ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

DIPLOMAMUNKA

OE-NIK	Hallgató neve:	**Bedő Sebestyén Péter**
2025	Hallgató törzskönyvi száma:	**T009082/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Bedő Sebestyén Péter**
Törzskönyvi száma: T009082/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Felhő alapú influenszer követő-like korreláció elemzés
Cloud-based influencer follow-like correlation analysis

Intézményi konzulens: Sipos Miklós László
Külső konzulens:

Beadási határidő: 2024. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Felhő szolgáltatási technológiák és IT biztonság specializáció

A feladat

A feladat egy olyan felhő alapú webalkalmazás készítése, amivel a felhasználó képes influenszerek like és követő számát lekérni, majd az ezekből készített kimutatásokat ltérehozni. Vizsgálja meg, hogy mely technológiák szükségesek a webes rendszer, valamint a mögötte álló felhő infrastruktúra elkészítéséhez – választását indokolja. A webes felület biztosítson keresési lehetőséget, ahol a felhasználó meg tudja adni azoknak influenszerek számát és azonosítóját, akikről be kíván gyűjteni adatokat. Továbbá, itt tekinthesse meg a felhasználó a begyűjtött adatokból készített kimutatásokat, statisztikákat. Készítsen adatbázist valamilyen felhő technológia segítségével, ahol tárolja el a begyűjtött adatokat. Biztosítsa továbbá valamilyen felhő megoldással, hogy az alkalmazás egyfolytában elérhető legyen a felhasználó számára. A lekérdezések fussanak le előre megadott időközönként automatikusan. Végezzen továbbá elemzést a tesztelési lehetőségekről és készítse is el az egyes teszt típusokat és teszteseteket.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- magas szintű szerkezeti mintát az alkalmazásról,
- az egyes funkciók megvalósításához felhasználható lehetséges technológiákat,
- az elkészített rendszer tervét, blokkdiagramját,
- a megvalósítás menetét, felmerülő problémákat és az azokra adott megoldást,
- a tesztelési tervet, a tesztelés módszertanát és a tesztelés eredményeit,
- az elkészült szoftver felhasználási lehetőségeit,
- az eredmények bemutatását és értékelését a teljes munkára vonatkozóan.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

Ph.

.....
Dr. habil. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2026. december 15.**
(ÓE HKR 54.§ (10) bekezdés szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2024. 12. 06.

.....
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: BEDŐ SEBESTYÉN PÉTER

Neptun Kód: [redacted]

Tagozat: NAPPALI TAGOZAT

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

FELHŐ ALAPÚ INFLUENSZER KÖVETŐ-LIKE KORRELÁCIÓ ELEMZÉS

Szakdolgozat címe angolul:

CLOUD-BASED INFLUENCER FOLLOW-LIKE CORRELATION ANALYSIS

Intézményi konzulens: SIPOS MIKLÓS LÁSZLÓ

Külső konzulens: -

Alk.	Dátum	Tartalom	Aláírás
1.	2024.02.19.	Félév teendőinek, menetrendjének és mérföldköveinek ismertetése.	
2.	2024.03.11.	Hasonló rendszerek elemzése.	
3.	2024.04.15.	Irodalomkutatás, annak összegzése valamint konzekvenciák meghatározása.	
4.	2024.05.06.	Tervezés, követelmények specifikációja. Véglegesítés. Leadással kapcsolatos teendők egyeztetése.	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. tantárgy követelményét teljesítette, beszámolóra bocsátható.

A konzulens által javasolt érdemjegy: 4

.....

Intézményi konzulens

Budapest, 2024.05.17.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: BEDŐ SEBESTYÉN PÉTER [redacted]

Neptun Kód: [redacted]

Tagozat: NAPPALI

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

FELHŐ ALAPÚ INFLUENSZER KÖVETŐ-LIKE KORRELÁCIÓ ELEMZÉS

Szakdolgozat címe angolul:

CLOUD-BASED INFLUENCER FOLLOW-LIKE CORRELATION ANALYSIS

Intézményi konzulens: SIPOS MIKLÓS LÁSZLÓ

Külső konzulens: -

Alk.	Dátum	Tartalom	Aláírás
1.	2024.09.29.	Fejlesztés tervezett menetének revíziója, szakmai egyeztetés, fejlesztési teendők megkezdése.	
2.	2024.10.20.	Fejlesztés állapotának és haladásának áttekintése. Technikai problémák és módosítások egyeztetése.	
3.	2024.11.17.	Felmerülő problémák egyeztetése és megoldása. Tesztelési fázis megindítása.	
4.	2024.12.01.	Dokumentáció véglegesítése. Leadással kapcsolatos teendők egyeztetése.	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat II. tantárgy követelményét teljesítette, védésre bocsájtható.

A konzulens által javasolt érdemjegy: 5

.....

Intézményi konzulens

Budapest, 2024. 12. 06.

Tartalomjegyzék

ABSZTRAKT

Absztrakt

A célom ezzel a dolgozattal, hogy egy olyan felhőalapú webalkalmazást hozzak létre, amely képes influenszerek követő- és like-metrikáit kigyűjteni, és ezeket statisztikák és grafikonok formájában a felhasználó elé tárni. Emellett cél, hogy az alkalmazás folyamatosan rendelkezésre álljon, és időzített adatlekérés funkciót is biztosítson. A dolgozat bemutatja, hogy milyen lépéseket tettem ahhoz, hogy eljussak a végeredményhez, valamint hogy a kutatómunka, irodalomkutatás és olyan hasonló rendszerek vizsgálata, mint az InsTrack, hogyan befolyásolta az általam használt eszközöket és a végtermék felé támasztott elvárásokat. Kutatómunkám során hangsúlyt fektettem az AWS és Azure különböző felhőszolgáltatásainak összehasonlítására, és végül az Azure megoldásai mellett döntöttem, mert ezek határozottan jól működnek a C# programozási nyelvvel és a Visual Studio 2022 fejlesztői környezettel. A végtermék lehetővé teszi, hogy a felhasználó Google-autentikációval azonosítsa magát, ezután YouTube-csatornák metrikáit lekérje, ezekről kimutatásokat tekintsen meg, csatornákat kövessen, és ezekre időzített lekérést állítson be. Az adat lekérdezések és eltárolásuk HTTP Trigger Azure Functions-ön keresztül történik, az időzített lekérdezés kezeléséről Azure Durable Function gondoskodik.

Abstract

My goal with this thesis is to create a cloud-based web application capable of collecting follower and like metrics from influencers and present them to users in the form of statistics and graphs. In addition, the application must remain available at all times and provide a scheduled data retrieval function. The thesis describes the steps I took to achieve the final result, as well as the way in which research, literature review and analysis were carried out. Investigating similar systems like InsTrack influenced the tools I used and what I expected from the final product, in terms of performance, and UI. During my research, I focused on comparing different cloud services from AWS and Azure and ultimately decided on Azure services due to its compatibility with the Csharp programming language and the Visual Studio 2022 environment. The final product allows users to authenticate through Google, get metrics for YouTube channels, view reports on those metrics, Follow channels and set up scheduled queries for them. Data queries and storage are handled via HTTP Trigger Azure Functions, scheduled querie is handled by Azure Durable Funtion.

1. BEVEZETÉS

Az elmúlt években a különböző közösségi média platformok életünk meghatározó részévé váltak. Sokan az itt megosztott hírek alapján tájékozódnak, itt tartjuk a kapcsolatot ismerőseinkkel, itt követjük az általunk érdekesnek tartott személyeket, valamint sokan itt osztják meg gondolataikat és életük fontosabb történéseit. Így tehát kijelenthető, hogy képernyőidőnk jelentős részét közösségi média platformokon töltjük. Ugyan a YouTube nem egy közösségi média platform, mégis úgy érzem, hogy ideköthető, tekintve, hogy az ott közzétett tartalom sok esetben befolyásolja az emberek gondolatmenetét, és akarva-akaratlanul hatással van a döntéseikre. Persze nem kellett sok idő, hogy a különböző cégek rájöjjenek, mekkora szabad reklámfelület áll rendelkezésükre a különböző platformokra tartalmat gyártó felhasználókon keresztül.

Egyrészt elhelyezhetnek reklámokat posztok formájában, melyeket a platform majd pénzért cserébe megjelenít a felhasználók számára. Másrészt a cégek hirdethetik termékeiket olyan embereken keresztül, akik nagy követőtáborral rendelkeznek. Ezeket a követőket befolyásolhatják közvetlen vagy közvetett módon is, hogy egy adott terméket részesítsenek előnyben a versenytársak termékeivel szemben. Továbbá nem mehetek el szó nélkül a különböző YouTube-videósok tartalmai mellett, melyek esetenként fizetett hirdetéseket tartalmaznak, és nagyban befolyásolhatják a nézőik döntéseit egy-egy termék, szoftver vagy szolgáltatás kiválasztásakor. Marketing szempontból fontos, hogy a terméket a megfelelő helyen, a megfelelő demográfiának hirdessék. Itt jönnek képbe az olyan alkalmazások, melyek segítségével követhetik és rangsorolhatják a vállalatok, hogy az egyes influencerek, és tartalomgyártók milyen eléréssel rendelkeznek. Az egyik első dolog, amire kíváncsiak, hogy hogyan viszonyul a követők száma a posztjaira érkező like-ok számához. Ebből ugyanis könnyedén kideríthető, hogy a követőkből ténylegesen hány felhasználót érdekelnek az influencer posztjai, vagy hogy a követőiken túl nagyjából hány embert érnek el a posztjaik, videóik.

Dolgozatom célja, hogy egy olyan alkalmazást építsek, amely képes egy előre megadott publikus YouTube-felhasználó profiljáról lekérni az olyan adatokat, mint a feliratkozók száma, valamint a videókra kapott like-ok és kommentek száma, majd ezen adatokból különböző kimutatásokat és statisztikákat készíteni. Ugyan ezt megtehetném Youtube csatornák helyett, Indtagram influencerekkel is hiszen ez jobban illene a szakdolgozat címéhez, és az eredeti tervem is ez volt, azonban kiderült, hogy a hivatalos Instagram Graph API-n keresztül a felhasználó bejelentkezés után csak a saját profil adatait kérdezheti le ilyen módon.

Célom továbbá az is, hogy az alkalmazás a felhasználók számára a nap minden percében elérhető legyen, és az egyes felhasználók képesek legyenek egyszerre akár 5 Youtube

csatornát követni. Mindezt úgy, hogy az egyes csatornák már említett mutatóit, ne csak manuálisan klóérdezhesse le, hanem periodikusan, a háttérben is képes legyen rá. Ezzel a funkcióval is gördülékenyebbé téve az adatkinyerés folyamatát. Ilyen alkalmazások esetében az adatok tárolása is kiemelt fontosságot élvez, hiszen az ezekből készülnek a későbbiekben a kimutatások. Emiatt rendkívül fontos, hogy az eltárolt adatokhoz gyorsan hozzáférhessünk, és megőrizzék integritásukat. Ahhoz, hogy az alkalmazás ezeket a funkciókat biztosíthassa, valamiféle infrastruktúrára van szüksége. Erre megoldást jelenthet az is, ha egy fizikai szervert használok és üzemeltetek, de én felhőszolgáltatások mellett döntöttem, mivel ezek jobban skálázhatóak, kisebb kezdőtőkét igényelnek, és sok esetben jobb rendelkezésre állást biztosítanak a felhasználónak. A későbbiekben össze fogom hasonlítani az AWS, valamint az Azure felhőszolgáltatók által kínált szolgáltatásokat, és a kutatás alapján kiválasztom az esetemben optimális szolgáltatót.

2. IRODALOMKUTATÁS

2.1. Hasonló rendszerek

2.1.1. Instrack bemutatása

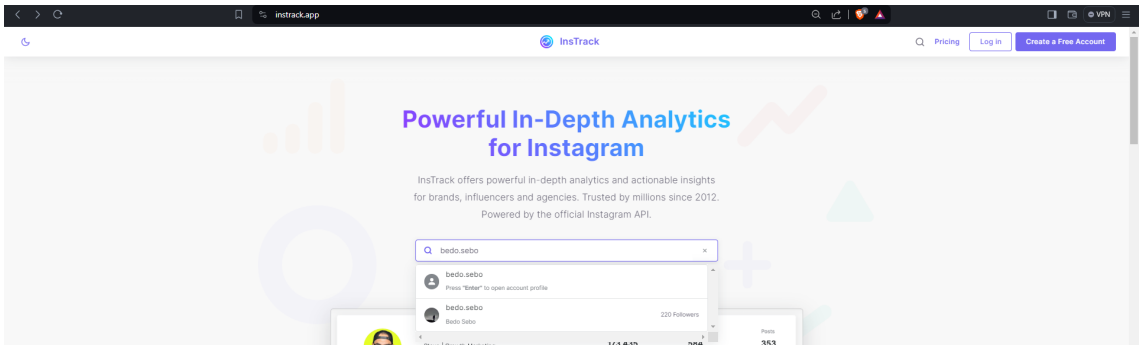
A szakdolgozatomban megvalósítandó webes alkalmazás elkészítéséhez több, hasonló funkciójú, már létező szoftvert vizsgáltam meg, Ugyan ezek főként Instagramra fókuszálnak, vagy több platformot kötnek össze egy felületre, dde csak YouTube-ra koncentráló alkalmazást nem nagyon találtam, vagy azok nem feleltek meg az igényeimnek. Szerencsére funkciójában ugyanazt tudják ezek, mint amit én is szeretnék megvalósítani funkciók, és külcsíny terén.

Az első ilyen az InsTrack [1] nevű webes applikáció. Itt szabadon lehet keresni Business és Creator Instagram profilok között, mindössze a profiljuk nevének beírásával. (2.1. ábra) Ehhez egy roppant egyszerű kezelőfelület áll a felhasználó rendelkezésére. Miután megtaláltuk a keresett profilt, nyomon lehet követni, hogy hány követővel rendelkezik, hányan lájkolják átlagosan a posztjait, valamint hogy a követőinek hány százaléka lájkolja a posztjait. (2.2. ábra) Ezentúl nyomon követhetjük azt is, hogy az elmúlt 90 napban hány százalékkal növekedett a profil követő száma, valamint azt is hogy az elmúlt héten hány új felhasználó csatlakozott a követők táborához. Annak érdekében, hogy a felhasználó ne csak nyers számokat kapjon vissza, különböző grafikonok is rendelkezésére állnak. (2.3. ábra) Ezen grafikonokon megtekinthető, hogy az elmúlt hónapban, hogyan változott a követők, és a posztokra érkező like-ok száma, valamint az, hogy az összes követő hány százalékát éri el 1-1 poszt.

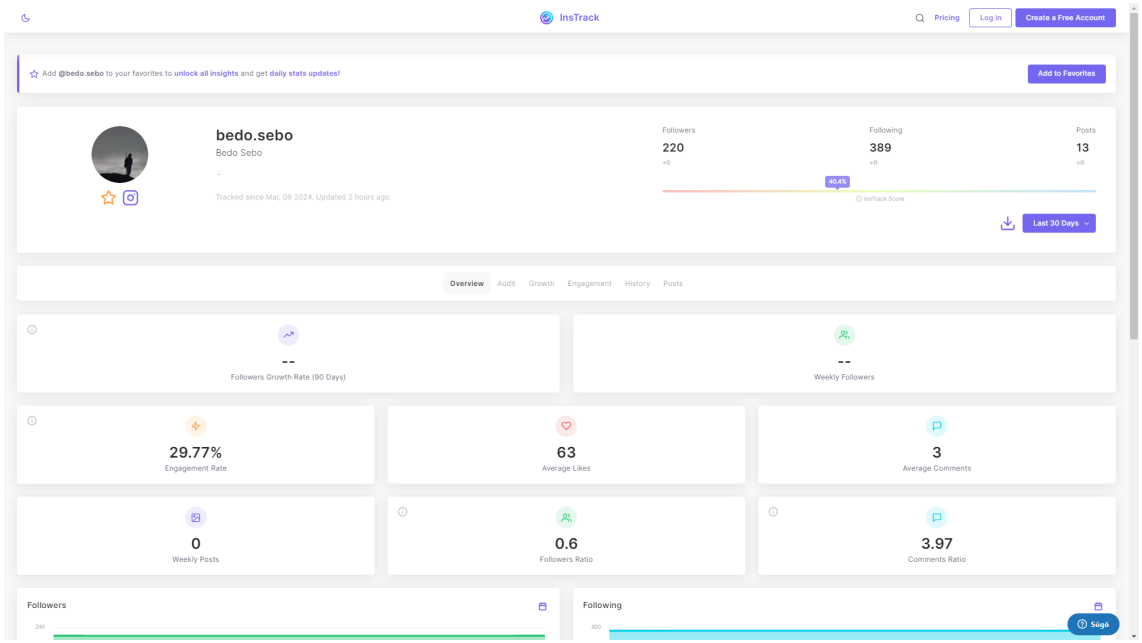
A fentebb felsorolt funkciók mind elérhetők regisztráció nélkül, azonban ha létrehozunk egy profilt, újabb szolgáltatások kipróbálására kapunk lehetőséget. Ezek közé tartozik például az egyes posztok elemzése, vagy az időzített jelentés, amely lehetővé teszi hogy a vizsgált profilokról előre meghatározott időközönként készüljön jelentés. A felhasználónak lehetősége van többféle előfizetési szint között választani, így mindenki megtalálhatja a neki megfelelő opciót, de az alkalmazást teljesen bérmentve, regisztráció nélkül is ki lehet próbálni. A drágább előfizetések esetén nem csak a követhető fiókok száma nő, de a kinyert információ is részletesebb, és segít megtalálni a legnagyobb elérést generáló hashtag-eket és kulcsszavakat. Amennyiben több felhasználót is nyomonkövetünk, egy lista fog rendelkezésünkre állni, ahol sorba rendezve tekinthetjük meg a profilokat, és pár fontosabb értéket.(2.4. ábra) A listában szereplő fiókokat, akár össze is hasonlíthatjuk egymással.

Úgy gondolom, hogy ez egy kiváló szolgáltatás olyan tartalomgyártók, és cégek szá-

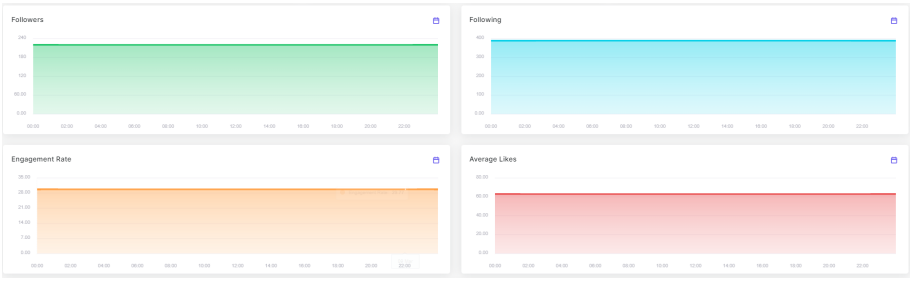
mára, akik nyomon akarják követni a saját, vagy versenytársaik profiljának a növekedését, és az elérés mértékét. A kezelőfelület egyszerű, letisztult, és intuitív. Az adatokból generált statisztikák és grafikonok ugyancsak könnyen értelmezhetőek, és fontos információkat tárnak a felhasználó elé. Ezzel is segítve a kinyert adatokból készült elemzés közérthetőségét.



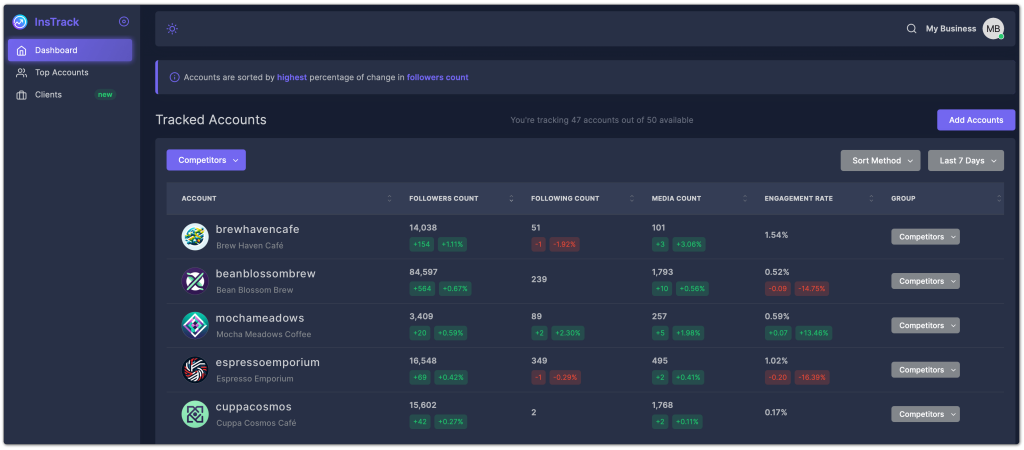
2.1. ábra: Profil keresése InsTrack-en.



