtathatók szinte bármilyen számítógépen vagy szerveren. Ez nagyban megkönnyíti a fejlesztést és az üzemeltetést.

Ennek a laboratóriumi munkának a célja egy saját, egyszerűsített streaming rendszer alapjainak létrehozása volt. A projekt során arra törekedtem, hogy megismerkedjek a streaming technológiák működésével, és kipróbáljam, hogyan lehet egy ilyen rendszert modern eszközökkel, mikroszolgáltatás-architektúra (azaz kisebb, önálló részekre bontott alkalmazás) és konténerizáció (Docker, Kubernetes) segítségével felépíteni.

A motivációmat az adta, hogy gyakorlati tapasztalatot szerezzek ezekkel a napjainkban kulcsfontosságú technológiákkal kapcsolatban.

1.2 Elméleti összefoglaló

Ahhoz, hogy megértsük a projekt lényegét és az alkalmazott megoldásokat, fontos áttekinteni azokat az alapvető technológiákat és koncepciókat, amelyekre építkezhetünk. Ez a fejezet bemutatja a streaming, a mikroszolgáltatások, a konténerizáció és a felhőalapú rendszerek világát, kontextusba helyezve az elvégzett munkát.

1.2.1 Streaming technológiák áttekintése

Az "élő streaming" lényege, hogy audio- vagy videótartalmat közel valós időben továbbítunk egy forrástól (pl. kamera, képernyő) egy vagy több néző felé az interneten keresztül. Ez eltér a hagyományos letöltéstől, ahol a teljes fájlt le kell tölteni a lejátszás előtt.

A streaming megvalósításához különböző protokollokat használnak, amelyek meghatározzák az adatok csomagolásának és továbbításának módját. Néhány elterjedt protokoll:

- **RTMP (Real-Time Messaging Protocol):** Eredetileg az Adobe Flash Playerhez fejlesztették ki. Ma főként a tartalom feltöltésére (ingestion) használják a streaming szerver felé, mivel alacsony késleltetést biztosít. Böngészőben való közvetlen lejátszása azonban visszaszorult. [1]

- **HLS (HTTP Live Streaming):** Az Apple által kifejlesztett protokoll. A videót kis, letölthető szegmensekre bontja, amelyeket szabványos HTTP protokollon keresztül továbbít. Előnye a nagyfokú kompatibilitás (szinte minden modern eszköz és böngésző támogatja) és az adaptív bitráta (a videó minősége a néző internetkapcsolatának sebességéhez igazodik). Hátránya a magasabb késleltetés (több tíz másodperc is lehet) a pufferelés miatt. [2]

A streaming rendszerek tervezésekor kulcsfontosságú szempontok:

- **Késleltetés (Latency):** Az az idő, ami eltelik az esemény rögzítése és a néző képernyőjén való megjelenése között. Az elfogadható mértéke az alkalmazástól függ (pl. egy interaktív közvetítésnél kritikus, egy film streamingnél kevésbé).

- **Sávszélesség (Bandwidth):** A rendelkezésre álló adatátviteli kapacitás. A streaming jelentős sávszélességet igényel mind a feltöltő, mind a néző oldalán. Az adaptív streaming (mint a HLS esetén) segít kezelni a változó sávszélességet.

- **Skálázhatóság (Scalability**): A rendszer képessége arra, hogy nagyszámú egyidejű nézőt is zökkenőmentesen ki tudjon szolgálni. Ez komoly infrastrukturális kihívást jelent.

1.2.2 Mikroszolgáltatás-architektúra és konténerizáció

A szoftverfejlesztésben két fő megközelítés létezik a rendszerek felépítésére:

- **Monolitikus architektúra:** A teljes alkalmazás egyetlen, nagy egységként működik. Kezdetben egyszerűbb lehet fejleszteni, de nehézkessé válik a módosítása, tesztelése, skálázása és az új technológiák bevezetése, ahogy a rendszer növekszik.

- **Mikroszolgáltatás-alapú architektúra:** Az alkalmazást kisebb, önállóan fejleszthető, telepíthető és skálázható szolgáltatásokra (mikroszolgáltatásokra) bontják. Minden szolgáltatás egy konkrét üzleti képességért felelős (pl. felhasználókezelés, videófeldolgozás, chat). Ez a megközelítés rugalmasabb, jobban skálázható (csak a szükséges részeket kell skálázni), és lehetővé teszi különböző technológiák használatát az egyes szolgáltatásokhoz. [3]

A mikroszolgáltatások kezelésének és futtatásának hatékony módja a konténerizáció. A konténertechnológiák, mint a népszerű Docker, lehetővé teszik, hogy az alkalmazásokat és azok függőségeit (könyvtárak, futtatókörnyezet) egy izolált csomagba, ún. konténerbe zárjuk. Ez biztosítja, hogy az alkalmazás ugyanúgy fusson a fejlesztő gépén, a tesztkörnyezetben és az éles rendszerben is. A containerd egy alacsonyabb szintű futtatókörnyezet, amelyet gyakran a Docker is használ a háttérben.

Amikor sok konténert kell kezelni (telepíteni, skálázni, frissíteni, hálózatukat menedzselni), szükség van egy konténer-orkesztrációs eszközre. A legelterjedtebb ilyen rendszer a Kubernetes (K8s). A Kubernetes automatizálja a konténerizált alkalmazások életciklusának kezelését. Főbb építőkövei: [4]

- **Pod:** A Kubernetes legkisebb telepíthető egysége, ami egy vagy több szorosan kapcsolt konténert tartalmazhat.

- **Service:** Absztrakció, amely egy logikai Pod-csoportot és egy hozzáférési házirendet definiál (pl. stabil IP címet biztosít egy változó számú Pod-halmaznak).

- **Deployment:** Leírja az alkalmazás kívánt állapotát (pl. hány példány fusson egy Pod-ból), és kezeli a Pod-ok frissítését (pl. gördülő frissítés - rolling update).

- **Ingress:** Kezeli a klaszteren kívülről érkező HTTP/HTTPS kéréseket, és irányítja azokat a megfelelő Service-ekhez.

Ezek az eszközök (mikroszolgáltatások, Docker, Kubernetes) együttesen lehetővé teszik komplex, skálázható és rugalmas rendszerek építését, mint amilyen egy streaming platform is.

1.2.3 Felhőalapú infrastruktúra

A modern alkalmazásokat, különösen a nagy skálázhatóságot igénylőket, gyakran felhőalapú infrastruktúrán (Cloud Infrastructure) futtatják (pl. Amazon Web Services - AWS, Google Cloud Platform - GCP, Microsoft Azure). A felhő számos előnyt kínál:

- **Dinamikus erőforrás-allokáció és automatizálás:** Lehetőség van az erőforrások (számítási kapacitás, tárhely, hálózat) igény szerinti, akár automatikus növelésére vagy csökkentésére. Ez költséghatékony, mivel csak a ténylegesen használt erőforrásokért kell fizetni, és biztosítja, hogy a rendszer elbírja a terhelési csúcsokat (pl. sok néző csatlakozik egy élő adáshoz). A Kubernetes kiválóan integrálódik a felhős környezetekkel ezen automatizálás megvalósításához. [5]

- **Magas rendelkezésre állás és hibatűrés:** A felhőszolgáltatók általában több adatközponttal rendelkeznek, ami lehetővé teszi redundáns, hibatűrő rendszerek építését.

- **Managed Services:** Számos kész szolgáltatást kínálnak (adatbázisok, üzenetsorok, terheléselosztók), amelyek leveszik a komplex infrastruktúra menedzselésének terhét a fejlesztőkről.

A biztonság kiemelt fontosságú minden rendszerben, különösen a felhőben. Az autentikáció (a felhasználó azonosítása) és az autorizáció (a felhasználó jogosultságainak ellenőrzése) alapvető követelmény. Elosztott rendszerekben, mint a mikroszolgáltatások, gyakran használnak szabványos megoldásokat:

- **JWT (JSON Web Token):** Egy kompakt és biztonságos módszer az információk (pl. felhasználói azonosító, jogosultságok) átadására a felek között egy digitálisan aláírt token formájában. Gyakran használják API-k védelmére és a szolgáltatások közötti autentikációra. A JWT (JSON Web Token) egy nyílt szabvány (RFC 7519). [6]

- **OAuth (Open Authorization):** Egy nyílt protokoll, amelyet elsősorban delegált autorizációra használnak, azaz arra, hogy egy alkalmazás hozzáférést kapjon egy felhasználó erőforrásaihoz egy másik szolgáltatónál (pl. "Bejelentkezés Google fiókkal") anélkül, hogy a jelszavát megosztaná. [7]

Ezek az elméleti alapok teremtik meg a szükséges hátteret a projekt során alkalmazott technológiák és a rendszerterv megértéséhez. A következő fejezetekben bemutatom, hogyan alkalmaztuk ezeket az elveket a saját streaming rendszer megtervezése és megvalósítása során.

1.3 A munka állapota, készültségi foka a félév elején

A félév elején a projektet nem teljesen üres lappal kezdtem, mivel rendelkezem már némi alapszintű tapasztalattal a modern webfejlesztés néhány kulcsfontosságú területén. Korábbi tanulmányaim és kisebb projektjeim során volt alkalmam megismerkedni a .NET keretrendszerrel backend oldali alkalmazások készítéséhez, valamint az Angular keretrendszerrel felhasználói felületek (frontend) fejlesztéséhez. Emellett tisztában voltam a Kubernetes alapjaival is, értettem a konténer-orkesztráció fontosságát és az alapvető koncepciókat, mint a Podok és Service-ek szerepét. Ez a meglévő tudás biztosította a szükséges alapot ahhoz, hogy belevágjak egy komplexebb, mikroszolgáltatásokra épülő streaming rendszer kidolgozásába.