2.2. ábra: InsTrack által a profilról kinyert adatok.



2.3. ábra: Adatok értelmezését segít grafikonok InsTrack-en.



2.4. ábra: Vizsgált profilokat összegző lista InsTrack-en.

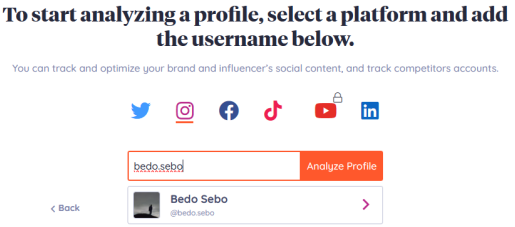
2.1.2. Keyhole bemutatása

A második applikáció amit megvizsgáltam a Keyhole [2]. Ez a webalkalmazás már nem csak Instagram fiókok követésére képes, hanem más közösségi média platformok profiljairól is képes elemzést készíteni. (2.5. ábra) Profilok követése mellett, itt is lehetősége van a felhasználónak hashtag-ek és kulcsszavak nyomonkövetésére is, ezáltal pontosabban lekövethető egy termék, vagy esemény népszerűsége. A követett fiókokat egy listában tudjuk megtekinteni. (2.6. ábra) A listában lévő profilok összehasonlíthatók egymással, így könnyedén össze lehet vetni a fiókok közötti növekedésbeli különbségeket, ezzel is egyszerűbbé téve a különböző profilok értékeinek összehasonlítását. (2.9. ábra) összehasonlíthatunk akár különböző közösségi média platformokról származó profilokat is. Ez roppant hasznos lehet, amennyiben ki akarjuk deríteni, hogy melyik platformon érjük el a legtöbb felhasználót. Ilyen információ alapján könnyebben kidolgozhatunk egy stratégiát a közösségi média jelenlétünk optimalizására. A profil analizálása után a felhasználó láthatja a követők számát, az adott időszakban kirakott posztok átlagos like, és elérés számát, és több grafikonon is láthat olyan információkat mint: a követő szám változása, az átlag like szám változása, valamint a legtöbb elérést generáló poszt. (2.7. és 2.8. ábrák)

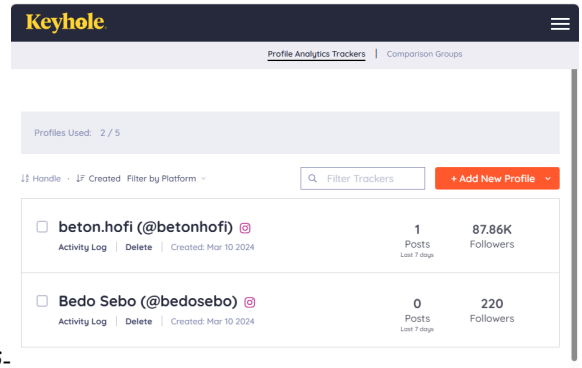
Sajnos a próbaverzióban csak egy szűk idő intervallumban bekövetkezett változásokat lehet megtekinteni.

Véleményem szerint ez az alkalmazás leginkább olyan cégek, és influencerek számára hasznos, akik több közösségi média platformon is rendelkeznek profilokkal, és az ezekről készült kimutatásokat szeretnék egy helyre összegyűjtve látni, és értékelni. Ugyanakkor maradéktalanul nyújtja ugyanazon szolgáltatásokat, mint a már fentebb említett InsTrack [1]. Előnyére válik az is, hogy képes piac kutatásra, és közösségi média trendek analizálására. Úgy gondolom, hogy ezek a szolgáltatások rendkívül hasznosak egy nagyobb cég számára, ahol kiemelten fontos az adott üzletágban felbukkanó trendek nyomonkövetése, vagy akár azoknak megelőzése.

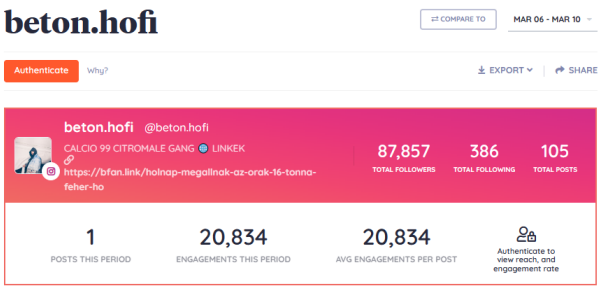
A próba verzió elégséges betekintést nyújt az applikáció működésébe ahhoz, hogy a felhasználó eldönthesse, hogy igényeinek megfelelő e a szolgáltatás. A kezelőfelület ezen applikáció esetében is letisztult, de kicsit kevésbé átlátható. Emiatt a felhasználók nehézségekbe ütközhetnek használata során. Továbbá a kinyert statisztikák, és grafikonok mennyiségével és részletességével sem voltam maradéktalanul megelégedve. Fontos azonban kiemelnem, hogy csak a próba verziót próbáltam ki, és ez csak limitált hozzáféréssel rendelkezik. Itt is több előfizetési szint közül lehet választani, és vannak kifejezetten nagyobb cégeknek ajánlott szint is, ezek sokkal jobban személyre szabhatóak.



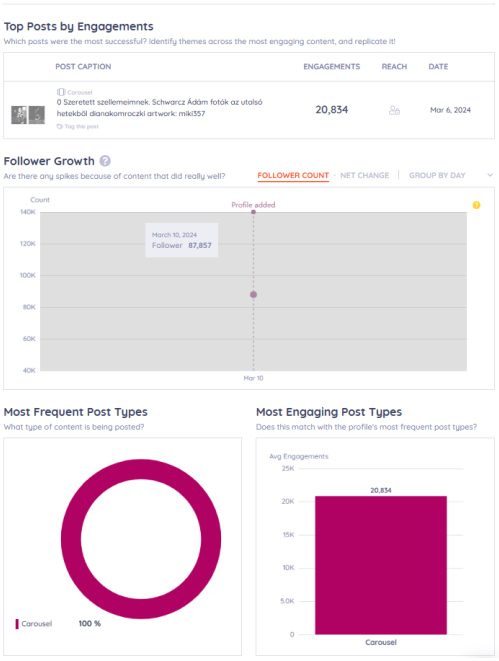
2.5. ábra: Profil keresése Keyhole-on. Több közösségi média platform közül választhatunk.



2.6. ábra: A Keyhole által elemzett profilok listája.



2.7. ábra: Keyhole által kinyert adatok ábrázolása.



2.8. ábra: Adatok emészthetőségét javító grafikonok.

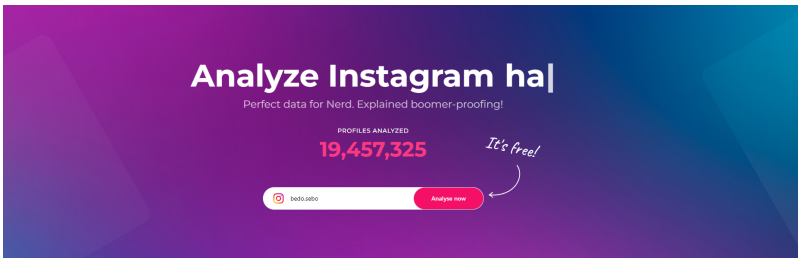
2.9. ábra: Azonos közösségi média platformról származó profilok összehasonlítása.

2.1.3. NotJustAnalytics bemutatása

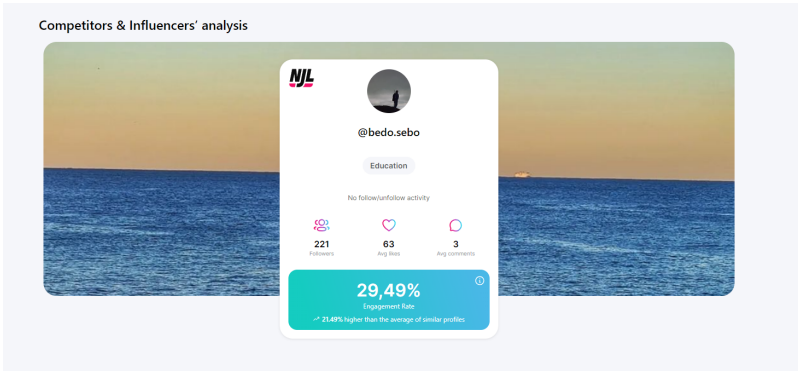
A harmadik vizsgált rendszer a NotJustAnalytics [3]. Ez egy webalkalmazás, amely az Instrack-hez [1] hasonlóan Instagram profilokról készít elemzéseket. Ahhoz hogy jelentést kapjunk egy profilról egyszerűen csak rá kell keresünk a felhasználó Instagrammon használt nevére. (2.10. ábra) Ezután 1-2 másodperc feldolgozási idő után a felhasználó szeme elé tárul egy letisztult csempés felosztású felület, ahol megtekinthetjük az elkészített kimutatásokat. (2.11. ábra) Megtekinthetjük a vizsgált profil követő számát a posztjaira kapott átlag like számot, valamint a posztjaira átlagosan érkező kommentek számát. A követő szám és az átlag like szám alapján kiszámol egy százalékos értéket, ami azt méri, hogy a követők hány százaléka lájkol egy-egy posztot. Természetesen az Instrack-hez [1], és a Keyhole-hoz [2] hasonlóan itt is grafikonok segítik a kinyert adatok értelmezését. (2.12. ábra) Az előző vizsgált rendszerekhe hasonlóan itt is láthatunk grafikonokat, amik megmutatják egy bizonyos idő intervallumban bekövetkezett növekedést vagy csökkenést. Az előzőekkel ellentétben itt a profil minden egyes korábbi posztjáról megtekinthetünk egy kimutatást, ahol láthatjuk a posztra kapott like-ok és kommentek számát, valamint a hashtag-eket, amelyeket a poszthoz fűzött a felhasználó. (2.13. ábra) Ezen információk rendelkezésünkre állnak egy szemléletes grafikon formájában is, amely megkönnyíti az adatok emészthetőségét. Rendelkezésünkre áll még egy elemzés arról is, hogy melyik hashtag generálta a legnagyobb elérést.

Ez az alkalmazás is kipróbálható ingyenesen regisztráció nélkül, de ilyenkor csak a vizsgált profilról elkészült elemzés tekinthetjük meg. A magasabb szintű szolgáltatások használatához előfizetésre van szükség. Ilyen szolgáltatás például az is, hogy a posztokról készített elemzés alapján tanácsokat kap a felhasználó. Ezekben a tanácsokban többet megtudhat a felhasználó, arról hogy milyen időben, mit és milyen hosszúságú leírással rakjon ki, ahhoz hogy optimalizálja posztjai elérését. De nem feledkezhetünk meg arról sem, hogy amennyiben előfizetünk lehetőségünk nyílik egyszerre több fiókot is nyomonkövetni, és összehasonlítani. Ezentúl a felhasználónak lehetősége van fix időközönként, automatikusan PDF kimutatásokat küldeni. Úgy gondolom ez nagyon megkönnyítheti egy olyan szakember feladatát, aki Instagram fiókok menedzselésével, vagy monitorozásával foglalkozik.

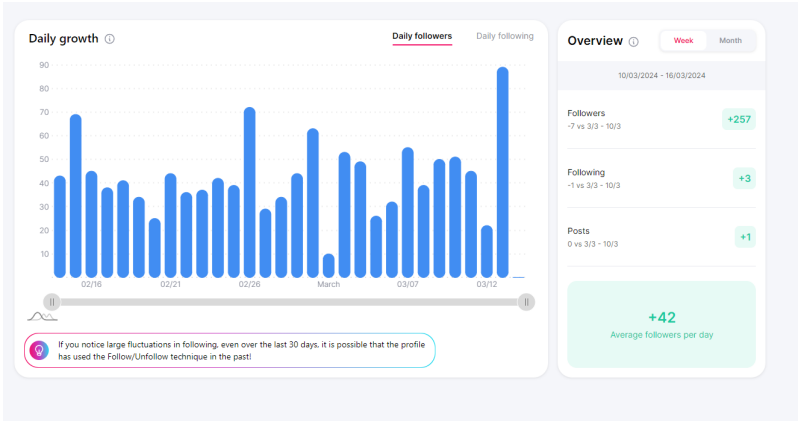
Összességében elmondható, hogy egy könnyen átlátható és használható kezelőfelület áll a felhasználó rendelkezésére. A kimutatások jól átláthatóak, tartalmasak, és nem tartalmaznak felesleges adatokat. A különböző előfizetési szintek sok plusz funkciót kínálnak, így minden felhasználó megtalálhatja a neki szükséges csomagot. Talán hasznos lenne, ha biztosítanának lehetőséget egyéni csomagok összeállítására.



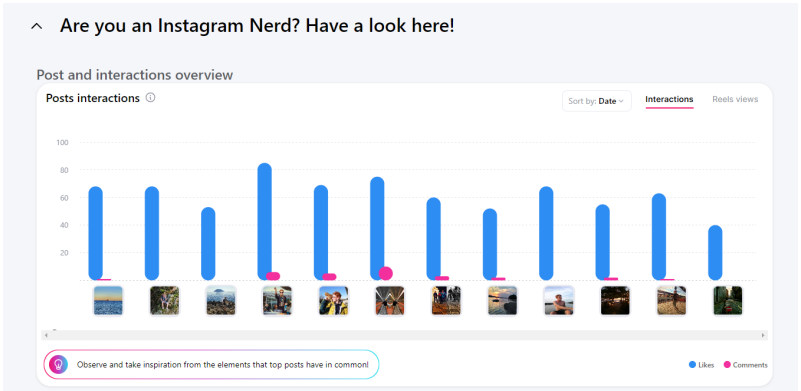
2.10. ábra: Profil keresése



2.11. ábra: Profil adatok



2.12. ábra: Grafikon a követőkről



2.13. ábra: Poszt elemzés

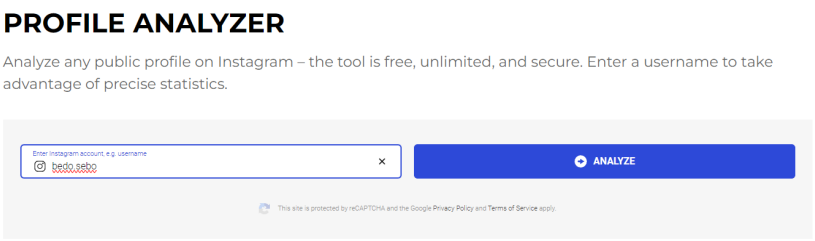
2.2. Inflact bemutatása

A negyedik rendszer, amit megvizsgáltam az Inflact [4]. Ennek a webes alkalmazásnak az a célja, hogy segítse egy közösségi média profil növekedését, és könnyítse annak kezelését. Az előzőleg vizsgált rendszerekhez képest, ez az alkalmazás több funkcióval is rendelkezik, amelyek egyszerűbbé teszik egy Instagram profil menedzselését. Ilyen például a tervezett posztolás, mások posztjainak automatikus like-olása, vagy az üzenetekre való automatikus válaszolás. Ezentúl az Inflact [4] segítséget nyújt a felhasználónak, abban is, hogy milyen tartalmat mikor érdemes megosztani, és milyen hashtag-et érdemes ráaggatni az elérés optimalizálása érdekében.

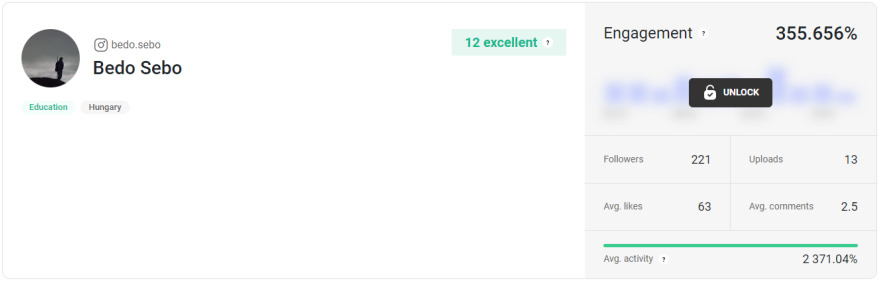
A többi rendszerhez hasonlóan ennek a rendszernek is van Instagram profil elemző szolgáltatása. Miután rákerestünk egy Instagram felhasználó nevére, nagyjából 30 másodpercet kell várnunk mielőtt megláthatjuk az eredményt. (Keresés: 2.14. ábra, Eredmény: 2.15. ábra) A fiók adataiból készített kimutatásokat egy letisztult, de zsúfolt felületen tekinthetjük meg. Ebben az esetben is rendelkezésünkre áll a követők száma, a posztok száma, valamint az átlag komment, és like szám is. Az alkalmazás értékeli a profilt olyam szempontok alapján, mint az emberi követők száma, valamint a posztok mennyisége, és minősége. A NotJustAnalytics-hez [3] hasonlóan itt is megtekinthetjük a profil posztjait like szám szerint rangsorolva, és láthatjuk a posztok leírásában legtöbbsör használt hashtag-eket, és kulcsszavakat. Ugyanakkor e rendszer esetében láthatunk egy listát a mi profilunkhoz hasonló profillal rendelkező Instagram felhasználókról.

Sajnos az ingyenes szinten a megszokott grafikonokból csak egyet tudunk megtekinteni, ez pedig a posztok posztolásának idejét tartalmazza, valamint egy tájékoztatást arról, hogy melyik időpontban kirakott poszt volt a legnépszerűbb. (2.16. ábra) A szokáso grafikonok megtekintéséhez itt szükség van előfizetésre. Ezért nem láthattam a követő szám, és a posztokra érkező like szám ingadozásáról készült ábrákat. A többi fentebb már taglalt funkció is a különböző előfizetési szintekhet tartozik, amelyekből ebben az esetben is több áll rendlekezésre. hogy mindenki megtalálhassa a neki optimális megoldást.

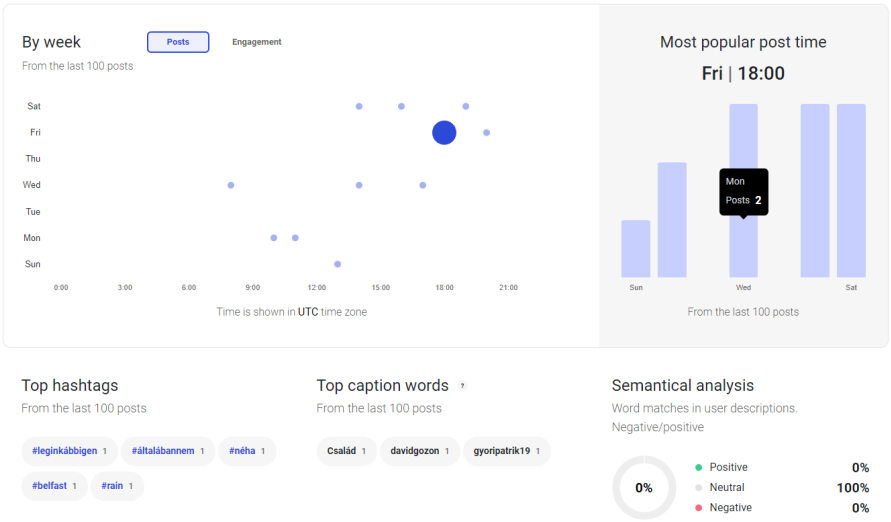
Mindezek után úgy gondolom, hogy az Inflact [4] leginkább olyan Inflencereknek, tartalomgyártóknak, vagy üzleteknek hasznos, akik könnyebbé akarják tenni a fiókjuk menedzselését. Az alkalmazás több olyan szolgáltatással rendelkezik, amik lehetővé teszik a profil, vagy valamilyen termék hirdetését és népszerűsítését. Ezentúl a felhasználó rendelkezésére állnak olyan funkciók is amelyek megkönnyítik a követőkkel való kommunikálást, és kapcsolattartást. Elmondható, hogy a kezelőfelület letisztult, és használata könnyedén elsajátítható, habár 1-2 funkciót a kelleténél jobban elrejtettek.



2.14. ábra: Profil keresése



2.15. ábra: Profil adatok



2.16. ábra: Grafikon a posztokról

2.3. Összegzés

Az applikációkat ezen főbb szempontok alapján hasonlítottam össze:

- Kezelőfelület
- Kinyert adatok ábrázolása
- Feldolgozási sebesség
- Elérhetőség

Ami az alapvető funkciókat és felület felépítést illeti, az InsTrack [1]szinte tökéletes példa arra, ahogyan a feladatom végeredményét elképzelem. A kezelőfelület egyszerű, célratörő, és kézreáll. Ettől egészen kicsivel van csak lemaradva a NotJustAnalytics [3], amelynek fő funciói teljesen megegyeznek az először vizsgált rendszerével. A Keyhole [2], és az Inflact [4]esetében nem ennyire egyértelmű a helyzet. Tekintve, hogy ezek több funkciót is nyújtanak, így bonyolultabbá válik a kezelőfelületük is. A kinyert adatokat hasonló módon ábrázolja a 4 vizsgált alkalmazás, de összességében az InsTrack [1], és a NotJustAnalytics [3] az én szempontomból fontosabb információkat közöl átláthatóbb formában.

A feldolgozás sebességére nem lehet panasz az első három alkalmazás esetében, hiszen a keresés után másodpercekkel a felhasználó rendelkezésére állnak az feldolgozott adatok. A Inflect [4] esetében azonban az elemzés befejezésre esetenként 30 másodpercet is várni kellett. Elérhetőség szempontjából sokkal jobban teljesít az először, és harmadjára vizsgált alkalmazás, hiszen ezeknél az alap funkciók kipróbálásához nincs szükség regisztrációra, és az előfizetés nélkül kipróbálható szolgáltatások száma is nagyobb. A második és főleg a negyedik vizsgált alkalmazás rosszabbul teljesít ilyen tekinteteben, mivel korlátozott az ingyenesen kipróbálható funkciók száma. Tagadhatatlan azonban, hogy a Keyhole [2] egy sokrétűbb szolgáltatást nyújt, így érthető hogy többe kerül a használata, és amellet se mehetek el, hogy az Instack [1] páratlan segítséget nyújt egy Instagram profil menedzselésében de az én feladatom szempontjából ezek a plusz szolgáltatások nem voltak fontosak.

Alkalmazások	**InsTrack**	**Keyhole**	**NotJustAnalytics**	**Inflact**
Kezelőfelület	10	8	9	8
Kinyert adatok ábrázolása	10	6	10	6
Feldolgozási sebesség	10	10	10	6
Elérhetőség	8	5	7	7

2.1. táblázat: Hasonló rendszerek értékelés 1-10 skálán

2.4. Virtualizáció

2.4.1. Virtualizáció

Nagy teljesítményű hardvereknél fontos, hogy az általa képviselt erőforrásra ne csak egy egységként lehessen tekinteni, hanem fel lehessen bontani kisebb egységekre, ahhoz hogy teljesen kihasználhassuk a számítási kapacitását. ([5]) Erre ad megoldást a virtualizáció, ami egy olyan technológia, amely lehetővé teszi, hogy egy fizikai erőforrást több virtuális erőforrásra osszunk szét. Ez lehetővé teszi az erőforrás hatékonyabb kihasználását, hiszen a teljes számítási kapacitást osztja el virtuális gépek között, így már rugalmasabban lehet elosztani, valamint a skálázhatóságot is lehetővé teszi. Továbbá a virtuális gépek igény szerint konfigurálhatóak, így könnyebben lehet őket speciális esetekben használni, és lehetővé teszik az üzleti igényekre való gyorsabb reagálást.

Erőforrásokat skálázni lehet horizontálisan és vertikálisan. Az első esetben újabb és újabb erőforrásokat üzemelünk be a teljesítmény növelésének érdekében, ezek lehetnek fizikai vagy virtuális erőforrások is. A második esetben a már meglévő hardvert fejlesztjük újabb, korszerűbb komponensekkel. Az első megoldás általában nagyobb teret hagy a későbbi fejlesztéseknek, de például ha limitált a fizikai hely, akkor csak a horizontális skálázás jöhet szóba.

Az on-premise fizikai hardver beszerzése és fenntartása költséges, valamint az üzleti igények váratlan változásával a hardver könnyedén alulkihasználtá válhat. Erre kínál megoldást a virtualizáció, hiszen összevonhatunk virtuális gépek segítségével két olyan erőforrást, amelyek kihasználtsága tényleges kapacitásuknak csupán harmada. Az így felszabadult hardvert le lehet állítani, és amennyiben a jövőben megnövekedik a terhelés, újra be lehet üzemelni. Így a cég fenntartási költséget spórolhat, vagy akár el is adhatja a hardvert. Előnyt jelent az is, hogy az egyes virtuális gépek izolált környezetet biztosíthatnak az alkalmazások és a felhasználók számára, ez pedig nagyban növeli a biztonságot. ([6])

Tagadhatatlan azonban, hogy egy virtualizációs infrastruktúra kiépítése nagy szakértelmet és gondos tervezést igényel, valamint lehetnek olyan esetek, amikor egy nagy teljesítményű erőforrásra van csak szükség. A gondatlan tervezés és üzemeltetés nagy biztonsági kockázatot jelenthet.

2.4.2. Konténerek

Mint már fentebb említettem, a virtuális gépek lehetővé teszik számítástechnikai erőforrások virtualizációját, de nem minden esetben ez az optimális megoldás. A virtuális gép egy

egész izolált operációs rendszert virtualizál az ehhez szükséges memóriával, processzorral, és tárhellyel együtt. A manapság népszerű mikroszolgáltatás alapú fejlesztés esetén nem optimális az egyes mikroszolgáltatásokat virtuális gépekben izolálni és üzemeltetni, mivel a mikroszolgáltatásoknak gyorsan és rugalmasan kell skálázódniuk a megváltozott terheléshez képest. Ehhez a feladathoz ezek az eszközök nem garantálnak elegendő rugalmasságot, gyakran lassú indulással kell számolni, és nem gazdaságos a fenntartásuk.

Említésre került már, hogy a virtuális gépek egy egész operációs rendszert és a működéséhez szükséges erőforrásokat virtualizálják. Ezzel ellentétben a konténerizáció során csak egy alkalmazás és annak a környezete kerül virtualizálásra. Ez azt jelenti, hogy több konténerizált alkalmazás fog osztozni egy operációs rendszeren és ugyanazokon az erőforrásokon, ami nagyobb rugalmassághoz és kevesebb kihasználatlan erőforráshoz vezet. Nagy előnye továbbá ennek a technológiának, hogy hordozhatóvá teszi az alkalmazást, így elindítható az alatta futtatott operációs rendszertől függetlenül. Azért képes erre, mivel az operációs rendszertől teljesen izolált és magában hordozza a szükséges függőségeket és könyvtárakat, amelyekre szüksége van az alkalmazásnak. ([7])

Összefoglalva tehát a konténer egy remek eszköz, amely gyorsabb indulást, jobb erőforrás kihasználást és rugalmasságot biztosít. Meg kell jegyezni azonban, hogy bizonyos esetekben nem ez lesz a megfelelő megoldás, és az izoláció is kisebb, mivel a konténerek egy operációs rendszeren osztoznak, így bizonyos hibák befolyásolhatják a többi konténer működését. ([8])

2.5. Felhőszolgáltatók

2.5.1. Felhőszolgáltatókról általánosságban

Mára már rengeteg vállalat és felhasználó veszi igénybe a különböző felhőszolgáltatók sokféle szolgáltatását. Felhőszolgáltatók alatt olyan cégekre gondolok, amelyek lehetővé teszik a felhasználóik számára, hogy alkalmazásokhoz vagy különböző informatikai erőforrásokhoz férjenek hozzá interneten keresztül. ([9]) Gyakori, hogy cégek így jutnak tárolási vagy számítási kapacitáshoz, ahelyett hogy dedikált hardvert vásárolnának és üzemeltetnének. A felhőszolgáltatások rugalmasságot biztosítanak a felhasználónak, hiszen az erőforrások igény szerint skálázhatóak. Amennyiben a skálázás hozzáértő módon van megtervezve, elkerülhető, hogy túl sok vagy éppen túl kevés erőforrás dolgozzon egyszerre. Így nem csak költségcsökkenés, de akár jobb felhasználói élmény is elérhető.

Ha egy vállalat, amellett dönt, hogy egy az addig használt fizikai alapokon nyugvó infrastruktúrát részben vagy teljes egészében felhő alapokra helyezné, a háttérben nagy valószínűséggel a költségcsökkentés igénye áll. Kisebb startupoknak döntő lehet az in-

dok, hogy nem kell a cég életének korai szakaszában az infrastruktúra kiépítésére nagyobb beruházási költséget szánni, hanem a gyakran hullámzó felhasználói igényeknek megfelelően lehet skálázni az erőforrásokat. Ezenkívül előnyt jelent az is, hogy a felhőszolgáltató erőforrás központjai a világ sok pontján helyezkednek el, így biztosítva földrajzi biztonságot, valamint jobb elérhetőséget az ügyfeleiknek. Hasznos funkció az is, hogy a gyakori automatikus mentéseknek hála egy hirtelen, nem eltervezett leállás után könnyedén visszállíthatóak a gépek, egy korábbi verzióra, így javítva a felhasználói élményt, és csökkentve az adatvesztés lehetőségét. ([10]) Ugyanakkor nem feledkezhetünk meg arról sem, hogy amennyiben egy nagyobb leállás következik be szolgáltatói oldalon, az hatalmas károkhoz vezethet a felhasználónál, hiszen amennyiben nem redundáns az infrastruktúra kialakítása, az egész rendszer használhatatlanná válhat.

Annak ellenére, hogy sok esetben előnyös lehet az, hogy csak azért kell fizetni, amit használunk, azonban fontos megjegyezni, hogy a könnyedén nagyon magasra szökhet a végösszeg. Ugyancsak hátrányt jelent az is, hogy így az adatokat egy harmadik félnél tároljuk. Ugyan rengeteg biztonsági intézkedés védi a felhőben tárolt, és kezelt adatokat, de ennek ellenére bizalmas adatokat nem feltétlenül jó ötlet így tárolni, és kezelni. Sok esetben a tárolt adatok bizalmassága miatt ezt törvény tiltja. Hátrányként említhető az is, hogy amennyiben az infrastruktúra jelentős része felhőben van, és kialakul egy függés a szolgáltatótól, az nagy fej fájást okozhat a felhasználónak jövőben, amennyiben a szolgáltató megszűnik, vagy jelentősen árat emel.

Összefoglalva tehát, a felhőszolgáltatások sok problémára kínálnak megoldást, de nem érdemes meggondolatlanul minden esetben ezekhez fordulni, mivel az, hogy egy adott felhasználónak milyen igényei és megoldandó problémái vannak a helyzettől függ. Ezért kiemelten fontos hangsúlyozni, hogy nagy szakértelemre és hosszas tervezésre van szükség, különösen egy nagyobb vállalat esetében, ahhoz, hogy egy ilyen típusú paradigma váltás valóban pozitív végeredménnyel járjon.

2.5.2. Terhelési minták

Mint fentebb is említettem a felhőszolgáltatások, a rugalmasságuk miatt, segíthetnek az infrastruktúra kihasználtságnak optimalizálásában. Négy gyakoribb terhelési mintáról beszélhetünk:

- Predictable Bursting
- Unpredictable Bursting
- On and Of
- Growing Fast

Mint az ábrán is látható, a "Predictable Bursting" esetén időszakosan növekszik meg

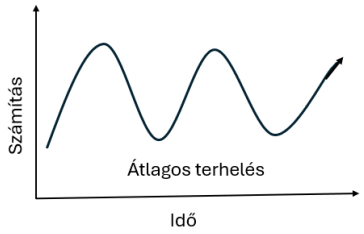
a szolgáltatást használni kívánó felhasználók száma. (lásd 2.17) Ilyen lehet például a "Black Friday", vagy a karácsonyi időszak sok online áruháznak. Ilyenkor a leárazások miatt jelentősen megnő a forgalmuk, ezért rendkívül nagy kapacitású infrastruktúrára van szükségük. Ám az év többi részében ezek az erőforrások kihasználatlanul, parlagon állnának. Erre kínálhat megoldást, ha egy cég az ilyen időszakokra felhő erőforrást kölcsönöz, így elkerülve a felesleges hardver beszerzésének és üzemeltetésének költségeit.

A második minta esetén előre meg nem jósolható időpontban jelentősen megnő a terhelés, és ezt sok esetben nem bírja el az infrastruktúra, ami kimaradásokhoz és elégedetlen felhasználókhoz vezethet. (lásd 2.18) Ebben az esetben mivel a terhelés növekedése nem volt előre látható, és a végtelenségig nem lehet túl biztosítani a rendszert, ezért kiváló megoldást nyújthat valamely felhőszolgáltatás, amely csak akkor indítja el és skálázza az igényekhez mérten a felhő erőforrásokat, amikor a fizikai erőforrások elfogytak.

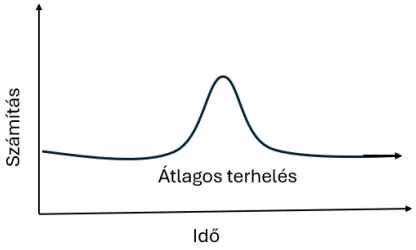
A harmadik esetben az átlagos terhelés ugyan egyenletes, viszont vannak időszakok, amikor semmilyen forgalom sincs. (lásd 2.19) Itt a problémát az jelenti, hogy az átlagos terhelésnél biztosított erőforrások az inaktív időszakok során nincsenek kihasználva.

A "Growing Fast" minta esetén a problémát a szolgáltatás sikere okozza, hiszen a felhasználók száma egyre csak nő, és az ilyen növekedéssel nehéz lépést tartani. (lásd 2.20) Gondot jelenthet az is, hogy meddig növelik a fizikai erőforrások számát, hiszen az, hogy pontosan meddig fog emelkedni a terhelés, nem megállapítható, és ha az erőforrások száma jelentősen meg lett növelve, de a felhasználók száma a jósoltnál hamarabb kezd el apadni, akkor ismét felesleges kiadásokba futott a cég. Ez a fajta minta jellemző felfutóban lévő startupokra, ahol a felhasználók száma kezd magassá válni, és a háttérben lévő infrastruktúra nem bírja el ezt a terhelést. Kiutat jelenthet, ha bizonyos szolgáltatások és erőforrások helyét felhőmegoldások veszik át, hiszen rugalmasságuknak hála egy kis teret nyerhet magának a cég, amíg a forgalom egyenletesebbé válik.

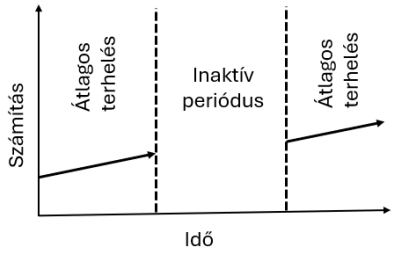
Az itt ismertetett mintákra sok esetben jó megoldást jelenthetnek különböző felhőszolgáltatások, és sok cég él is ezekkel. Azonban, mint már fentebb is említettem, nem minden esetben jelentenek jó megoldást.



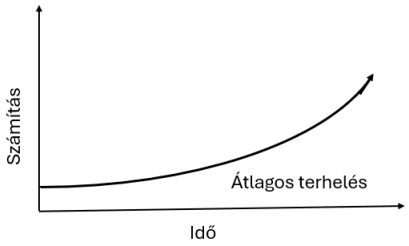
2.17. ábra: "Predictable Bursting" terhelési minta.



2.18. ábra: "Unpredictable Bursting" terhelési minta.



2.19. ábra: "On and Off" terhelési minta.



2.20. ábra: "Growing Fast" terhelési minta.

2.5.3. Szolgáltatási modellek

Szolgáltatásaik széles skáláján sok minden fellelhető, és ezen szolgáltatásokat 3 nagyobb csoportra lehet bontani:

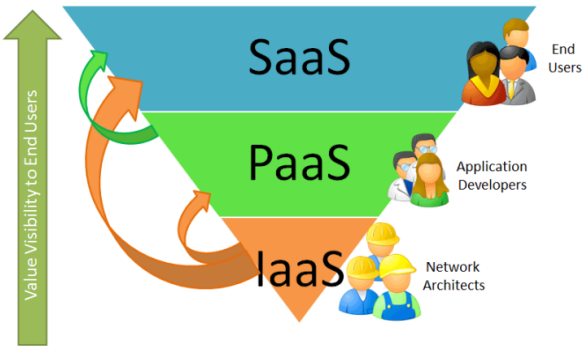
- Software as a Service(SaaS)
- Platform as a Service(PaaS)
- Infrastructure as a Service(IaaS)

A legnagyobb különbséget az jelenti a három modell között, hogy milyen szintű az absztrakció, amely elrejti a háttérfolyamatokat. Ezalatt azt értem, hogy a szolgáltató mennyire rejti el a háttérfolyamatokat a felhasználó elől. Mint az ábrán is látható, az IaaS biztosítja a legszélesebb körű hozzáférést a három modell közül, hiszen itt a felhasználó szabadon létrehozhat és kezelhet erőforrásokat a felhőszolgáltató által biztosított infrastruktúrában. (lásd 2.21) Természetesen ehhez szükséges a legnagyobb szakértelem is, mivel itt több lehetősége van a felhasználónak arra, hogy komolyabb hibát vétsen egy

infrastruktúra felépítése során. Ide tartozik például az AWS és az Azure is, amelyek lehetővé teszik, hogy a felhasználó az általuk biztosított erőforrásokból egy testre szabott rendszert hozzon létre. ([11]) Itt a legalacsonyabb az absztrakció szintje, hiszen ebben az esetben csak a fizikai erőforrások és az ezeket összehangoló szoftverek vannak elrejtve a felhasználó elől. Természetesen ezek a platformok rengeteg szolgáltatással rendelkeznek, amelyek segítik az infrastruktúra létrehozását, monitorozását, valamint karbantartását.

Mint az ábrán is látszik, ez a modell egyfajta átmenetet képez az IaaS és a SaaS között. (lásd 2.21) A PaaS egy fejlesztési környezetet biztosít a fejlesztőknek, amely segíteni hivatott az alkalmazások fejlesztését, tesztelését és üzemeltetését. Például ha egy online áruház forgalma hirtelen megnő, akkor automatikusan több erőforrást rendel az alkalmazáshoz, majd a forgalom apadásával csökkenti azt. További automatizációs megoldások is a felhasználó rendelkezésére állnak, olyan műveletek esetén mint az alkalmazások telepítése, frissítése és konfigurálása. Előnyt jelenthet az is, hogy a fejlesztés megkezdésekor nem kell a fejlesztési környezetet előkészíteni és konfigurálni, mivel ezek általában előre konfiguráltak. Sok beépített szolgáltatás is segíti a munkát, ilyenek például az adatbázisok, felhasználókezelés vagy az integrációt segítő megoldások. Hátrányt jelenthet azonban, hogy a platformokat nem lehet teljesen testreszabni, és fejlesztők csupán korlátozott kontrollal rendelkeznek a környezet konfigurálásában. Ezenkívül itt is felmerülhetnek aggályok a biztonsággal kapcsolatban, hiszen a szolgáltató felel az alkalmazás és az abban felhasznált adatok védelméért, és ez bizonyos esetekben biztonsági kockázatot jelenthet.

A legmagasabb szintű absztrakció az SaaS esetén fordul elő, mivel itt a felhasználó csak egy kész alkalmazást használ, amit felhőkörnyezetben érhetnek el, tehát csupán korlátozott felhasználói beállításokhoz férnek hozzá. Ez azt is jelenti, hogy a felhasználónak nem kell bajlódni a szoftverek karbantartásával, és frissítésével, mivel ez a szolgáltató feladata. Ez azzal is jár, hogy sok esetben nem dönthet a felhasználó arról, hogy milyen változtatásokat hajtanak végre az alkalmazáson. Az alkalmazáshoz dedikált eszközökről, alkalmazásokból vagy gyakran böngészőből lehet hozzáférni. Ilyen alkalmazás szolgáltatás például a "Microsoft 365" is, amely tartalmazza a mindennapi irodai munkához és munkahelyi kommunikációhoz szükséges alkalmazásokat.



2.21. ábra: A szolgáltatási modellek megértését segítő ábra.