Konkrétan ehhez a streaming témához kapcsolódóan a tanszéken korábban nem végeztem munkát, és nem volt tudomásom olyan közvetlen előzményprojektről sem, amelynek a kódját vagy specifikus eredményeit fel tudtam volna használni. Így a feladat ebből a szempontból új kihívást jelentett. A tanszéki támogatás elsősorban az általános iránymutatásban és a szükséges infrastruktúra (például a fejlesztői környezet elemeihez való hozzáférés vagy tanácsok) biztosításában nyilvánult meg, konkrét, előre elkészített streaming-specifikus mintakódot vagy komponenst nem kaptam a munkához.

A fejlesztés megkezdése előtt kulcsfontosságú volt a megfelelő fejlesztői környezet előkészítése. Ez magában foglalta a Docker Desktop telepítését a konténerek helyi létrehozásához és futtatásához, egy helyi Kubernetes klaszter indítását szimulálására, valamint a Visual Studio Code (VS Code) kódszerkesztő konfigurálását a backend és frontend kódok írásához és hibakereséséhez. Ezek az eszközök tették lehetővé, hogy a teljes rendszert – a backend szolgáltatásoktól a frontendig, beleértve a konténerizációt és a Kubernetes deploymentet – a saját gépemen tudjam fejleszteni és tesztelni.

2. Az elvégzett munka és eredmények ismertetése

2.1 Tervezett funkciók és felhasználás

A rendszer célja, hogy a felhasználók élő videó közvetítéseket indíthassanak, nézhessenek, és interakcióba léphessenek egymással egy webes felületen keresztül. Az alkalmazás tervezett fő funkciói a következők:

- **Regisztráció és bejelentkezés:** A felhasználók saját fiókot hozhatnak létre, majd bejelentkezhetnek
- **Élő stream indítása:** A bejelentkezett felhasználók saját élő közvetítést indíthatnak. Ehhez a rendszer egyedi stream kulcsot generál, amit például OBS-be lehet beállítani.
- **Streamek böngészése és nézése:** Minden látogató böngészheti az aktuálisan élő közvetítéseket. A nézők valós időben láthatják a nézőszámot és a stream adatait.
- **Stream ajánlások:** A rendszer ajánl élő streameket a felhasználónak, például népszerűség vagy követések alapján.
- **Felhasználói profil:** Minden felhasználó szerkesztheti saját adatait, például a nevét, e-mail címét vagy jelszavát.
- **Követés:** A felhasználók másokat követhetnek. Az oldalsó menüben látható, hogy a követett streamerek közül ki van éppen élőben.
- **Nézőszámlálás:** A rendszer valós időben számolja, hogy egy adott streamet hányan néznek.
- **Stream beállítások:** A felhasználók módosíthatják saját streamjük nevét, leírását, kategóriáját, és új stream kulcsot is generálhatnak.
- **Biztonság:** A rendszer minden védett művelethez JWT alapú hitelesítést használ.

A felhasználók a frontend Angular alkalmazáson keresztül érik el a szolgáltatásokat. A backend mikroszolgáltatások biztosítják az adatok kezelését és a valós idejű funkciókat. A rendszer skálázható, minden fő komponens külön konténerben fut Kubernetes klaszterben. Az adatokat egy relációs adatbázis kezelőben tárolja és egy kulcs-érték gyorsítótárat használ.

2.2 Architektúra

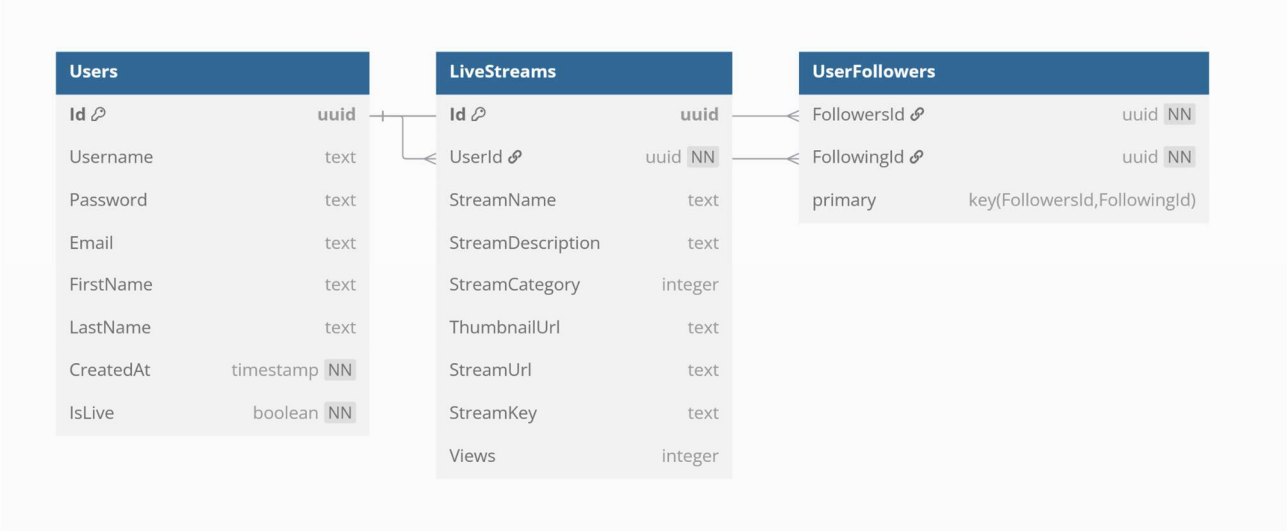
Az elkészült streaming rendszer architektúrájának kialakításakor a fő szempont az volt, hogy a különböző funkciók (például a videóközvetítés, a felhasználókezelés vagy a nézőszámlálás) egymástól függetlenül, mégis szorosan együttműködve működjenek. Ezért a rendszer több, jól elkülöníthető részből épül fel, amelyek mindegyike egy-egy konkrét feladatot lát el. Ezek a komponensek önállóan fejleszthetők, frissíthetők és szükség esetén külön-külön is skálázhatók, így a teljes szolgáltatás könnyen igazítható a felhasználói igényekhez vagy a terheléshez. Az egyes részek közötti kommunikáció szabványos protokollokon keresztül történik, ami egyszerűsíti a fejlesztést és a hibakeresést is. Az architektúra kialakítása során törekedtem arra, hogy a rendszer átlátható, bővíthető és hosszú távon is jól karbantartható legyen.