2.5.4. AWS bemutatása

Az AWS a legrégebben indult felhőszolgáltató, ami az Amazon vállalathoz tartozik. Mint már fentebb is említettem, bizonyos időszakokban nagyon megnőhet egy weboldal terhelése. Ez az Amazon esetében is igaz volt, hiszen karácsony vagy "Black Friday" alkalmával nagyon megugrott a felhasználók száma, és ahhoz, hogy ezt a terhelést elbírják, megnövelték a háttérben dolgozó erőforrások méretét és teljesítményét. Ennek a hátránya, hogy az év többi részében, amikor kisebb volt a terhelés, nem voltak rendesen kihasználva. Erre a problémára azt a megoldást találták ki, hogy létrehoznak olyan szolgáltatásokat, amelyeket más vállalatok vagy cégek ki tudnak bérelni, és így plusz erőforrásokhoz juthatnak az interneten keresztül. ([12])

Az első két szolgáltatás, amit nyújtott az AWS, az "Elastic Compute Cloud" valamint a "Simple Storage Service" voltak. Ezek lehetővé tették, hogy a felhasználók különböző teljesítményű virtuális szervereket hozzanak létre, valamint adatokat tároljanak a felhőben. Ezt a két alapvető szolgáltatást még sok másik követte az évek során. Eleinte a szakma szkeptikusan állt hozzá a felhőszolgáltatásokhoz, de az idő az Amazon-t igazolta, és mára már több felhőszolgáltató közül választhatunk az igényeinknek megfelelően.

2.5.5. Azure bemutatása

Az Azure a Microsoft válasza az AWS-re, tehát ez is egy felhőszolgáltatási platform, ahol a felhasználók hozzáférhetnek IaaS, SaaS és PaaS szolgáltatásokhoz is. Mivel pár évvel később indult, ezért egy ideig jócskán le volt maradva a szolgáltatások számát illetően, de mára az AWS legfőbb versenytársává nőtte ki magát. Így a felhasználó létrehozhat különböző teljesítményű virtuális szervereket (Azure Virtual Machines), rendelkezésre állnak fejlesztői környezetek, valamint felhőadatbázisok (Azure App Service, Azure SQL

Database), és olyan kész szoftvercsomagokhoz is hozzáférhet, mint a Microsoft 365.

Az Azure előnyére válik, hogy irodai környezetben rengeteg cég használt, és használ Microsoft termékeket, ez megkönnyíti bizonyos folyamatok felhőbe való áthelyezését olyan vállalatoknak, ahol eddig is Microsoft szoftvereket és technológiákat használtak. Ez azt is jelenti, hogy az Azure függ a Microsoft ökoszisztémájától, ilyen szempontból az AWS egy kicsit nagyobb szabadságot és rugalmasságot kínál az ügyfeleinek. Az Azure kiváló hibrid felhőmegoldásokat kínál, ami a felhasználóknak olyan környezetet biztosít, ahol egyidejűleg tudják használni mind a helyszíni infrastruktúrát, mind pedig a felhőre épülő infrastruktúrát és szolgáltatásokat. Ez tulajdonképpen az arany középutat biztosítja, ahol a cég a bizalmas adatokat tárolhatja és kezelheti a helyszíni infrastruktúrával, azonban rendelkezésére áll egy jól skálázható, rugalmas környezet is. Az AWS is rendelkezik hasonló szolgáltatással, de ez több cikk szerint is elmarad tudásban a Microsoft megoldásához képest. [13]

2.5.6. AWS és Azure összehasonlítása

Az alapvető szolgáltatások, amelyek ahhoz szükségesek, hogy a felhasználó virtuális gépet, tárhelyet és adatbázist béreljen, valamint kezeljen mindkét szolgáltató esetén megtalálhatóak. A különbségek inkább abban mutatkoznak meg, ahogyan specifikus funkciókat valósítanak meg, és hogy milyen területekre fókuszálnak. Mint fentebb is említettem, az Azure, tekintve hogy egy Microsoft termék, erősen integrált a Microsoft ökoszisztémájába, míg az AWS sokkal több lehetőséget kínál ilyen téren. Természetesen Azure használata esetén is van lehetőségünk Microsoft-tól független szoftverek, könyvtárak és programozási nyelvek használatára, és megoldható az is, hogy hibrid Windows-Linux rendszereket üzemeltessünk. Open-source projektekhez ellenben sokkal jobban használható az AWS, mivel kompatibilis Linux-szal, és sok open-source alkalmazást támogat. Az Azure lehetőséget biztosít a vállalatoknak, hogy a már létező könyvtáraikat a felhő környezetükben futtassák.

Az elmúlt években egyre fontosabb témává vált a gépi tanulás és a mesterséges intelligencia, viszont a különböző modellekhez nagyon nagy számítási kapacitásra van szükség. Erre megoldást kínál mindkét szolgáltató, de míg az AWS megoldása kifejezetten már tapasztalt és mély szakmai tudással rendelkező fejlesztőket célozza, addig a Azure-nél kód nélkül lehet gépi tanulás modelleket építeni. Fontos különbség az is, hogy hogyan osztják el és használják a virtuális gépek erőforrásait. Az AWS esetében minden folyamathoz ugyanonnan rendelnek számítási kapacitást, és amennyiben növelni kell ezt a mennyiséget, innen kapnak plusz erőforrást. A virtuális gépet egy "Machine Image"-ből készíti el, amiben a felhasználó az igényeihez igazíthatja, hogy mennyi tárhellyel, memóriával és milyen teljesítményű processzorral rendelkezzen a gép. Azure esetén a felhasználó a

virtuális gépet egy virtuális merevlemezről hozza létre, és kevesebb beállítási lehetősége van. Az Azure esetén az is korlátozza a virtuális gépek rugalmasságát, hogy gyakran szorosan együtt kell működjenek más felhő szolgáltatásokkal, és ez valamelyest korlátozza a virtuális gépek független működését, de segíti a felhőalapú fejlesztést, és gyakran jobb nagy vállalati használatra

Az én esetemben a legfontosabb szempont, ami alapján össze kell hasonlítanom őket, hogy milyen szolgáltatásaik teszik lehetővé a kitűzött célok elérését. Ezért kiemelt figyelmet szenteltem a két felhőszolgáltató által kínált PaaS szolgáltatásoknak, az AWS Elastic Beanstalknak és az Azure App Service-nek. Mindkét szolgáltatás lehetővé teszi az alkalmazások felhőben való fejlesztését, telepítését és üzemeltetését. Mint már fentebb is említettem, ennek a legnagyobb előnye, hogy a fejlesztőnek nem kell az infrastruktúra kialakításával és karbantartásával törődnie. Sok esetben előny, ha nagyobb ráhatással bírunk az infrastruktúra felépítésére, de arra jutottam, hogy nekem ebben az esetben nincs erre szükség

Feladat	**AWS**	**Azure**
Adatok felhő alapú tárolása	AWS Relational Database Service biztosít egyszerű relációs adatbázist	Azure SQL Database biztosít egyszerű relációs adatbázist
Rendelkezésre állás	AWS Elastic Beanstalk biztosítja az infrastruktúrát az alkalmazás fejlesztéséhez, és üzemeléséhez	Azure App Service biztosítja az infrastruktúrát az alkalmazás fejlesztéséhez, és üzemeléséhez
Automatikus lekérdezések futtatása	AWS CloudWatch Events események, vagy idő alapján alapján képes műveleteket végrehajtani	Azure Functions események, vagy idő alapján hajt végre műveleteket

2.2. táblázat: Azure és AWS összehasonlítása a feladatlap alapján

Mint a táblázatban is látszik (2.2) mindkettő szolgáltató biztosítja a nekem fontos szolgáltatásokat. Ebben az esetben a különbség leginkább az ár számításában, és a támogatott nyelvek, könyvtárak felhozatalában különbözik.

2.6. Keretrendszerek

2.6.1. Általánosságban a keretrendszerekről

A keretrendszerek feladata, hogy egyszerűbbé tegyék a fejlesztők munkáját eszközök és struktúrák gyűjteményével [14]. Biztosítanak egy keretet, így nem kell a nulláról kezdeni a fejlesztést. Napjainkban rengeteg keretrendszer áll rendelkezésre, és a választás nehéz lehet. Általában egy keretrendszer több komponensből épül fel, és segítenek olyan gyakran előforduló feladatok elvégzésében, mint az alkalmazás telepítése vagy tesztelése. Ezentúl a keretrendszerek segíthetnek a kód biztonságosságának növelésében, és elősegítik a "best practice" követését.

Természetesen hátrányai is vannak a keretrendszereknek. Mint említettem, sok keretrendszer áll rendelkezésünkre, azonban a legtöbb különbözik egymástól, így sok esetben újra kell tanulni a használatukat. Ez természetesen idővel és rutinnal könnyebbé válhat. Valamint frissítések során nagy változások következhetnek be, ilyenkor ezekhez a változásokhoz is hozzá kell szokni. Azt is meg kell említenem, hogy ugyan remek, hogy ennyi eszköz áll rendelkezésünkre, ám ez a teljesítmény és a rugalmasság kárára megy.

2.6.2. Keretrendszer választása

Szerettem volna egy JavaScript-re épülő keretrendszert kiválasztani az alkalmazás kezelőfelületének építéséhez, valamint egy C Sharp-ra vagy Python-ra épülő rendszert a háttér folyamatok és API hívások kezeléséhez. Mivel már mind Python, mind pedig C Sharp nyelvekkel volt tapasztalatom, ezért a .NET és a Django keretrendszerek között végül az döntött, hogy a .NET keretrendszert sokkal könnyebb Azure-el együtt használni. Másik keretrendszernek a React JS-t választottam, mivel mindig is meg akartam tanulni a használatát, valamint sok információ és dokumentáció érhető el róla.

2.7. Irodalomkutatás Összegzése

Ennek a fejezetnek a célja, hogy összefoglaljam az irodalomkutatás eredményét. Elsőnek hasonló rendszereket kerestem, és vizsgátam meg alaposabban. Ezáltal láttam példákat arra, hogy milyen az én feladatomhoz hasonló feladatot ellátó programok, milyen kezelőfelületet használnak, és milyen olyan funkciókkal rendelkeznek, amiket szívesen látnék a saját programomban is. Ezen alkalmazások vizsgálata során a 2 legfontosabb szempont a kezelőfelület felépítése, és a feldogozott adatok ábrázolása voltak. (2.1) Mivel az alkalmazásomnak felhő alapokon kell állnia, ezért kutató munkám során kiemelt fontosságot

élveztek a virtualizáció, és felhőszolgáltató témakörök. Az elmúlt pár évben a fizikai erőforrások virtualizációja, valamint a bizonyos folyamatok felhő platformokra való költöztetése bevett szokássá vált költség csökkentés céljából. Ezeket a témákat jobban is kifejtem a virtualizációról, valamint a felhőszolgáltatókról szóló fejezetekben. Írtam továbbá még a konténerizációról is, ami egy napjainkban rengeteget használt technológia, és kitértem a felhőszolgáltatások esetén előforduló terhelési mintákról, és szolgáltatási modellekről is. Ezenkívül összehasonlítottam az AWS és az Azure azon szolgáltatásait, melyek biztosítanak felhő alapú adattárolást, rendelkezésre állást, és automatikus lekérdezések futtatását. (2.2) Az összehasonlítás végére arra jutottam, hogy ebben az esetben az Azure lesz a számomra megfelelő felhőszolgáltató, mivel rendelkezik ugyanazon szolgáltatásokkal, mint az AWS és .NET keretrendszerrel nagyon jól lehet együtt használni. A keretrendszerek témakörében is végeztem kutatást, és végül arra jutottam, hogy a kezelőfelület létrehozásához React.js, míg a háttérfolyamatok kezeléséhez .NET keretrendszereket fogok használni. Úgy érzem, hogy a kutatómunka segített jobban megérteni a feladatomat, és felnyitotta a szemem sok olyan problémára, amelyekkel fájdalmas lett volna először a fejlesztés közben találkozni. Ezenkívül kapta egy alapot a tervezés megkezdéséhez.

3. KÖVETELMÉNY SPECIFIKÁCIÓ

3.1. Követelmény specifikáció

A szakdolgozatom során az a célom, hogy egy olyan rendszert készítsek, aminek segítségével a felhasználók lekérdezhetnek adatokat egy Instagram influencer profiljáról, és ezeket az adatokat statisztikák és diagramok formájában láthatják. Ha lebontom ezt a feladatot kisebb egységekre, akkor a rendszernek ezeknek a követelményeknek kell megfelelnie.

Először is a rendszernek biztosítania kell egy könnyen értelmezhető és használható kezelőfelületet, ahol rákereshetnek az influencerekre az Instagram azonosítójuk alapján. Továbbá szükséges egy funkció, ami lehetővé teszi, hogy az alkalmazás képes legyen az Instagram influencerek like- és követőszámának lekérdezésére, valamint ezek eltárolására. Az adatokat egy felhőalapú adatbázis szolgáltatásban kell eltárolnom, ilyen például az Azure SQL Database szolgáltatás. Fontos továbbá, hogy a kinyert adatokból emberi szem számára könnyen értelmezhető statisztikákat és diagramokat készítsen a webalkalmazás.

Fontos követelmény továbbá, hogy a webalkalmazás mindig elérhető legyen a felhasználó számára. Ehhez egy felhőszolgáltató valamely szolgáltatására lesz szükség, ilyen például az Azure App Service. Ugyancsak felhő alapon meg kell oldanom, hogy a felhasználó képes legyen automatikus lekérdezések beállítására. Ezek a lekérdezések automatikusan kell, hogy lefussanak a felhasználó által megadott időközönként.

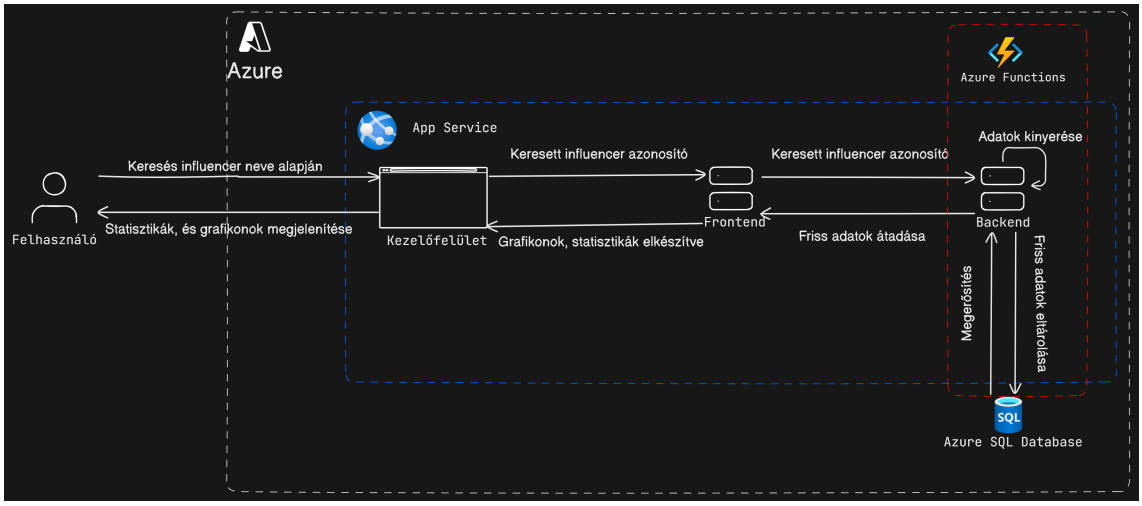
Ezenkívül fontos még az alkalmazás futása során felmerülő hibák nyomonkövetése és feljegyzése. Valamint arról sem feledkezhetünk meg, hogy a felhasználó visszajelzéseket kapjon arról, hogy ha valami hiba következik be, és amennyiben ez az ő hibája, akkor útmutatást kapjon az alkalmazás helyes használatához.

4. TERVEZÉS

4.1. Rendszerterv

Ahhoz, hogy a fentebb ismertetett funkciókat biztosítsa a rendszer, az alábbi infrastruktúrát terveztem meg (4.1). Ezen az ábrán jól látható, hogy milyen nagyobb egységekre osztottam fel a feladatot. A frontend feladata a kezelőfelület biztosítása és a kinyert adatok ábrázolása statisztikák és grafikonok formájában. Ezeket az adatokat a backend-től kapja meg, amelynek legfőbb feladata, hogy API hívásokkal lekérdezze a szükséges adatokat az Instagram Graph API-tól és eltárolja ezeket az Azure SQL adatbázisban. Mivel biztosítanom kell egy olyan funkciót, ahol a felhasználó által beállított időközönként le kell futnia a lekérdezésnek, ezért a backendet mint serverless Azure Function App-ot fogom telepíteni és kezelni. Ez lehetővé teszi, hogy vagy esemény bekövetkezése esetén, vagy pedig időzített módon lefusson a lekérdezés. A rendelkezésre állást az Azure App Service segítségével fogom biztosítani.

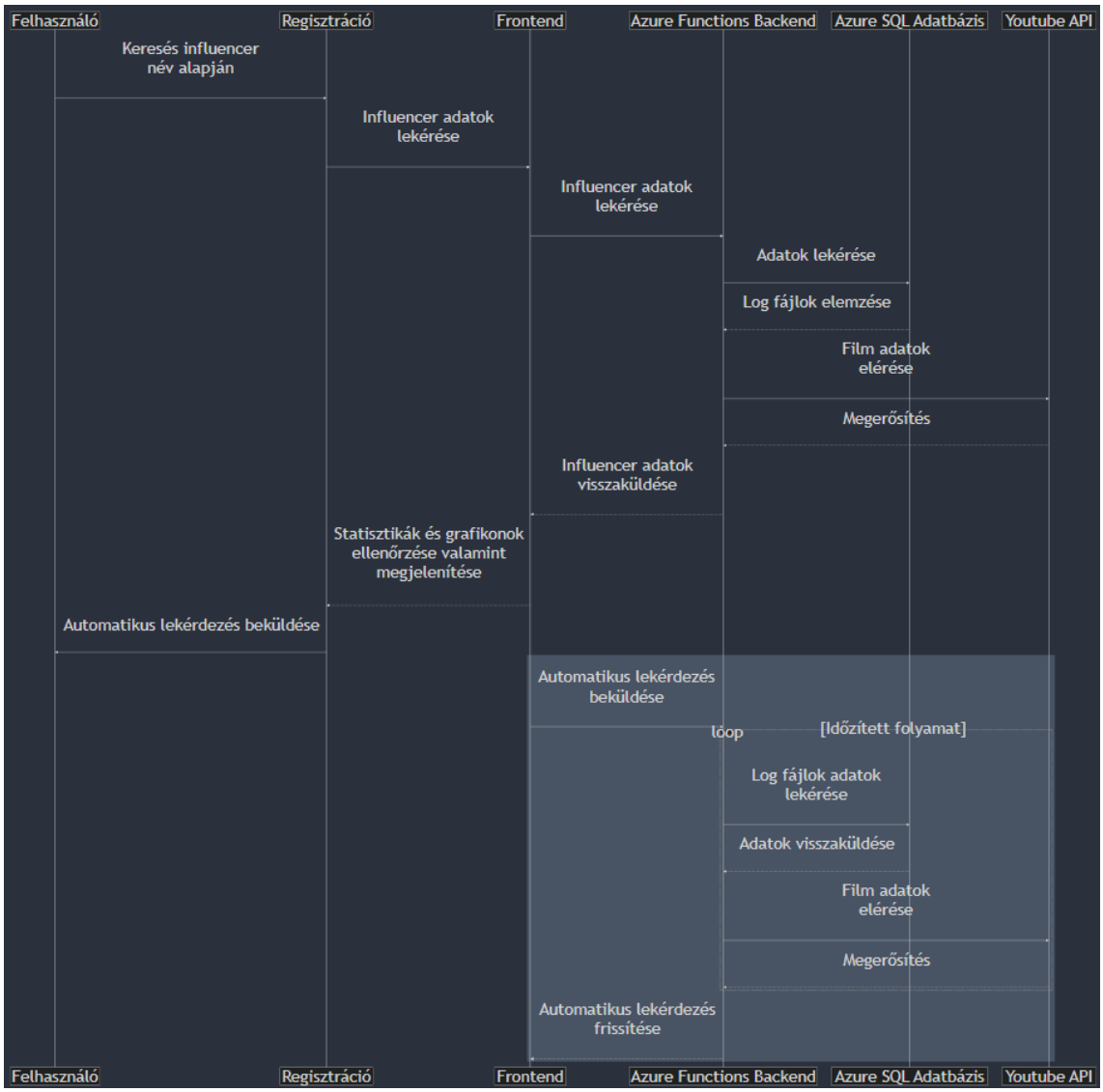
4.1.1. Teljes rendszer diagram



4.1. ábra: Teljes rendszert ábrázoló diagram

4.1.2. Teljes rendszer szekvencia model

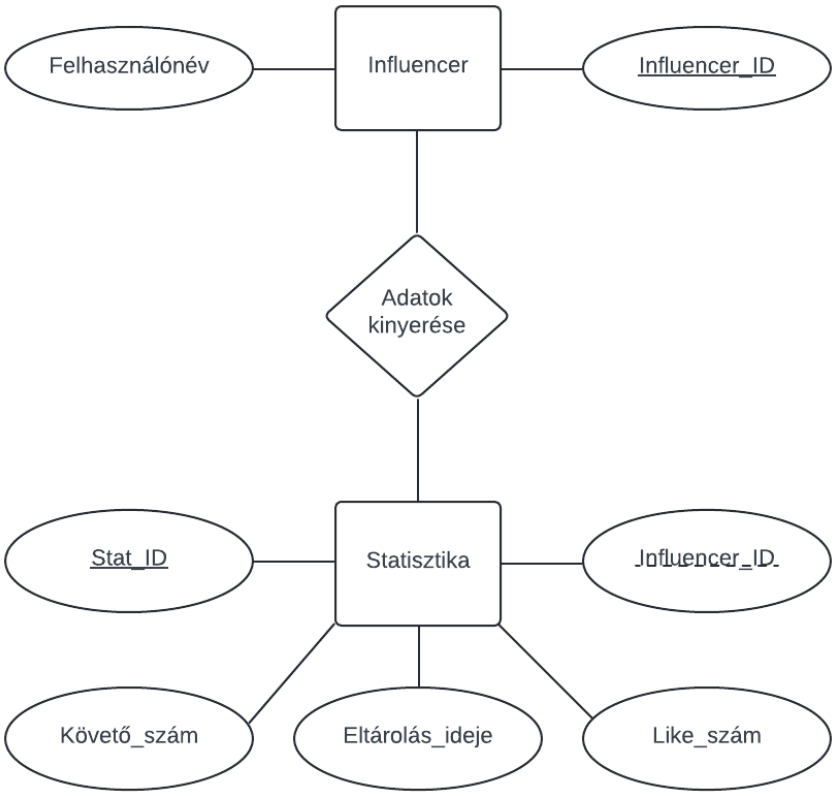
Az alábbi ábrán (4.2) egy átfogó szekvencia model látható arról, hogyan mennek végbe a főbb folyamatok a webalkalmazás futása során.



4.2. ábra: Teljes rendszer működését ábrázoló szekvencia model.

4.1.3. Adatbázis egyed-kapcsolat (EK) diagram

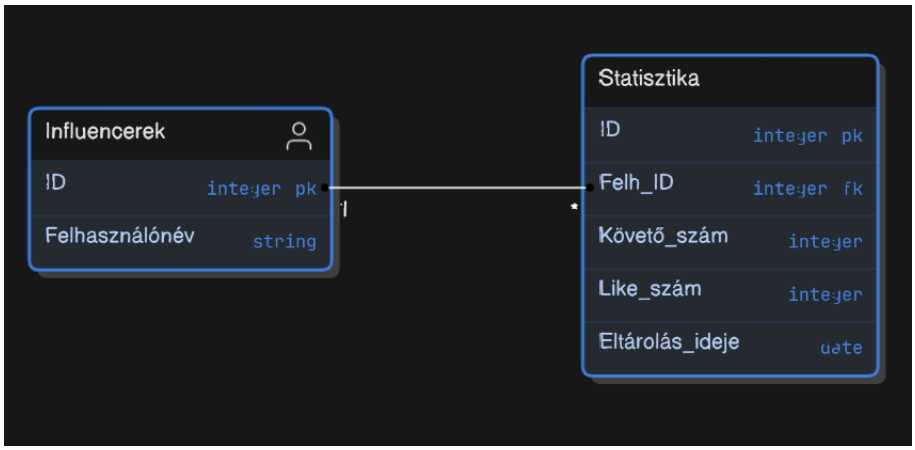
Az adatok tárolására relációs adatbázis modellt tervezek használni, ami jól működik klasszikus SQL adatbázisokkal. Ezen az egyed-kapcsolat diagramon (4.3) az látható, hogy milyen attribútumai vannak a két táblának, amit használni tervezek, valamint a funkció, ami során a kapcsolat létrejön. Mint látható az ábrán, az Influencer entitás két attribútummal rendelkezik: az egyik egy egyedi azonosító, ami kulcsként is funkcionál, valamint egy felhasználónév. A Statisztika nevű entitás tartalmazza az influencerhez tartozó adatokat, egy egyedi azonosítót, valamint egy az influencert azonosító ID-t.



4.3. ábra: Adatbázis EK diagram.

4.1.4. Adatbázis táblák kapcsolata

Mint már fentebb említettem 2 táblában tervezem eltárolni az adatokat. Azért dönöttem 2 tábla mellett, mivel 1 inlfuencerhez tartozhat több statisztika is (4.4), amennyiben a felhazsnáló automatikus, ütemezett lekérdezéseket használ. Ezért van a Statisztika táblában egy az eltárolás idejét rögzítő mező.

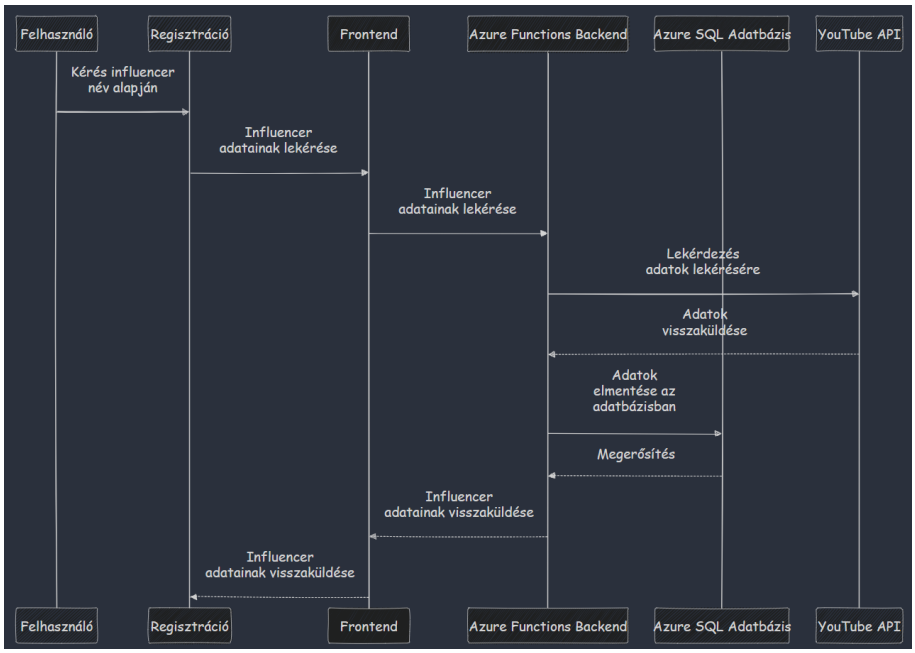


4.4. ábra: Táblák tartalmát és kapcsolatukat ábrázoló ábra.

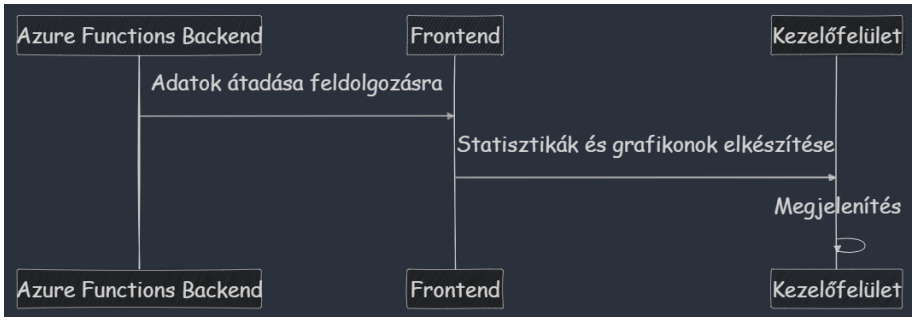
4.2. Funkciólista

- Influencer fiókok keresése és adatok kinyerése (4.5) A felhasználó rákereshet influencerek Instagram nevére, és a rendszer megpróbálja kinyerni és eltárolni az Instagram Graph API-ból a felhasználóhoz tartozó like- és követőszámot.
- Adatok eltárolása felhőalapú adatbázisban (4.5) Miután a rendszer kinyerte az adatokat, eltárolja azokat egy felhőalapú adatbázisban. Az adatok sikeres eltárolásáról visszajelzést küld az adatbázis.
- Kinyert adatok feldolgozása és megjelenítése (**??**) Miután sikeresen le lettek kérdezve és el lettek tárolva az adatok, meg is kell jeleníteni őket a felhasználó számára.
- Influencer fiókok like- és követőszámának automatikus, időzített lekérdezése (4.7) A felhasználónak lehetősége van automatikus, időzített lekérdezéseket indítani. Ilyenkor a rendszer X óránként lekérdezi az adott influencer adatait és eltárolja azokat az adatbázisban.
- Folytonos rendelkezésre állás Fontos funkció, hogy a webalkalmazás a nap bármely szakaszában elérhető és működőképes legyen.
- Hibák nyomonkövetése és feljegyzése Természetesen felmerülhetnek különböző hibák a rendszer futásának különböző szakaszaiban. Ilyen lehet például, ha a felhasználó nem létező influencer nevére keres rá (4.8), hiba merülhet fel az Instagram Graph API-val való kommunikáció közben (4.10), vagy akár az adatok adatbázisba való mentése közben is (4.9).

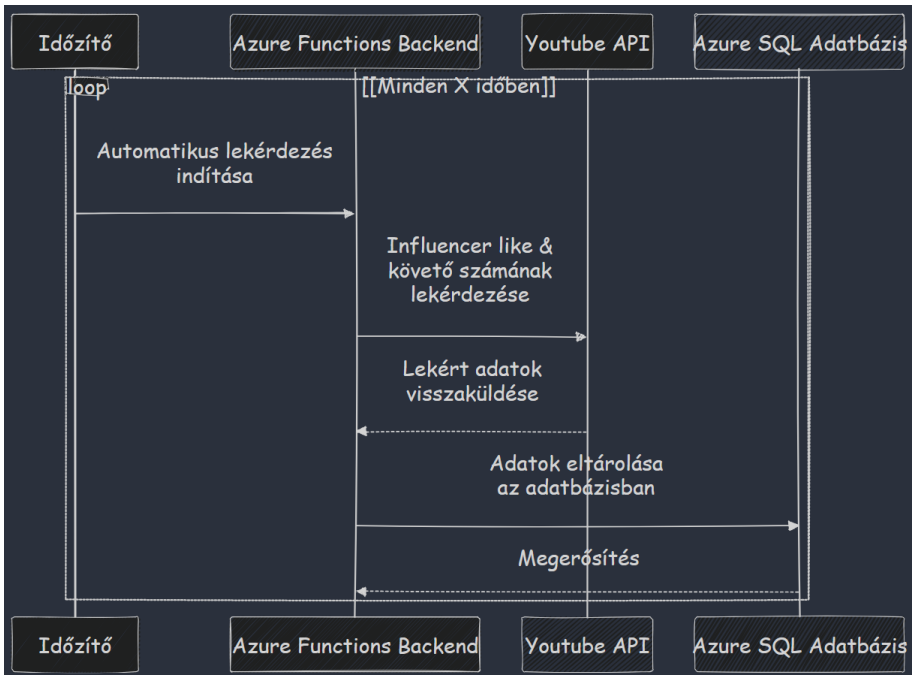
4.2.1. Funkciók megértését segítő ábrák



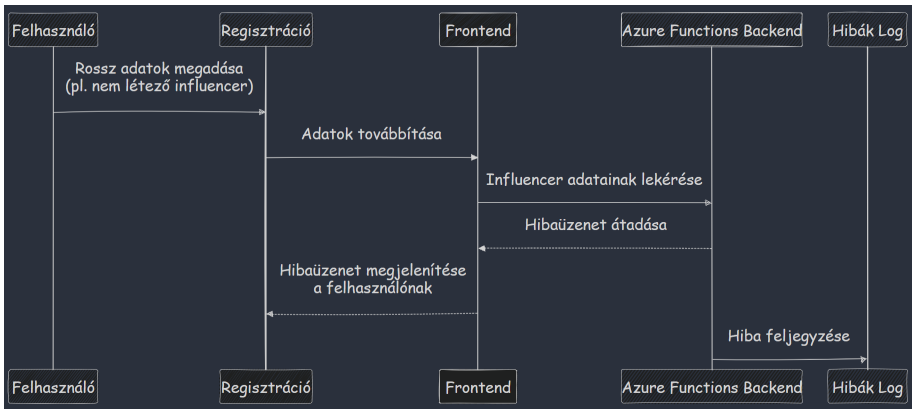
4.5. ábra: Az ábrán az látható, ahogyan egy influencer keresése, és az adatainak eltárolása megtörténik.



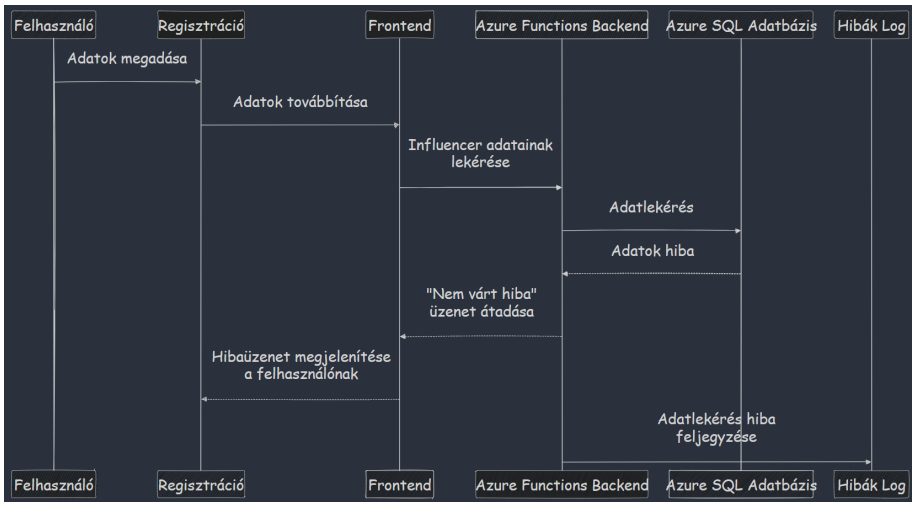
4.6. ábra: Az ábrán az a folyamat látható, ahogyan az adatok kinyerés, és eltárolása után feldolgozásra, majd megjelenítésre kerülnek.



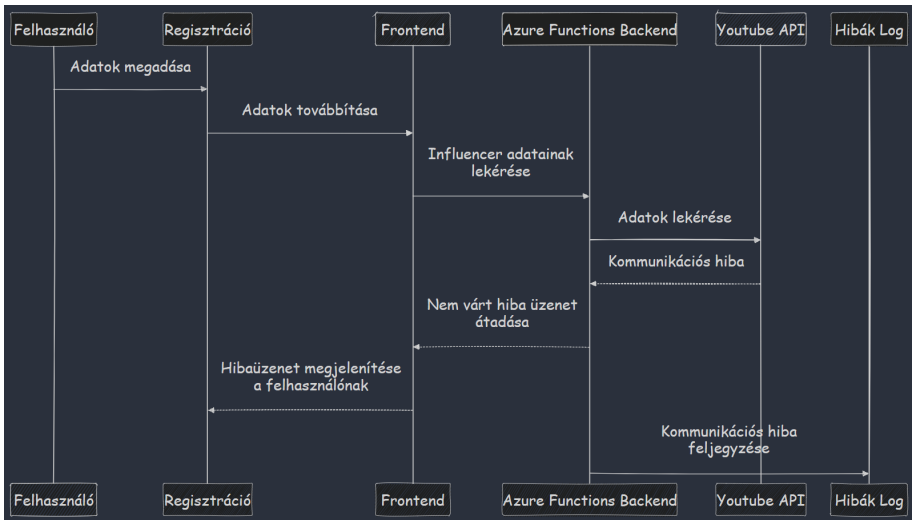
4.7. ábra: Az ábrán az a folyamat látható, amely végbemegy amikor a felhasználó igénybe veszi az automatikus lekérdezés funkciót.



4.8. ábra: A rossz felhasználónév esetén lefolyó folyamat átfogó ábrázolása.



4.9. ábra: Adatok adatbázisba való eltárolása esetén fellépő hiba jelzésének átfogó ábrázolása.



4.10. ábra: Instagram API-val való kommunikáció során fellépő hiba jelzésének átfogó ábrázolása.

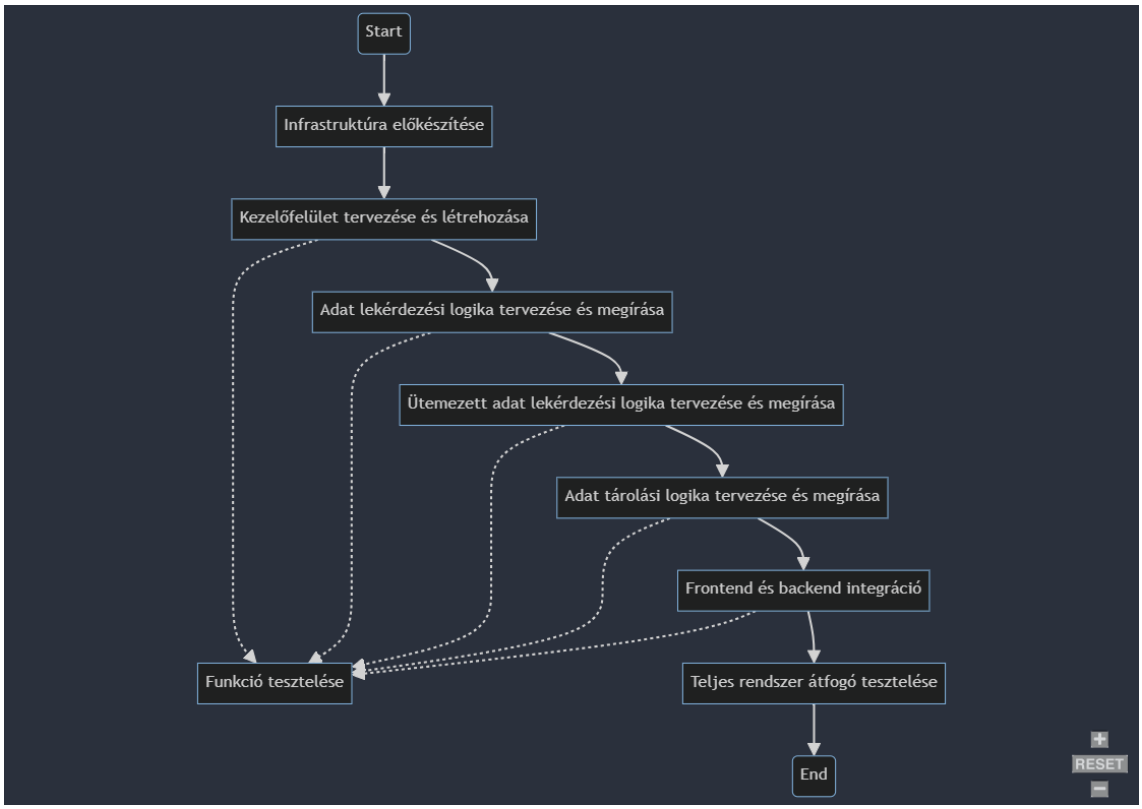
4.3. API végpont lista

HTTP Kérés	Végpont	Várt paraméterek	Visszaadott érték
GET	/api/GetYoutubeData/	username: string, trackChannel: bool	influencer, statisztika adatok
GET	/api/GetChannelDataNoAuth/	username: string, trackChannel: bool	influencer, statisztika adatok
GET	/api/WebAppUserGetData	username: string	követett influencerek, statisztikák, időzített lekérdezés intervallum, lekérdezés munkafolyamat Id
POST	/api/WebUserCreate		felhasználó adatai
POST	/api/StartYoutubeDataOrchestration	username: string, time: int	Időzített lekérés elindult üzenet
POST	/api/StopTracking/string	munkafolyamat azonosító	Időzített lekérés törölve üzenet
DELETE	/api/RemoveTrackedInfluencer	influencerId: int	Influencer törölve a követett influencerek közül
DELETE	/api/DeleteStatistic/int	statisticId: int	Statisztika törölve üzenet

4.1. táblázat: API végpontok és leírásuk az adatok kezeléshez. A GET, POST és DELETE kérések a különböző API funkciókat szolgálják, például adatok lekérését, tárolását és törlését.