2.3 Adatstruktúra

A rendszer adatbázisának felépítése három fő részből áll (1.ábra), amelyek együtt biztosítják a streaming szolgáltatás alapvető működését. Az első rész a Users tábla, amely minden felhasználóról tárolja a legfontosabb információkat, például a felhasználónevet, e-mail címet, jelszót, valamint azt is, hogy mikor regisztrált és éppen élőben közvetít-e. Ez a tábla minden felhasználóhoz egyedi azonosítót rendel, ami alapján később könnyen összekapcsolhatók a többi adattal.

A második rész a LiveStreams tábla, amely az élő közvetítéseket írja le. Itt minden streamhez tartozik egy azonosító, valamint meg van adva, hogy melyik felhasználó indította azt (UserId). Emellett tárolja a stream nevét, leírását, kategóriáját, a hozzá tartozó képet (thumbnail), a stream elérési útját, kulcsát, valamint a nézettségi adatokat. Ez lehetővé teszi, hogy minden egyes közvetítéshez részletes információkat rendeljünk, és könnyen visszakereshető legyen, hogy ki, mikor és milyen tartalmat közvetített.

A harmadik rész a UserFollowers tábla, amely a felhasználók közötti követési kapcsolatokat tartalmazza. Ez azt jelenti, hogy minden sorban két felhasználó azonosítója szerepel: az egyik a követő, a másik pedig az, akit követnek. Így egyszerűen nyilvántartható, hogy ki kit követ, és ennek alapján megvalósíthatók a közösségi funkciók, például a követett streamerek listázása.



1. ábra
Az adatbázisban használt táblák és ezek összeköttetése

2.4 Használt technológiák

A rendszer fejlesztése során modern, ipari szabványokat követő technológiákat választottam, amelyek lehetővé teszik a skálázható, megbízható és könnyen karbantartható streaming platform kialakítását. Az alábbiakban röviden bemutatom a fő technológiákat és azok előnyeit:

- **.NET:** Egy modern, nyílt forráskódú fejlesztői keretrendszer, amelyet főként szerveroldali alkalmazások és mikroszolgáltatások készítésére használnak. Lehetővé teszi a nagy teljesítményű, stabil és jól skálázható backend rendszerek fejlesztését, és széles körben támogatott az iparban. [8]

- **Angular:** Egy fejlett, nyílt forráskódú frontend keretrendszer, amelyet a Google fejleszt. Segítségével könnyen lehet reszponzív, dinamikus webes felületeket létrehozni, főként TypeScript nyelven. Az Angular támogatja a komponens-alapú és moduláris fejlesztést, így ideális nagyobb webes projektekhez. [9]

- **PostgreSQL:** Egy nyílt forráskódú relációs adatbázis-kezelő rendszer, amely a tartós adatok megbízható tárolását biztosítja. Fejlett funkciókat kínál, jól skálázható, és elterjedt választás üzleti alkalmazásokban is. [10]

- **Redis:** Egy memóriában futó kulcs-érték típusú adatbázis, amelyet elsősorban gyorsítótárként és valós idejű adatok kezelésére használnak. Kiemelkedően gyors adatkezelést tesz lehetővé, ezért gyakran alkalmazzák olyan rendszerekben, ahol fontos a gyors válaszidő. [11]

- **Nginx-RTMP:** Egy streaming szerver, amely az RTMP protokoll segítségével fogadja a videófolyamokat, majd HLS formátumban továbbítja azokat a nézők felé. Stabil működést és alacsony késleltetést biztosít, ezért népszerű választás élő közvetítésekhez. [12] [13]

- **Shaka Player:** Egy nyílt forráskódú JavaScript alapú lejátszó, amely lehetővé teszi HLS és más modern streaming formátumok böngészőben történő lejátszását. Platformfüggetlen, és támogatja a korszerű streaming funkciókat. [14]

- **Docker:** Egy konténerizációs platform, amely lehetővé teszi az alkalmazások és szolgáltatások izolált, hordozható futtatását. Segítségével könnyen kezelhető, egységes környezetben futtathatók a fejlesztett komponensek. [15]

- **Kubernetes:** Egy konténer-orkesztrációs rendszer, amely automatizálja a konténerek telepítését, skálázását és menedzselését. Lehetővé teszi a magas rendelkezésre állású, rugalmas és könnyen üzemeltethető rendszerek kialakítását. [4]

2.5 Rendszer felépítés

A streaming rendszer több, egymással együttműködő mikroszolgáltatásból (service-ből) áll (2. ábra), amelyek mindegyike egy-egy jól körülhatárolható funkcióért felelős. Ezek a szolgáltatások külön konténerekben futnak, és szabványos HTTP API-kon keresztül kommunikálnak egymással. Az alábbiakban röviden bemutatom az egyes fő komponensek szerepét.