4.4. Fejlesztés tervezett menete

A fejlesztés tervezett menetéről az alábbi ábrán látható egy átfogó kép (4.11). Először szeretném az infrastruktúrát kialakítani. Ez alatt arra gondolok, hogy létrehozom és telepítem az Azure App Service és Azure Function App vázát, valamint az Azure SQL adatbázist is létrehozom. Ezután tervezem megtervezni, majd React.js-ben létrehozni a kezelőfelületet. Mint az ábrán is látható, minden funkció létrehozása után tesztelni tervezem az egységeket. A frontend megírása és tesztelése után szeretném a backendhez köthető funkciókat létrehozni és letesztelni. Ez a része a rendszernek Azure Function App lesz, hogy a különböző API-kéréseket jobban szét tudjam bontani, és így akár a későbbiekben a skálázás is egyszerűbbé válhat. Ide fog tartozni az influencer adatainak lekérdezése és adatbázisban való eltárolása. Ezután fogom kiépíteni és összehangolni a frontend és backend közötti kommunikációt. Ezt RESTful API hívásokkal tervezem megtenni, így a frontend API-kéréseken keresztül tudja elérni a keresett influencerek adatait, amelyekből statisztikákat és grafikonokat készít.



4.11. ábra: Fejlesztés általános tervezete.

5. FEJLESZTÉS

5.1. Fejlesztés gyakorlati menete

A fejlesztés menete a gyakorlatban végül merőben másképp nézett ki. A menetrend borulásának fő oka az volt, hogy azokkal a részekkel kezdtem, amelyekről még sokat kellett tanulnom. Ezért például az infrastruktúra előkészitése után, nem a kezelőfelületet hoztam létre, hanem az Azure Functionok-ön kezdtem dolgozni, és ezzel együtt először a backendel jutottam el egy olyan fázisba, ahol már magabiztosan dolgoztam a technológiával. Ezután kezdtem csak el a kezelőfelület kialakitását, és csak utána töltöttem fel Azure Static Web App-ként, amikor már nagyjából működött egyben az egész projekt.

5.2. Infrastruktúra kialakítása

Már az infrastruktúra kialakitásának menete is eltért a tervezettől. Ennek a fő oka az volt, hogy végül a UI-t később kezdtem el fejleszteni, és azt csak akkor akartam az Azure App Service szolgáltatására feltölteni Azure Static Web App-ként amikor már nagyjából kész volt. Ettől függetlenül az Azure Function-ök, és Azure SQL DB létrehozása első lépésként történt meg, bár a fejlesztés folyamán újabb és újabb Azure Function-t hoztam létre a funciók igényeihez igazítva. Az adatbázison csak egyszer kellett változtatnom, mivel az elején beállitott típus a tesztelés folyamán megnövekedett forgalom miatt túllépte az ingyenes szolgáltatás határait, és előre nem látott költséget számolt fel. Ez ugyan meglepetésként ért, de szerencsére volt már hasonló tapasztalatom, ezért bállitottam ellenőrzőpontokat a költség monitorozására, és időben értesítést kaptam. Ez is egy fontos lecke volt a tervezés alaposságának fontosságáról, és arról hogy az ember soha nem lehet elég óvatos, ha a használt felhő szolgáltatás költségei nemelőre rögzitettek.

Kétféle Azure Function-t használtam.Többségében a HTTP Trigger tipust, mivel ezeket lehet HTTP API hivásokkal meghivni, és igy egyszerűen hivhatóak frontendből. A másik tipus a Durable Function volt, amivel az időzitett adatlekérés funkciót biztositom a felhasználónak. Az utóbbi egy fokkal összetetteb felépitésében, és működésében, egyúttal több lehetőséget is biztosít. Fő funkciója, hogy állapotmegőrző munkafolyamatokat lehet létrehozni, és követni serverless környezetben. Ez alkalmasság teszi például olyan funkció implementálására, mint adatok időzített lekérése, dinamikusan beállítottt időközönként.

A fő indok, mai miatt végül Scheduled Azure Function-ök helyett ezt választottam, az pont az, hogy így lehetősége van a felhasználónak, hogy eldöntse a lekérdezések gyakoriságát. 3 fő részre osztható. A HTTP Trigger típushoz hasonlóan ezt is HTTP kéréssel

lehet meghívni. A Kliens Funkció rész fogadja a kérést, és indítja el az agyat, az orkesztrátort. Az orkesztrátor feladata, hogy a munkafolyamatokat meghívja, és vezérelje. Többféle folyamatot is meg lehet határozni a projekten belül, ezek az "Activity Function"-ök. Ezek végzik el a munkát, ami az én esetemben adatok lekérdezése és eltárolása. Mint említettem több Kliens Funkció is létrehozható, ezeket nem muszáj meghívnia az orkesztrátornak, viselkedhetnek szimpa HTTP Trigger Azure Function-ként is. Én az egyes munkafolyamatok leállítására használok egy olyat, ami akkor sül el, ha a felhasználó a kezelőfelületen törli az időzített lekérdezést. Ez ilyenkor a kapott munkafolyamat azonosító alapján leállítja azt, de a többi zavartalanul futhat tovább. Ennél sokkal többet is ki lehet hozni ebből a típusból, de nekem csak ennyire volt szükségem.

5.3. Adatbázis

Egy másik olyan része a tervezésnek, amit sokkal komolyabban fogok venni a jövőben az az adatbázis megtervezése. A tervezésnél ábrázolt adatbázishoz képest lényegesen bonyolultabb lett a végeredmény, és sokat kellett változtatnom rajta fejlesztés közben. A változtatások már ott elkezdődtek, amikor autentikációt implementáltam és emiatt be kellett vezetnem egy új User nevű táblát, amiben a bejelentkezett felhasználókat, és az általuk nyomon követett influencereket tároltam.

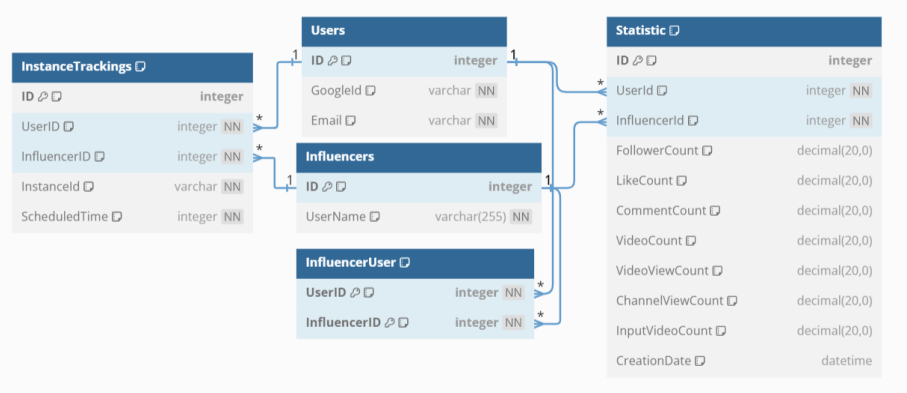
Erre a változtatásra ugyan számitottam, és az implementálása is aránylag simán ment, bár a felhasználókat és influencereket összehangoló JOIN tábla kialakitás kisebb kihívást jelentett. Azonban a későbbiekben újabb problémába ütköztem, ami elkerülhető lett volna, hogy ha legkésőbb ennél a lépésnél alaposan átgondolom, hogy pontosan mire is van szükségem, ahhoz hogy a különböző felhasználókhoz köthető influencereket, és statisztikáikat egymástól jól elkülönithető módon tároljam el. Mivel ezt nem tettem meg, és a statisztikák csak influencerekhez voltak kötve, ezért hamar belefutottam abba, hogy amikor törlök egy statisztikát az A felhasználóhoz tartozó infleuncertől, akkor az a statisztika törlődik a B felhasználó alatt tárolt influencertől is. Ezért bővitettem a statisztika táblát, úgy hogy felhasználóhoz is köthető legyen. Ezen a ponton azt hittem, hogy ami az adatbázist illeti, ugyan nem a legegyenesebb útvonalon, de célba értem. Mint később kiderült ez igencsak távol állt az igazságtól.

A harmadik, vátoztatásra akkor került sor, amikor elkezdtem implementálni az időzített statisztika lekérés funkciót, és az egyes futó munkafolyamatokhoz rendelt azonosítót kellett eltárolnom. Jó ötletnek tűnt, hogy simán csatoljam csak hozzá az influencer táblához, mivel amúgyis egy influencerhez egyszerre csak egy időzitett lekérés köthető. Ez a gondolatmenet jónak is tűnt egészen addig, amíg meg nem próbáltam két különböző felhasználón, ugyanazon az influenceren beállítani a lekérést, és meg nem tapasztaltam, hogy a később beállított felülírja a korábbit mindkét felhasználón. Ez vezetett az adatbázis

utolsó módositásához.

A negyedik módositásra, már fentebb említett körülmények miatt volt szükség. Ahelyett hogy a lekérés azonosítóit influencerekhez kötném, létrehoztam egy InstanceTracking táblát, amiben eltároltam a lekérdezések közötti idő intervallumot, valamint megadtam a felhasználóval, és az influencerrel való kapcsolatát. Ennek a beépítése elég macerás volt, mivel létre kellett hoznom a backend-ben egy új modellt, hozzá egy repository-t, és sok helyen át kellett írnom a kódot, hogy az új adatbázis modellnek megfelelően kérje le, és tárolja el az adatokat.

Mint látható a tervezett adatbázis struktúrához képest sokban változott a végeredmény, és ez esetenként nagyban megnehezítette a fejlesztést, ezért a jövőben kiemelt figyelmet fogok fordítani a tervezési fázisra, és egy olyan adat struktúra kialakítására, amely lefedi az alkalmazás minden igényét.



5.1. ábra: A végleges adatbázis struktúrája.

5.4. Backend

A backend kialakítása nem változott radikálisan a tervezetthez képest, csupán finomodott. Tulajdonképpen Repository tervezési mintát használtam, hogy elválasszam az adataelérési logikát az üzletitől, valamint könnyen megakadályozható a kódismétlés az adat lekérések egy-egy modell köre csoportosításával. Az Azure Function-ök az én esetemben a kontrollereket váltották ki. Az egyes Function-ök csak 1-1 funkciót látnak el, például felhasználó létrehozása, törlése, vagy adatok kinyerése, és eltárolása. Ebből következik hogy a Repository tervezési minta jól illeszkedik hozzájuk, mert így nem kell az egyes gyakorta használt kód részleteket másolgatni, hanem centralizálva interfészeken keresztül, és dependency injection használatával elérhetőek.

Az első kézenfekvő kihívást az jelentette, hogy rendelkezésre álljon egy közös réteg, amit minden Function el tud érni. Ehhez egy Shared Project-et használtam. Ebben hoztam

létre a modelleket, a DbContextet, a közös konfigurációt és a modellekhez tartozó repository interfészeket és repository-kat. A Shared Project azért kínált tökéletes megoldást az én esetemben, mert nem hoz létre önálló DLL-t, hanem az őt felhasználó projekttel együtt fordul le. Ez a viselkedés tökéletesen illik az Azure Function-ök működéséhez, mert így a közös rész, mindegyik projektbe belekerül. Ennek nagyobb projekteknél azzal jár, hogy mindegyik projekt mérete nagyra nő, de az én esetemben ez a probléma még nem állt fent.

Mivel korábban nem dolgoztam még Azure Function-ökkel, ezért olyan funkciókat akartam fejleszteni, amiket később felhasználhatok, és korábban már fejlesztettem őket. Ezért először csináltam a statisztikákhoz és Influencerekhez CRUD műveleteket végrehajtó Function-öket. A későbbiekben ugyan ezek közül direktbe csak egyet használtam fel, de így könnyebben értettem meg a technológia működését, és az ekkor, repositoryban megírt lekéréseket is fel tudtam használni később. Az adatbázis fejezetben említett struktúrabeli változások természetesen minden alkalommal backendbeli változásokat is jelentettek.

A Youtube API-hoz köthető lekérdezések, és Azure Function-ök tervezését, és fejlesztését élveztem a legjobban az egész projekt készítése alatt. Ennek oka az, hogy itt már magabiztosabban dolgoztam az új technológiával, és az API használata is meglepően egyszerűnek biztonyult. A hivatalos dokumentációból könnyen ki tudtam deríteni pontosan milyen statisztikákat kell a csatornáról, és a videókból kinyernem. A legnagyobb kihívást itt az jelentette, hogy egyszerre csak 50 videót azonosítót tudott visszaadni az API, ezért az azokat kinyerő metódust úgy kellett megírnom, hogy dinamikusan kinyerjen legfeljebb 500 videó azonosítót. Ezt úgy oldottam meg, hogy a felhasználó által kért videószámra kiszámolja, hogy hány 50-es blokkra lesz szükség, és addig kér újabb, és újabb 50 azonosítót, amíg el nem éri ezt a számot, vagy a csatornán maximum feltöltött videók számát.

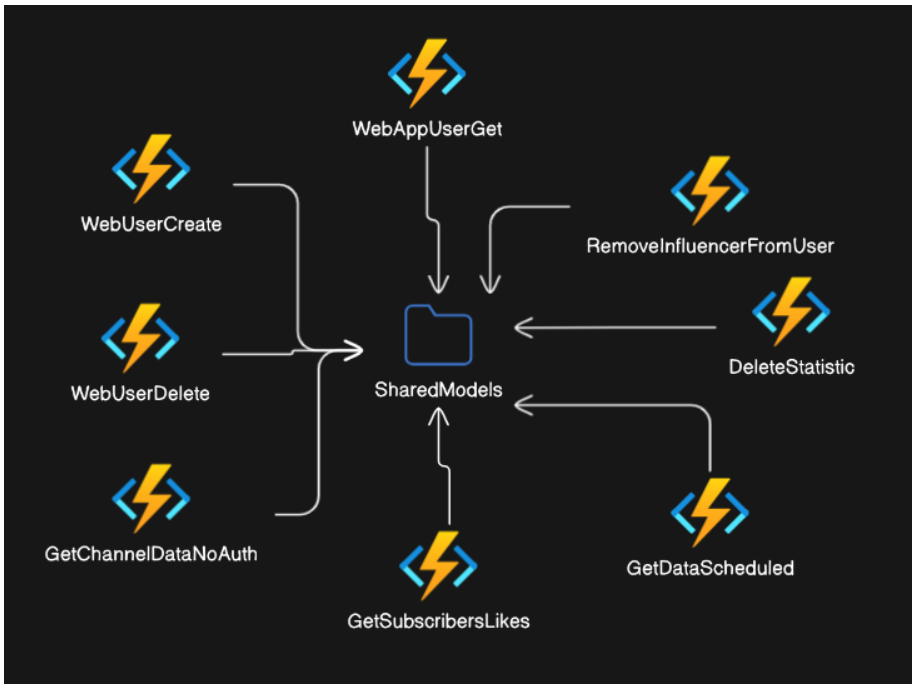
Az autentikáció implementálása nem volt különösebben nehéz. Pár kivétellel mindegyik HTTP kérés továbbít egy Bearer token-t, amit egy Interceptor told hozzájuk. Ezt validáltam, majd kinyertem a GoogleId-t, és email címet. A hívásokból még olyan információkat is kinyerhettem mint a keresett csatorna neve vagy, lekért videószám. Igyekeztem ugyan minél inkább centralizálni az autentikációs kódot, mert minden Azure Function elején validálni kell a kérést. Sajnos ez nem sikerült teljes mértékben, de így is kevés kódot kellett csak másolnom.

Aránylag nagy kihívás volt, amikor autentikáció implementálása után bevezettem a felhasználó modellt, és ennek a tábláját össze kellett kötnöm az influenszerek táblájával egy JOIN tábla segítségével. Ezen a ponton még manuálisan SQL-el hoztam létre az adatbázis tábláit. Ez is teljesen felesleges lépés volt, mert egyszerűen összeköthettem volna az Azure SQL adatbázist a Visual Studioval. Ezt a későbbiekben meg is tettem, és így a már kisebb problémát jelentett, ha újabb modelt kellett bevezetnem. A már fentebb, az

adatbázis fejezetben említettem többször is ki kellett bővítenem az adatbázis tábláit, és ezek a változások kisebb nagyobb mértékben a backend kód változását is magukkal vonzották. A legtöbb változást az InstanceTracking tábla bevezetése hozta, ez egy új modellt, repository-t is magával vont, és sok helyen át kellett írnom a már megírt lekérdezéseket. Utólag sokkal logikusabbnak tűnik ez a megoldás, és jó lecke volt, ha nem is kellemes.

A legnagyobb akadályt az időzített lekérés funkció implementálása jelentette. Ehhez Azure Durable Function-t használtam, és ennek a működése valamivel bonyolultabb mint a másik típusé. Miután megértettem a működését, már egészen gyorsan haladtam vele, mivel sok dolgot át tudtam venni korábbi kódjaimból. Akkor ütköztem falba, amikor az időzített lekérdezés már rendben működött, azonban ennek a dinamikus leállítása sehogy sem akart összejönni. Az nyilvánvaló volt, hogy az egyes lekérdezések között nem futhat aktívan a Function mivel ez nagyon megdobná a fogyasztást, és egy-egy lekérdezés között akár 1 hét is eltelhet. Elsőnek aszinkron megoldásokkal próbálkoztam, de sajnos ezek nyitva tartották az adatbázissal való kapcsolatot, ezért az egyfolytában kifutott az időből, és megszakította a kapcsolatot. Végül kiderült, hogy egy olyan problémát próbálok megoldani, ami tulajdonképpen meg van oldva beépítve, mivel az orkesztrátor egyfolytában figyeli az adott elindított folyamat példány állapotát, és ha ezt "Terminated"-re állítom, akkor azonnal megszünteti azt.

Összességében a projekt készítésének ezt a fejezetét élveztem a legjobban, mert rengeteget tanultam belőle, és teljesen megváltoztatta azt ahogyan egy ilyen összetett projekt tervezéséhez hozzákezdenék. Az Azure Function-ökkel való munka is érdekes volt, és örülök, hogy megismertem ezt a technológiát. Végső soron a "cold start" jelenség se volt vészes egy-egy hosszabb szünet utáni indítás esetén, de természetesen megfigyelhető ilyenkor egy kis késleltetés. Szerencsére ez az én esetemben sohase volt több 1,5-2 másodpercnél, ami sokkal jobban hangzik mint a kutatómunkám alatt talált információk, amik akár 10-15 másodpercről is írtak. Ez valószínűleg annak köszönhető, hogy egy kis projektről beszélünk, és a lekérdezések sem bonyolultak, 1-1 Funtion gyorsan fel tud ébredni.

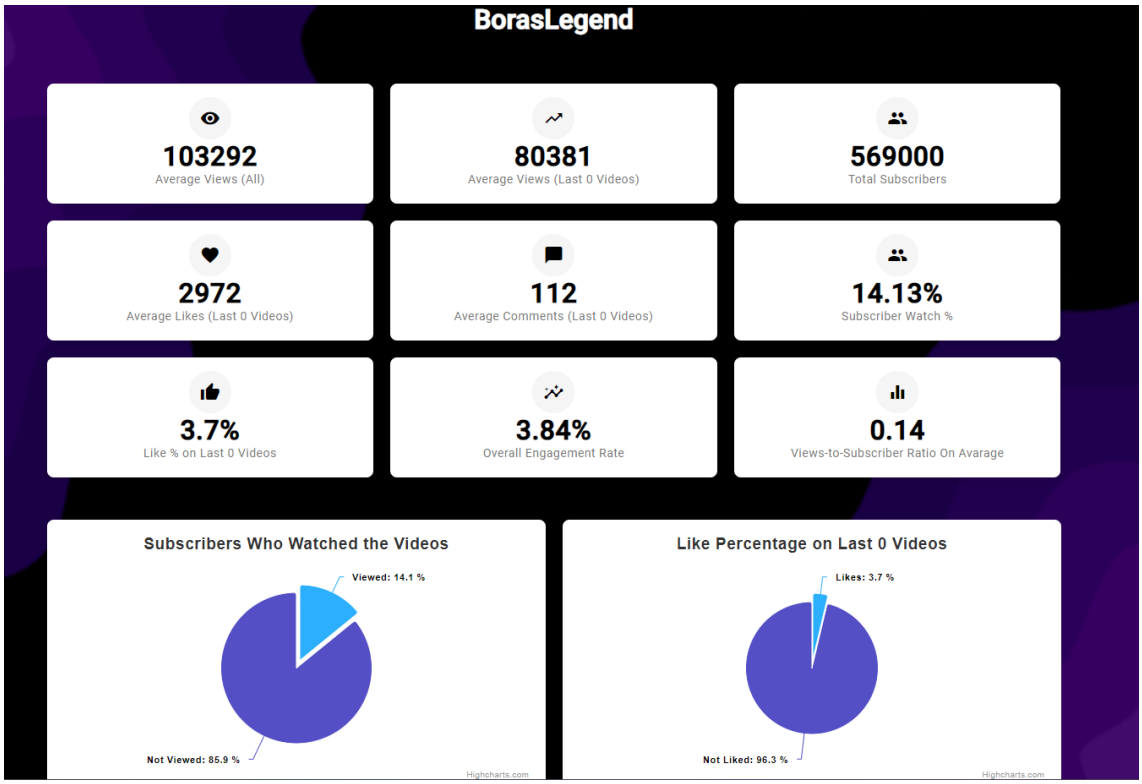


5.2. ábra: Közös Shared Project és Azure Function-ük kapcsolata.

5.5. Frontend

A frontend kialakításához végül Angular keretrendszert használtam React helyett, mivel az előző, tavaszi szemeszterben Angular-t használtam egy projekt megvalósításához, így rendelkeztem némi gyakorlati tapasztalattal. Sajnos azonban azóta sokat felejtettem, így a frontend fejlesztése is kihívásokkal teli volt. Végül a Standalone komponenseket használtam a klasszikus moduloktól függő megközelítés helyett. Azért döntöttem így, mert kíváncsi voltam, hogy hogy is működik gyakorlatban, valamint úgy éreztem, hogy tisztább, és könnyebben újrahasználható kódot kapok ezen a módon.

Az első mérföldkő, amit el akartam érni a kezelőfelület fejlesztésében, az volt, hogy legyen lehetőség egy Youtube csatornára rákeresni, majd az onnan kinyert statisztikákat ábrázolni, és ezekből grafikonokat gyártani. Ezen a ponton az adatkinyerés funkciót ellátó Azure Function már készen állt, és Postman-ben tesztelve a működése is megfelelő volt, így már csak a kezelőfelület hiányzott mellőle. A kereső rész implementálása egészen könnyű volt Angular Material elemeket használva, és az adatok igényes megjelenítése is meglepően simán ment, köszönhetően a könnyen használható grafikon készítő könyvtárnak. A legnagyobb kihívást ezen a ponton, amellett hogy kiszámítsam a visszakapott nyers adatokból a különböző átlagokat, és százalékokat, az jelentette, hogy a statisztikákat, és grafikonokat tartalmazó csempéket valamilyen igényesnek mondható módon, lehetőleg középre rendezzem, egymástól egyenlő távolságra.



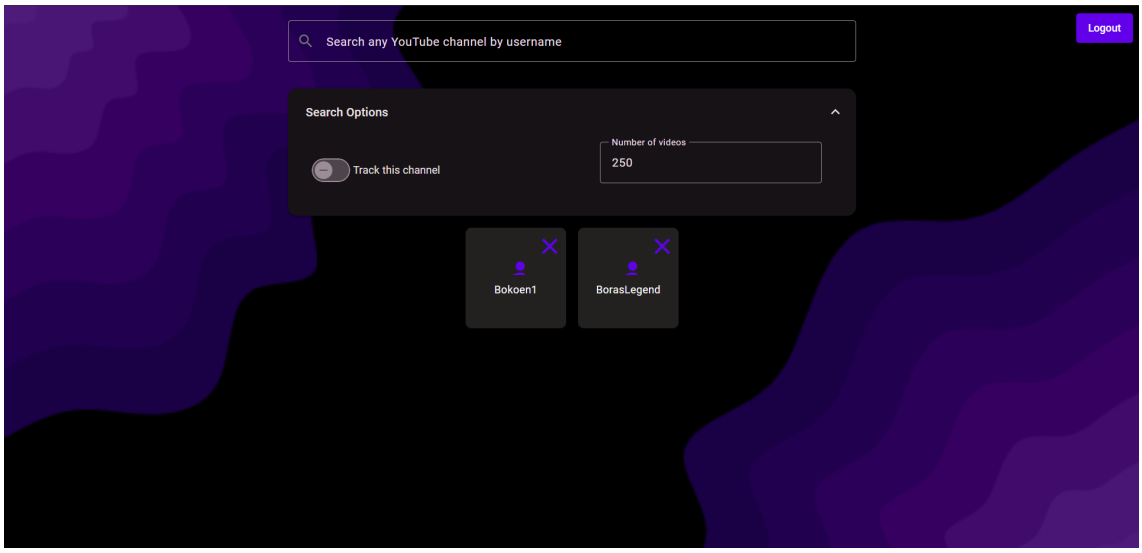
5.3. ábra: Kiszámított, és ábrázolt statisztikák, és grafikonok BorasLegend nevű Youtube csatorna keresése után.

A következő lényegesen nagyobb kihívást jelentő akadály az autentikáció, és autorizáció implementálása volt. Kutatómunkám eredményeként úgy döntöttem, hogy mivel amúgyis Google API-t használok a Youtube adatok kinyeréséhez, és gyakorlatilag mindenkinek van Google fiókja, ezért Google autentikációt fogok használni. Semmiképpen se akartam magamtól valami ilyesmit összerakni, mert a végeredmény biztosan rosszabb, és legfőképp sérülékenyebb lett volna. Ennek az implementálása is jó ütemben haladt, köszönhetően a korrekt dokumentációnak, és átlátható felhasználóbarát felhasználófelületnek. Az Api kulcsok igénylése mind a Youtube Data API, mind a Google OAuth2 esetén roppant egyszerű volt, és könyedén lehetett szabályozni, hogy milyen weboldalakról fogadjon kéréseket, és milye Google API kérések kötődhetnek az adott tokenhez ezzel is növelve a biztonságot. Az egyetlen hátránya, hogy napi 10000 kéréseket enged csak, ezt azért sikerült párszor lemerítenem, amikor már nem kézzel teszteltem, hanem több megírt tesztel.

Tehát gyakorlatban végül Google OAuth2 alapú autentikációt használtam. Ez lekéri a security tokent, amit utána local storage-ban tároltam el. Azért döntöttem emellett, mert ugyan sérülékeny lehet(például XSS támadással szemben), de ezek a sérülékenységek könnyen kivédhetőek, ha a különböző bemenetek rendesen korlátozva vannak. Valamint a token lejárásával együtt kijelentkezteti a felhasználót, ekkor visszavonja a tokent, és

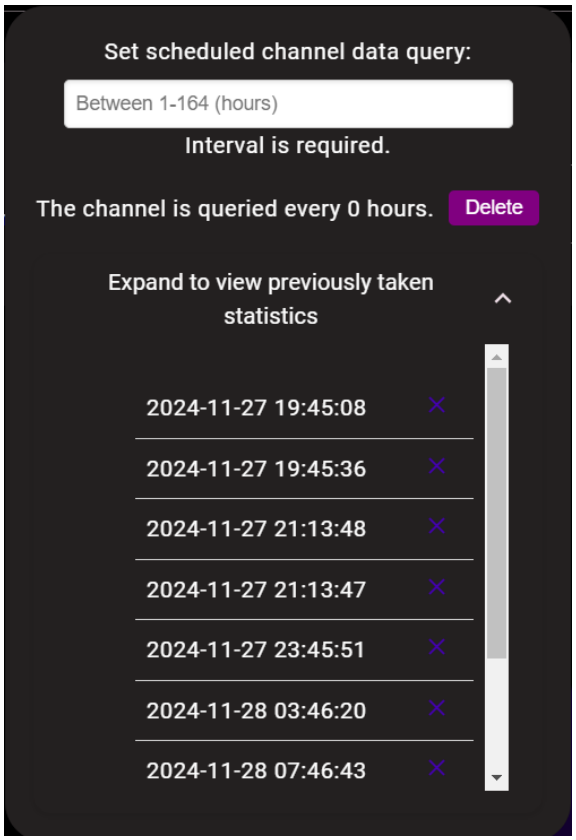
törli azt a local storage-ból. Cserébe nagyon egyszerű az implementálása. Ezen a ponton még csak a frontend rendelkezik a tokennel, amiben olyan információk vannak mint a felhasználó email címe, vagy egyedi Google azonosító, de ezt valahogy el kell juttatni a backand-be, hogy az egyes Azure Function-ok is felhasználókhoz kötve fussanak le, és validálni tudják a kapott tokent. Ezt egy interceptor-al oldottam meg, aminek az a lényege, hogy különleges esetek kivételével minden HTTP híváshoz egy Bearer token-t kapcsol, amit minden egyes Azure Function feldolgoz, és hitelesít. Ez azért is jött kapóra, mert így a Function-ök is sokkal biztonságosabbá váltak. Az autentikáció backend-ben implementált részéről a Backend fejezetben írok bővebben.

A következő kihívást az jelentette, hogy ne csak lekérni tudja a felhasználó a Youtube csatornák adatait, hanem ezeket a statisztikákat el is tudja tárolni, és a későbbiekben megtekinteni. Ehhez muszáj volt megjelenítenem a követett csatornákat, és biztosítanom valamiféle funkciót, hogy a felhasználó válogathasson a korábban lekért statisztikák közül. Itt a legnagyobb kihívást az jelentette, hogy a megjelenített, követett csatornák szinkronban legyenek az adatbázisban eltárolt, és felhasználóhoz kötöttekkel. Eleinte ezt session storage-al próbáltam megoldani, azonban ez nem volt konzisztens, és rosszul kezelte a kereső oldal és a statisztikákat ábrázoló oldal közötti váltásokat. Ezért bevezettem egy külön felhasználó service-t, ami egységesítette a követett felhasználók-hoz kötött műveleteket, mint a session storage-hoz való hozzáadást, és az adatok onnan való lekérdezését. Ezeket egy "trackedInfluencersSubject" nevű objektumon hajtotta végre, amire a többi komponensből fel lehet iratkozni, így elérve, hogy ha változik az értéke, akkor az visszaköszönjön a többi komponensben is. Még így sem tudtam elérni az elvárt működést, mert a ez ugyan a frontenden belül frissen tartotta az objektumot, de sok esetben elavultá vált, ha nem jött újabb adat a backendből. Ezért csináltam egy dedikált metódust, ami meghívja a felhasználó létrehozására is használt Azure Function-t, ami a felhasználó létezése esetén visszaadja a felhasználó adatait az adatbázisból, majd ez a válasz alapján frissíti a service a követett csatornák objektumot. Mint látható, ahhoz hogy elérjem a kívánt hatást, és a csatornák megjelenjenek oldal újratöltés, oldal váltás, és kijelentkezés után is mindháromra szükségem volt, és hosszas tesztelés, próbálkozás, és kutatás után értem csak el a kívánt eredményt.



5.4. ábra: A fő oldal bejelentkezés után, itt láthatóak a követett felhasználók, és itt lehet keresni, valamint beállítani annak paramétereit.

A fentebb ecsetelt folyamat alatt természetesen mással is foglalkoztam, például az ütemezett statisztika lekérés funkció implementálásával, ez a frontend esetén inkább csak megemlítem, mert ezt a részét könnyű volt megcsinálni, és inkább csak azért említem meg külön, mert itt kezdtem el korlátozásokat bevezetni a különböző bemeneteken. Mivel ez a funkció azt biztosítja, hogy a felhasználó szabadon beállíthat egy idő intervallumot 1 és 164 óra között, ami megszabja, hogy milyen időközönként fusson le a lekérdezés. Amennyiben a beírt érték nagyobb vagy kisebb ennél, egy erre figyelő metódus a megengedett minimum, vagy maximum értékre javítja azt. Tört számok esetén kerekít a rendszer, erre azért volt szükség, hogy a backend Azure Function-ben már ne kelljen ezt lekezelni.



5.5. ábra: Egy követett felhasználóra kattintás után ezt látja a felhasználó. Láthatóak a korábbi statisztikák, és itt indítható, és leállítható az ütemezett adat lekérés.

Az utolsó nagyobb változás, akkor következett be, amikor bevezettem egy olyan mezőt, ahol megadhatja a felhasználó, hogy hány videó adatait szeretné lekérni. Itt a maximum az 500 videó, az alap érték pedig 250, amennyiben a csatornán kevesebb videó van, mint a lekért, visszaadja az összeset probléma nélkül. Ebben a fázisban biztosítottam opciót arra is, hogy bejelentkezés nélkül is rá lehessen keresni csatornákra, azonban ilyenkor az adatbázist nem érinti a folyamat, és a csatornák követésére sincsen opció. Ilyenkor az interceptor nem csatol token-t a HTTP híváshoz, és a hívás is egy új Azure Function-t hív. Azért, hogy ne kelljen telibe lemásolnom a már meglévő kereső metódust, létrehoztam egy segéd metódust, ami megnézi hogy a keresés pillanatában be van e jelentkezve a felhasználó és ez alapján adja vissza a megfelelő Azure Function URL-jét a kereső metódusnak.

6. TESZTELÉS

6.1. Bevezetés

A tesztelés nyilvánvalóan egy nagyon fontos része a fejlesztésnek, és a munkafolyamat során természetes módon mindenképp tesztelünk, amikor vizsgáljuk, hogy egyes implementált funkciók úgy működnek-e, ahogyan azt szeretnénk. De persze vannak olyan funkciók, amelyeket nem lehet jól letesztelni kézzel, mert túl sok időt venne igénybe az összes variáció kipróbálása, vagy csak kényelmetlen hozzáférni más funkciók érintése nélkül. Ezekre a problémákra is megoldást kínálnak az olyan különböző tesztelési típusok, mint az Unit tesztek, az End-to-End tesztek, az integrációs tesztek, vagy éppen az API tesztek. Ezek mindegyike különböző módon közelíti meg a tesztelést és a tesztelendő funkciót/funkciókat, de erről bővebben a következő szekcióban írok.

Ezeknek a tesztelési megoldásoknak természetesen az a céljuk, hogy a terméket jobbá, annak működésének vizsgálatát pedig könnyebbé tegyék. Ezenkívül a fejlesztési folyamatot is gördülékenyebbé tehetik, és megmenthetik a fejlesztőt rengeteg manuális teszteléstől. Például, ha korábban elkezdek E2E teszteket futtatni, amelyek felhasználói interakciót hivatottak szimulálni az én esetemben, rengeteg manuális kattintgatástól menthettem volna meg magamat. Az okom arra, hogy nem implementáltam előbb a tesztelést, leginkább annak tudható be, hogy először a többi fő funkcióval akartam elkészülni, és a teszteléssel nem volt nagy gyakorlatom, ezért nem tudtam megbecsülni, mennyi időt fog felemészteni. De ez is felkerül az egyre népesebb listára, amely azokat a dolgokat tartalmazza, amelyeket feltétlenül másképp csinálnék, ha újrakezdeném az egész projektet.

6.2. Tesztelési típusok

Az első teszttípus, amellyel foglalkozni szeretnék, az a Unit teszt. Ennek célja a kisebb kódrészletek, mint például metódusok tesztelése izolált környezetben. Ez azt jelenti, hogy a valós függőségeit áll függőségekkel helyettesítjük. Ilyen lehet például egy adatbázis vagy API-hívások. Erre azért van szükség, hogy kontrollált környezetben vizsgálhassuk a kódrészlet működését, és más egységek hibái ne befolyásolják az eredményt. Végül nem használtam ezt a típust, mert az E2E tesztek, az integrációs tesztek és az API-tesztek mellett feleslegesnek tűnt a projekt végén megírni őket. Ha újrakezdeném, akkor írnék Unit teszteket a repository-k ellenőrzésére; összességében valószínűleg sok időt spórolhattam volna velük.

Az API tesztek feladata, hogy a rendszer API-jainak működését teszteljék. Ez nálam azt jelenti, hogy megnézik, hogy helyes HTTP-kérésekre az elvárt választ adják vissza,

legyen az státuszkód vagy valamilyen adat. Ezenkívül készítek negatív teszteket is, amelyek célja, hogy kiderüljön, autentikációs token hiánya vagy hiányzó paraméter esetén a rendszer megfelelő státuszkódot és hibaleírást ad vissza. Tehát az én esetemben azt vizsgálják, hogy adott Azure Function HTTP-hívása esetén a rendszer megfelelő választ ad-e vissza.

Az integrációs tesztek feladata, hogy az alkalmazás különböző moduljainak együttes működését teszteljék. Az én esetemben azt vizsgálják, hogy az adatok YouTube-ról való lekérdezését és adatbázisba való eltárolását végző Azure Function megfelelően működik-e. Tehát azt vizsgálják, hogy az adatok lekérését és azok eltárolását végző modulok szinkronban vannak-e, és hozzáférnek-e a külső rendszerekhez, mint az Azure SQL DB és a YouTube API. Látható, hogy a Unit tesztekhez képest itt nem az egyes komponensek izolált tesztelése a cél, hanem annak vizsgálata, hogy ezek hogyan dolgoznak együtt, jól kezelik-e az adatáramlást, és képesek-e kapcsolatot teremteni, valamint fenntartani a külső rendszerekkel.

Az End-to-End tesztek tűnnek nekem a legátfogóbb teszt típusnak, hiszen ennek célja, hogy az applikációban egy teljes folyamatot végigcsináljon, a felhasználó lépéseit szimulálva. Emiatt sok esetben minden komponenst érintenek, legyen az API-hívás, adatbázisból való lekérdezés vagy oda való eltárolás, és megmutatják, hogy ezek a komponensek jól kommunikálnak-e egymással. Ezenkívül ez a teszt ellenőrzi azt is, hogy a különböző UI-elemek jókor, jó helyen jelennek-e meg, valamint hogy az adatmezők kitölthetők-e, és a visszakapott adatok láthatók-e a felhasználó számára. Én is alkalmazok ilyen teszteket, hogy a főbb funkciók munkafolyamatát leellenőrizzék felhasználói oldalról. Az én esetemben ez azt jelenti például, hogy bejelentkezik egy dedikáltan tesztelésre létrehozott Google-fiókkal, rákeres egy YouTube-csatornára, és ellenőrzi, hogy létrejönnek-e a statisztikákat ábrázoló csempék. Más tesztek nem állnak meg itt, hanem ellenőrzik, hogy egy bekövetett csatorna megjelenik-e a neki fenntartott helyen, vagy hogy elindítható-e az adott csatornán egy időzített lekérdezés. Mivel összesen 8 ilyen teszt van, és az első lépések sokszor ugyanazok, ezért vannak olyan ellenőrzések, amelyek csak ritkábban történnek meg, például a statisztikacsempék ellenőrzése minden esetben felesleges lenne. Vannak persze kritikus dolgok, mint az adatbeviteli komponensek és a különböző gombok megjelenítése, amelyeket minden teszt ellenőriz.

6.3. Playwright

A Playwright a Microsoft kreálmánya, és egy nyílt forráskódú eszköz, amelyet webalkalmazások tesztelésére és böngészőautomatizálásra terveztek. Azért választottam ezt például a Selenium helyett, mert frissebb technológia, és a tesztguru kollégám is ezt ajánlotta. Ugyan vannak funkciói, amelyek hasznossá teszik más teszttípusokhoz is, én kifejezetten

E2E teszteléshez használtam. Tetszett, hogy könnyen hozzáférhettem a local storage-ban tárolt tokenhez, így azt könnyen tudtam validálni, valamint ki tudtam nyerni és eltárolni későbbi használatra. Egy másik nagyon hasznos funkciója, hogy bevárja, hogy a keresett elem vagy szöveg megjelenjen. Bár eleinte belefutottam pár esetbe, amikor emiatt a várakozás miatt kifutott az időkorlátból, mert nem jelent meg az elem, ezt szerencsére ki lehet kerülni, ha finomhangoljuk a késleltetést. Fontos funkció még, hogy könnyedén futtathatja a teszteket párhuzamosan több böngészőablakban. Ezzel próbálkoztam is, de az E2E teszteknél nem működött igazán jól, mert túl sok erőforrást emésztettek fel, és kifutottak az időből, mielőtt végigcsinálták volna a lépéseiket. Biztos vagyok benne, hogy meg tudnám oldani, de most sajnos nem tudtam ráfordítani kellő energiát. Hasznos funkciója még, hogy ha egy teszt elakad valahol valamilyen hiba hatására, akkor képernyőképet készít, és így könnyebben beazonosítható a hiba forrása.