Backend

1. **AuthService (Hitelesítés)** Ez a szolgáltatás felelős a felhasználók bejelentkeztetéséért és a jogosultságok kezeléséért. A felhasználó itt kap egy JWT tokent, amivel a többi szolgáltatás védett végpontjait is elérheti. A token-alapú hitelesítés biztosítja, hogy csak jogosult felhasználók férhetnek hozzá a rendszer funkcióihoz.

2. **UserDbHandler (Felhasználókezelés)** Ez a modul tárolja és kezeli a felhasználók adatait (pl. név, e-mail, jelszó, regisztráció dátuma). Lehetővé teszi új felhasználók regisztrációját, adataik módosítását, valamint azt is, hogy egy felhasználó éppen élőben közvetít-e.

3. **StreamDbHandler (Streamek kezelése)** Ez a szolgáltatás felelős az élő közvetítések (streamek) adatainak tárolásáért és kezeléséért. Itt jönnek létre az új streamek, itt tárolódik minden streamhez tartozó információ (pl. stream neve, leírása, kategóriája, kulcsa, elérési útja, nézettségi adatok). A StreamDbHandler biztosítja, hogy minden stream egyedi azonosítóval és a hozzá tartozó felhasználóval legyen összekapcsolva.

4. **StreamService (Stream logika, kulcsgenerálás, indítás/leállítás)** Ez a modul végzi a streameléshez kapcsolódó üzleti logikát. Itt történik például a stream kulcsok generálása, a stream indítása és leállítása, valamint a felhasználóhoz tartozó ajánlott streamek előállítása. A StreamService kommunikál a StreamDbHandler-rel, hogy frissítse a streamek állapotát, illetve a UserDbHandler-rel, hogy a felhasználók élő státuszát kezelje.

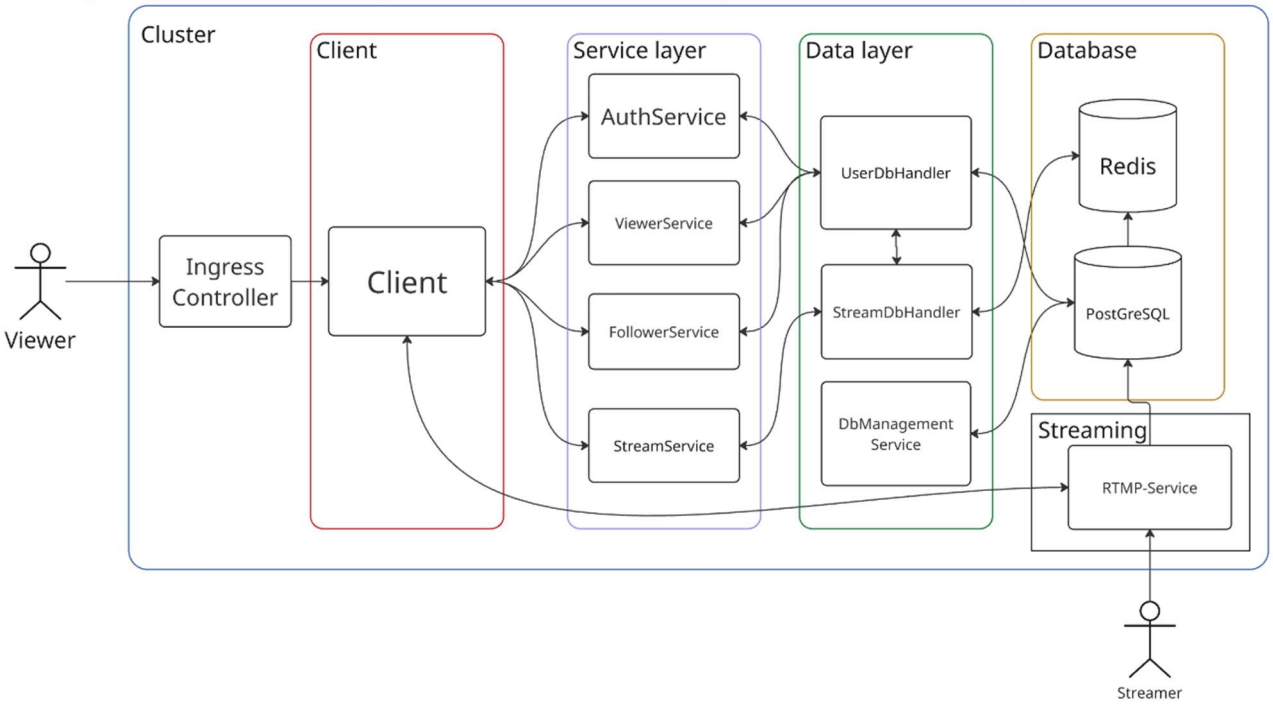
5. **ViewerService (Nézők kezelése, nézőszámlálás)** Ez a szolgáltatás tartja nyilván, hogy egy adott streamet éppen hányan néznek. Redis alapú cache-t használ a nézők gyors számolásához, és minden néző csatlakozásakor vagy távozásakor frissíti a nézettségi adatokat a StreamDbHandler-ben is. Így valós idejű nézőszámot tudunk mutatni minden streamhez.

6. **FollowerService (Követések kezelése)** Ez a modul kezeli a felhasználók közötti követési kapcsolatokat. Lehetővé teszi, hogy egy felhasználó másokat kövessen, és ezek alapján például személyre szabott ajánlásokat vagy értesítéseket lehessen küldeni.

7. **Nginx-RTMP (Streaming szerver)** Ez a komponens maga a streaming szerver, amely az RTMP protokollon keresztül fogadja a videófolyamokat (pl. OBS-ből vagy egy scriptből), majd HLS formátumban továbbítja azokat a nézők felé. Az Nginx-RTMP szerver a rendszer központi eleme, amely a tényleges videóadatokat kezeli.

8. **Database (Adatbázis) és Redis (Gyorsítótár)** A PostgreSQL adatbázis tárolja a felhasználók, streamek és követések tartós adatait. A Redis cache gyors elérésű, ideiglenes adatokat (pl. nézők listája, aktív streamek) tárol, hogy a rendszer gyorsan tudjon reagálni a nézőszám változásaira.

9. **DbManagementService (Adatbáziskezelő):** Ez a komponens az adatbázis sémájának karbantartását és a migrációk futtatását végzi. Segítségével biztosítható, hogy az adatbázis szerkezete mindig naprakész legyen a fejlesztések során.
10. **Ingress:** Az Ingress komponens a Kubernetes klaszterben az Nginx Ingress Controller segítségével valósul meg. Az Ingress felelős a bejövő HTTP/HTTPS kérések irányításáért a megfelelő szolgáltatások (például frontend, streaming backend, AuthService) felé.

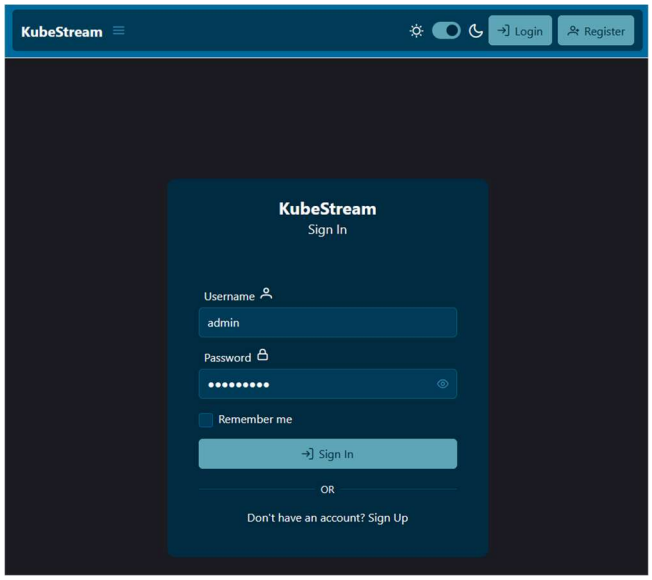


2. ábra
A kubernetes cluster logikai felépítése

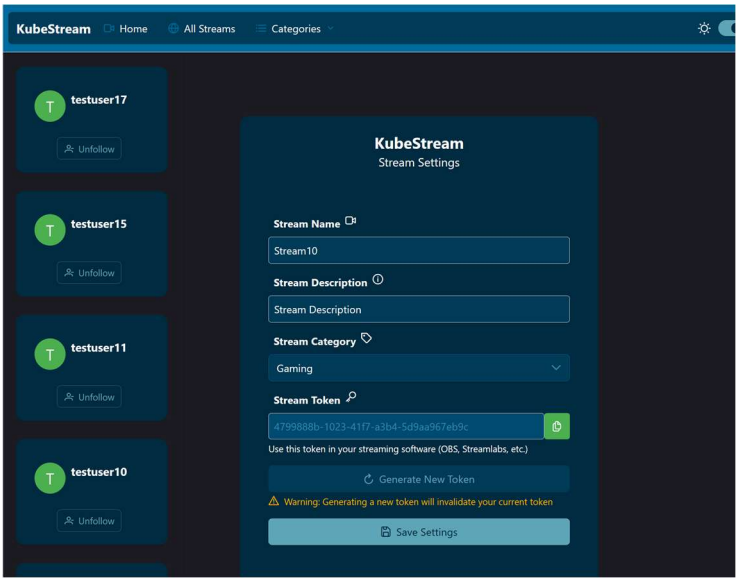
Frontend

A kliens egy Angular alapú, modern, reszponzív webalkalmazás, amely lehetővé teszi élő streamek böngészését, megtekintését, saját stream indítását, felhasználói profilok kezelését, valamint a felhasználók közötti interakciókat (pl. követés). A frontend valós idejű nézőszámlálást, stream ajánlásokat, regisztrációt, bejelentkezést és stream beállításokat is biztosít. A lejátszás HLS-en keresztül, Shaka Playerrel történik. Főbb komponensei:

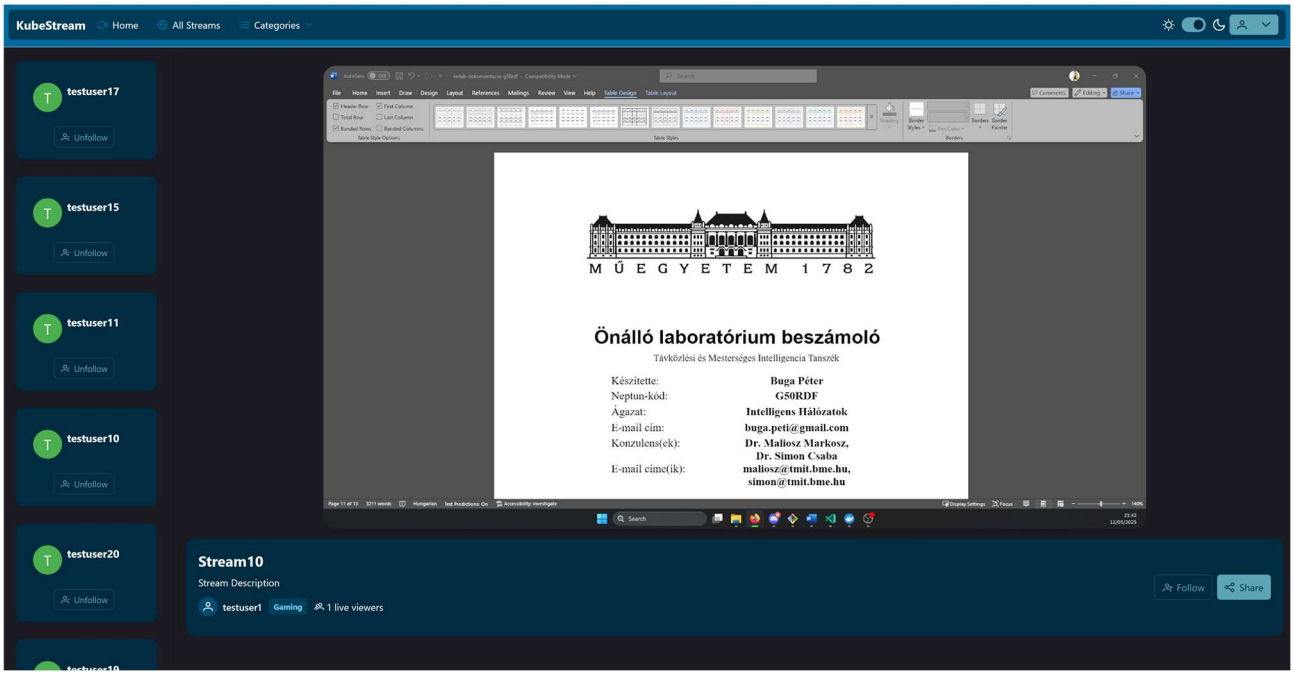
- **Stream:** Egy adott élő stream megtekintését biztosító oldal (5.ábra). Betölti a stream adatait, elindítja a videólejátszót (Shaka Player), kezeli a nézőszámlálást, lehetőséget ad a streamer követésére/leválasztására, valamint a stream linkjének megosztására.
- **Ajánlások:** Stream ajánló oldal, amely a felhasználó számára releváns vagy népszerű élő streameket jelenít meg. Kategória szerint is lehet szűrni, és minden streamhez valós idejű nézőszámot is mutat.
- **Beállítások:** A felhasználó streamjének beállításait kezelő felület (4.ábra). Itt lehet módosítani a stream nevét, leírását, kategóriáját, valamint új stream kulcsot is lehet létrehozni, ha még nincs. A beállítások mentése után a változások azonnal érvénybe lépnek.
- **Profil:** Felhasználói profil oldal, ahol a felhasználó megtekintheti és szerkesztheti saját adatait (pl. név, email, jelszó). Az adatok módosítása után visszajelzést kap a sikerességről vagy hibákról.
- **Regisztráció/Belépés:** Űrlap a megfelelő attribútumokkal (3. ábra), ellenőrzi az adatok hitelességét, hiba esetén visszajelzést ad.
- **Felső menü:** Tartalmazza az alkalmazás nevét és a fő navigációs linkeket (pl. főoldal, felfedezés, profil), valamint a felhasználói fiókhoz kapcsolódó műveleteket (pl. bejelentkezés, kijelentkezés).
- **Oldalsó menü:** Egy listát tartalmaz a követett streamer-ekről (4.ábra), látszik, hogy éppen élő adásban van-e az adott streamer és a jelenlegi nézőszám is meg van jelenítve.



3. ábra
Bejelentkező oldal



4.ábra
Beállítások oldal



5. ábra
Stream néző felület

2.6 Összefoglalás

A laboratóriumi munka során egy élő videó streaming rendszert fejlesztettem ki, amely a modern mikroszolgáltatás-alapú és konténerizált architektúra elveit követi. A projekt fő célja az volt, hogy egy olyan platformot hozzak létre, amelyben a különböző funkciók – mint a felhasználókezelés, stream kezelés, nézőszámlálás és jogosultságkezelés – egymástól függetlenül, mégis szorosan együttműködve működnek. A rendszer minden fő komponense külön szolgáltatásként, saját konténerben fut, így könnyen bővíthető és skálázható. A fejlesztés során kiemelt figyelmet fordítottam arra, hogy a komponensek közötti kommunikáció megbízható és biztonságos legyen, valamint, hogy a rendszer képes legyen valós időben kezelni a nézőszámokat és a felhasználói interakciókat.

A munka során számos kihívással találkoztam, a fejlesztés különböző területein. Például előfordult, hogy a ViewerService-ben frissült a nézőszám, de a StreamDbHandler-ben nem jelent meg azonnal a változás, ezért külön szinkronizációs logikát kellett írni. A streaming szerver (Nginx-RTMP) és a backend szolgáltatások közötti kommunikációt is külön kellett kezelni, ezt webhookokkal és API hívásokkal oldottam meg. A különböző mikroszolgáltatások (például a StreamService és a StreamDbHandler) közötti adatátadásnál gondot okozott, ha egy szolgáltatás vagy az adat, nem volt elérhető ezért hibakezelést kellett beépíteni. A felhasználók bejelentkezése után a JWT tokenek érvényességét minden szolgáltatásban ellenőrizni kellett, amihez közös token handler-t kellett fejleszteni.

Ezek megoldása során mélyebb ismereteket szereztem a felhőalapú rendszerek, a Kubernetes és a konténerizáció működéséről. A projekt eredményeként egy stabil, jól működő streaming platform jött létre, amely a jövőben további funkciókkal – például chat, értesítések vagy újabb streaming protokollok – is bővíthető

3. Irodalom, és csatlakozó dokumentumok jegyzéke

A tanulmányozott irodalom jegyzéke:

[1] Veriskope, „Real-Time Messaging Protocol (RTMP) Specification,” 2019. [Online]. Available: https://rtmp.veriskope.com/docs/spec/. [Hozzáférés dátuma: 05 10 2025].

[2] „HTTP Live Streaming,” [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8216. [Hozzáférés dátuma: 10 05 2025].

[3] Amazon, „What’s the Difference Between Monolithic and Microservices Architecture?,” [Online]. Available: https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/. [Hozzáférés dátuma: 10 05 2025].

[4] Kubernetes, „Learn the basics,” [Online]. Available: https://kubernetes.io/docs/tutorials/kubernetes-basics/. [Hozzáférés dátuma: 10 05 2025].

[5] ITConvergence, „Navigating the peaks and valleys advantages of auto scaling in cloud computing for agile demand management,” 14 January 2024. [Online]. Available: https://www.itconvergence.com/blog/navigating-the-peaks-and-valleys-advantages-of-auto-scaling-in-cloud-computing-for-agile-demand-management/. [Hozzáférés dátuma: 10 05 2025].

[6] JWT, „Introduction,” [Online]. Available: https://jwt.io/introduction. [Hozzáférés dátuma: 10 05 2025].

[7] D. Hart, „RFC6749,” 2012, October. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749. [Hozzáférés dátuma: 10 05 2025].

[8] „.NET,” [Online]. Available: https://dotnet.microsoft.com/en-us/. [Hozzáférés dátuma: 10 05 2025].

[9] „Angular,” [Online]. Available: https://angular.dev/. [Hozzáférés dátuma: 10 05 2025].

[10] „PostgreSQL,” [Online]. Available: https://www.postgresql.org/. [Hozzáférés dátuma: 10 05 2025].

[11] „Redis,” [Online]. Available: https://redis.io/. [Hozzáférés dátuma: 10 05 2025].

[12] Arut, „NGINX-RTMP modul,” [Online]. Available: https://github.com/arut/nginx-rtmp-module. [Hozzáférés dátuma: 10 02 2025].

[13] tiangolo, „Nginx-RTMP Docker Image,” [Online]. Available: https://hub.docker.com/r/tiangolo/nginx-rtmp/. [Hozzáférés dátuma: 10 05 2025].

[14] „Shaka Player,” [Online]. Available: https://github.com/shaka-project/shaka-player. [Hozzáférés dátuma: 10 05 2025].

[15] „Docker,” [Online]. Available: https://www.docker.com/. [Hozzáférés dátuma: 10 05 2025].

Csatlakozó egyéb elkészült dokumentációk / fájlok / stb. jegyzéke:

GithHub Repository: https://github.com/GriffManoue/live-streaming-kubernetes