6.4. Tesztelés menete

Mint mindig, a kezdet itt is nehéz volt. Ez betudható annak is, hogy nincs még nagy gyakorlatom tesztek írásában, és annak is, hogy új módszerekkel és technológiákkal kellett megismerkednem. Első mérföldkőnek azt jelöltem meg, hogy legyen két E2E tesztem, amelyek a be- és kijelentkezés folyamatát tesztelik. Azért ezzel akartam kezdeni, mert mindenképp a valós Google Authentication tokent akartam használni az API- és integrációs tesztekben. Ez a része még elég könnyen és gyorsan ment. Először frontendben feltageltem a különböző HTML-komponenseket, és ezeket könnyedén beazonosította és használta a Playwright. Validálás során megvizsgálta a teszt, hogy létezik-e a local storage-ban az Authentication token, és ha igen, akkor lementette azt egy környezeti változóba, ami az adott futás ideje alatt elérhető. Később ezt kimentettem egy JSON-fájlba is, hogy futási időn kívül is elérhető legyen más tesztek számára.

Miután ezzel megvoltam, elkezdtem megírni a többi E2E tesztet, és itt kezdődtek a problémák, mert ezek a tesztek már jobban belenyúltak az adatbázisba. Itt meg is jegyezném, hogy az E2E tesztek írása alatt végig a production adatbázist használták, ami tudom, hogy hiba, de ezen a ponton nem találtam még jó módszert arra, hogy az Azure Function-ök dinamikusan tudjanak adatbázisok között váltani. Úgy éreztem, megöli az E2E célját, ha a valóstól eltérő környezetet hozok létre, a teszt írással pedig haladnom kellett. Azt, hogy ezt hogyan oldottam meg végül, a következő bekezdésben fogom leírni. Ezenkívül ezeknél a teszttípusoknál nem volt nagyobb problémám, csak pepecsmunkát jelentett, amíg megcsináltam a sok helper osztályt, amelyek a különböző validálásokat és az oldalon való navigációt végezték. De legalább a tesztek elég letisztultak lettek így.

Következő lépésként elkezdtem az API tesztekkel foglalkozni, és itt térnék vissza az adatbázis kérdésére. Ezen a ponton már elkerülhetetlennek éreztem, hogy ezek a tesztek

egy olyan adatbázist használjanak, amely rendelkezik minden tesztnél egységesen ugyanazon elemekkel. Ekkor sikerült megoldanom, hogy az Azure Function-ök ne csak egy adatbázisra tudjanak csatlakozni, hanem amennyiben létezik egy Enviromental Variable, akkor annak a kulcsnak az értékét állítsák be Connection String-nek. Ezen az eredményen felbátorodva létre is hoztam egy-egy adatbázis osztályt az E2E és API tesztekhez, amelyek egy közös ős absztrakt osztályon osztoztak. Ez az ősosztály létrehozta magát az adatbázis kontextust, de azt már a leszármazott alosztályok felülíró metódusai döntötték el, hogy mivel töltik azt fel. Az E2E tesztek esetén itt csak teljesen megtisztította az osztály a táblákat, API teszteknél pedig feltöltötte azokat alapadatokkal. Fontos volt, hogy ne törölje magukat a táblákat vagy az adatbázist két teszt setup futása között, mert ez egy a production adatbázis struktúrájával teljesen megegyező Azure SQL adatbázis volt. Ezt egyfolytában törölni, létrehozni és táblákkal feltölteni költséges lett volna pénzben és időben. Valószínűleg lenne jobb megoldás a tesztadatbázisra, de tudtam, hogy ez működik az Azure Function-ökkel, ezért ennél maradtam.

```
[SetUp]
0 references
public async Task SetUp()
{
    _e2eTestDb = new E2E_Test_DB();
    _dbContext = _e2eTestDb.CreateTestDbContext();

    _playwright = await Playwright.CreateAsync();
    _browser = await _playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
    {
        Headless = false,
        //needed to convince google it's not an automated test
        Args = new[]...
    });

    Environment.SetEnvironmentVariable("TEST_DB_CONNECTION_STRING", ConfigurationHelper.connectionStringTestDb);

    var context = await _browser.NewContextAsync(new BrowserNewContextOptions...);
    //helps bypass bot detection
    await context.AddInitScriptAsync("Object.defineProperty(navigator, 'webdriver', { get: () => undefined })");
}
```

6.1. ábra: Az E2E teszt Setup. Minden teszt indulása előtt lefut, és beállítja a használandó böngészőt, és adatbázist. Itt állítja be az "Enviromental Variable"-t is.

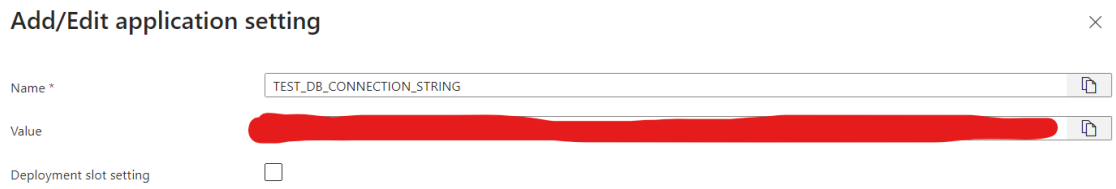
```
var host = new HostBuilder()
    .ConfigureFunctionsWebApplication()
    .ConfigureServices(services =>
    {
        services.AddApplicationInsightsTelemetryWorkerService();
        services.ConfigureFunctionsApplicationInsights();

        var connectionString = Environment.GetEnvironmentVariable("TEST_DB_CONNECTION_STRING") ?? ConfigurationHelper.connectionString;
        services.AddDbContext<AppDbContext>(options =>
                    options.UseSqlServer(connectionString));

        services.AddScoped<IUserRepository, UserRepository>();
        services.AddScoped<IUserAuthenticationRepository, UserAuthenticationRepository>();
    })
    .Build();

host.Run();
```

6.2. ábra: Ezt minden Azure Function Program.cs-ben be kellett állítani. Ha nincs beállítva az adott változó, az alap "Connection String"-et használja.

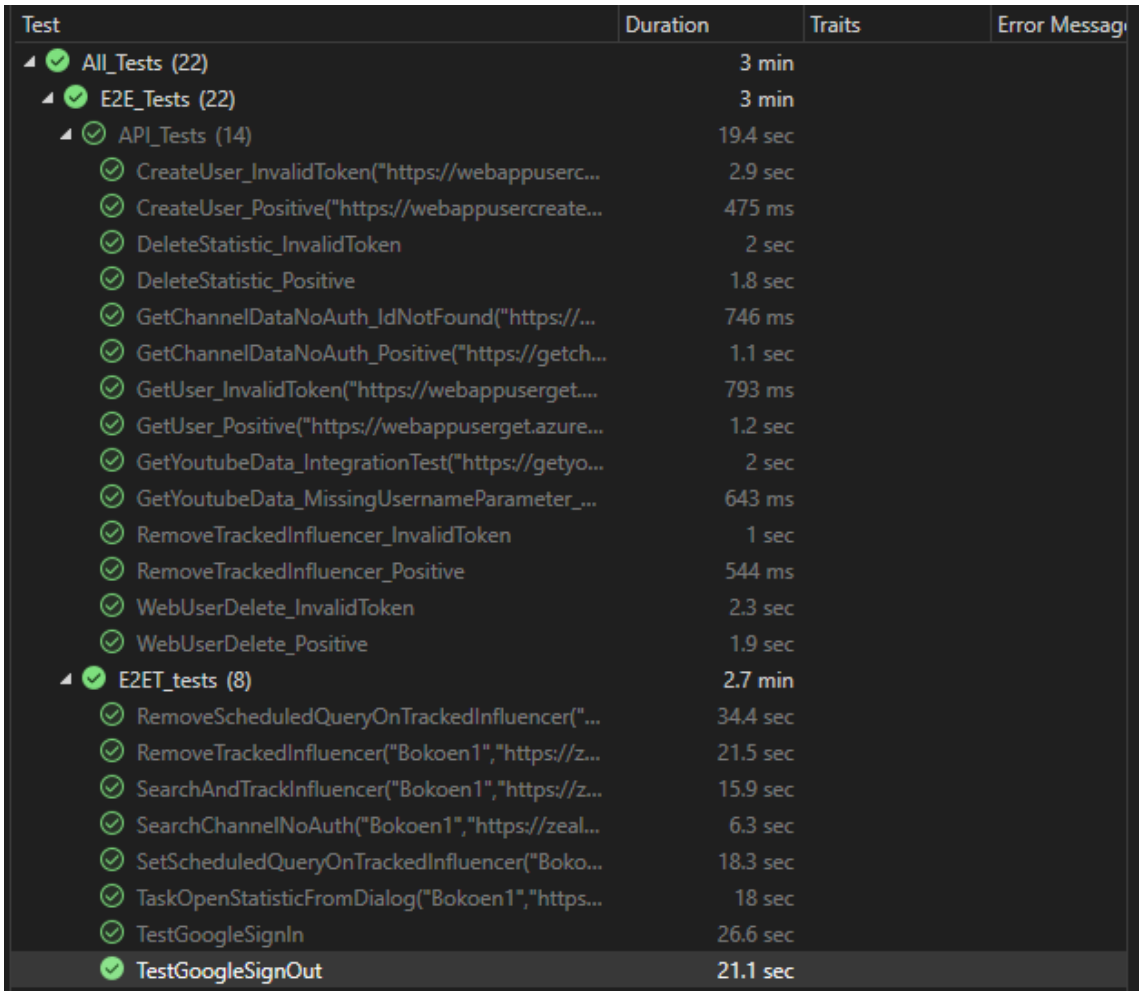


6.3. ábra: Ezt minden Azure Function beállításaiben be kellett állítani.

Miután végre kialakult a teszti nfrastruktúra, már aránylag gyorsan ment az API- és az integrációs tesztek kialakítása. Itt igazából az volt a célom, hogy a főbb funkciókat leteszteljem helyes bemenet esetén, és leteszteljek néhány esetet hibás autentikációs tokennel vagy hibás bemenettel is. Csak egy integrációs tesztet készítettem, amely azt vizsgálja, hogy a YouTube API-ról lekérdezett és eltárolt adatok megegyeznek-e a felhasználónak válaszként visszaadott adatokkal. Miután minden teszt lefordult helyesen, elkezdtem olyan plusz dolgokkal foglalkozni, mint a tesztek futásának párhuzamosítása vagy a tesztek GitHub Action-nel való automatizálása. A párhuzamosítással foglalkoztam is egy keveset, de hamar beláttam, hogy ahhoz, hogy jól megcsináljam, jobban át kéne gondolnom a dolgot, és nem kapkodva megcsinálni. Ezt a jövőben meg is szeretném valósítani, mert sokkal elegánsabb lenne, és jelentősen gyorsítaná a tesztelés menetét. Ugyanerre a sorsra jutott a GitHub Action-nel kapcsolatos törekvésem is. Itt is a szükséges tudás megszerzéséhez szükséges idő hiánya miatt hagytam abba. A későbbiekben azonban ezt is szeretném megvalósítani, mert érdekesnek tartom.

6.5. Összefoglalás

Nagyon tanulságos volt a tesztelés folyamata, és a jövőben figyelni fogok arra, hogy ha olyan projekten dolgozom, ahol segíthet, akkor hamarabb és rendeltetésüknek megfelelő időben kezdjem el a teszteket implementálni. Itt is maradtak olyan funkciók, amelyeket még szívesen beépítettem volna, de már nem maradt rájuk időm vagy kapacitásom. Ezek ugyan nem szerepeltek az előre meghatározott követelmények között, de a későbbiekben igyekszem beépíteni őket.



6.4. ábra: A lefutott tesztek eredményei, és futási idejük.

7. ÉRTÉKELÉS

A végtermékkel összességében elégedett vagyok, mivel sikerült elérnem az összes előre kitűzött célt. Sikerült egy olyan felhő infrastruktúrát létrehozni, amely kielégíti a meghatározott elvárásokat. Ez alatt azt értem, hogy az alkalmazás Azure Static Web App-ként 0-24 a felhasználók rendelkezésére áll, az adatok egy Azure SQL adatbázisban vannak eltárolva, és a háttérmunkát Azure Function-ök végzik. Az Azure Function-ök működésével különösen meg vagyok elégedve, és úgy érzem, hogy a közös Shared Project-es megvalósítás kifejezetten jól működik. Ennek ellenére persze lehetne még rajtuk javítani, például a hibakezelés terén, de a feladatukat teljes mértékben ellátják. Viszont az Azure SQL adatbázisról a jövőben szívesen váltanék valamilyen más megoldásra, mert nem vagyok maradéktalanul elégedett az árával.

Ezenkívül úgy gondolom, hogy az irodalomkutatás során megvizsgált rendszerekből levont tanulságokat is jól sikerült felhasználni, és egy letisztult, átlátható kezelőfelületet tudtam létrehozni. Ehhez azért még hozzáadnék pár dolgot, például a követett csatornák profilképe látszódhatna a csempéiken, és a bejelentkezett felhasználó Google-profilképére is igaz ugyanez. Amit még nagyon szívesen látnék, az az, hogy ha a felhasználó elkezd beírni egy YouTube-csatorna nevet, akkor választhasson a lehetőségek közül, mert jelenleg a csatornán található, @ kezdetű azonosító alapján találja meg csak pontosan a csatornákat. Mindenképpen csiszolnék még az alkalmazás hibakezelésén, mert az jelenlegi állapotában hagy maga után kívánnivalót.

Teljes mértékben elégedett vagyok a funkciókkal, mert ugyan az eredetileg tervezettel ellentétben nem Instagram-influencerek profiljairól nyer ki adatokat és ábrázol, hanem YouTube-csatornákról, ez annyi különbséget jelent, hogy nem lájkokat és követőszámot vizsgál, hanem lájkokat és feliratkozószámot. A lekért adatokat jól eltárolja, és igényes formátumban ábrázolja. Indítható időzített lekérés, és a felhasználó megtekintheti régebben lekért adatokat, valamint követhet több YouTube-csatornát is. Az adatok kinyerése és ábrázolása meglepően gyors, amikor nem kell hidegindítással számolni, de még olyankor is gyorsabb, mint egyes rendszerek, amelyeket vizsgáltam. Ez azonban lehet egyszerűen az Instagramról és a YouTube-ról való adatkinyerés komplexitásának különbsége. Jó lett volna készíteni egy olyan kimutatást, amely összehasonlít két különböző időpontban lekért statisztikát, de ez nem volt elvárás, és sajnos nem maradt rá idő.

A tesztek megírása nehezebben ment, mint vártam, de a végeredménnyel összességében elégedett vagyok. A tesztek lefedik a működés legnagyobb részét, és a főbb elemek külön is meg vannak vizsgálva. Az E2E tesztek megírása vett el jelentős időt, mert az oldalon való navigációhoz rengeteg segédmetódust írtam. Ezenkívül nagy kihívás volt megtalálni a megfelelő megoldást arra, hogy az Azure Function-ök ne a production adat-

bázist használják teszteléskor, hanem annak valamilyen replikáját. Ehhez is szívesen hozzáadnék extra funkciókat és további teszteket. Mindenképp ilyen funkció lenne, hogy a tesztek automatikusan lefussanak bizonyos Git-eseményekkor, ezzel is könnyítve a tesztelési folyamatot. A másik fontos fejlesztési pont a tesztek futásának párhuzamosítása, amely a tesztelés kezdetekor nem volt még szempont, de a végén már szívesen láttam volna.

Itt rá is térnék azokra a dolgokra, amelyeket másképp csinálnék, ha elölről kezdeném a projektet. Sokkal nagyobb figyelmet fordítanék a modellek és adatbázistáblák megtervezésére, mert rengeteget küzdöttem azzal, hogy utólag kellett változtatnom az adatbázis struktúráján. Ez többször is radikális változásokhoz vezetett az adatbázist használó metódusok körében. A tesztelés is sokkal nagyobb szerepet kapna a fejlesztés korai szakaszaiban, mert bár megvan a helye a manuális tesztelésnek, rengeteg felesleges UI kattintgatástól menthettem volna meg magamat. Ráadásul, ha többet teszteltem volna az egyes egységeket, néhány hibára hamarabb fény derült volna. Jobban átgondolnám azt is, hogy pontosan milyen Azure Function-ökre van szükségem, mert nagyjából hét olyan Function-t és hozzájuk tartozó repository-t írtam, amelyeket végül nem használtam élesben. Ezek ugyan kellettek ahhoz, hogy megszokjam a technológiát és a munkafolyamatot, de a jövőben ezt szeretném elkerülni.

A frontend fejlesztését is szervezettebben és határozottabban csinálnám, mert itt esetenként nem vagyok elégedett a kóddal, és rengeteg dolgot refaktorálnék, ha engedné az idő. Az összevisszaságnak leginkább az az oka, hogy sokáig nem volt egy nagy összkép a szemem előtt, csak mindig azt csináltam, amit úgy éreztem, hogy közelebb visz a következő kis mérföldkőhöz. Emiatt egyes megoldásaim ütköztek egymással, vagy olyan eredményt nyújtottak, ami ugyan működik, de nem vagyok vele teljesen elégedett. A backend kódom egy fokkal rendezettebb és jobbnak is érzem. A YouTube API-ról visszakapott adatok kezelésekor úgy gondolom, hogy voltak jó megoldásaim, és úgy működik az egész, ahogy vártam. Persze itt is rengeteg dolgon tudnék javítani, és már egy hónap múlva is kissé elborzadva fogok ránézni a kódra.

Ezek ellenére is elégedett vagyok az eredménnyel, mert úgy érzem, hogy az alkalmazás megfelel az előre lefektetett elvárásoknak. Sikerült egy olyan webalkalmazást készítenem, amely ha nem akarja a felhasználó direkt eltörni teljesen jól működik, mindezt egy szerintem érdekesnek mondható infrastruktúrán.

8. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

Az előző fejezetben is leírtam már néhány dolgot, amelyet szívesen implementálnék a jövőben. Ezek közül számomra a legmegvalósíthatóbbak a teszteléssel kapcsolatosak, nevezetesen a párhuzamosítás és a tesztek CI/CD pipeline-ba integrálása. Mindenképpen szeretném majd úgy átalakítani, akár a teszt projekt teljes felépítését refaktorálni, hogy az egészet jobban centralizáltá és átláthatóbbá tegyem. A refaktorálás részét képezné a tesztek futásának párhuzamosítása is. Ezzel már próbálkoztam, azonban nagyon vegyes eredményeket értem el, de egyértelműen látom benne a potenciált. Hasonlóan szeretném az Azure DevOps folyamatokba bekötni az Azure Function-öket is, hogy egy jobban kontrollált fejlesztési és tesztelési folyamatot kapjak. Ehhez azonban még több dolgot kell megvizsgálnom, többek között azt is, hogy ez milyen költségekkel járna.

Ahogyan az értékelési fejezetben említettem, a jövőben szeretném a YouTube-csatornák profilképét is lekérni, és megjeleníteni a követett csatornák csempéin. Ezenkívül szeretném "okosabbá" tenni a keresőmodult, hogy segítsen a felhasználónak megtalálni az általa keresett csatornát. Ennek azonban még nem néztem utána, és úgy érzem, hogy ez elég komplikált feladat. Ide köthető még egy olyan funkció is, amelynek az implementálásán már gondolkoztam, és úgy vélem, hogy különösebb probléma nélkül meg is tudnám oldani. Ez az, hogy egy követett csatornához tartozó különböző időpontokban készített lekérdezések összehasonlíthatóak legyenek, és ezekből kimutatásokat lehessen generálni, amelyek ábrázolják a metrikák változását az idő múlásával. Úgy gondolom, hogy ez látványos és hasznos funkció lenne, azonban az alapfunkciók mellett már nem maradt időm ennek a fejlesztésére.

Az utolsó dolog, amelyet a későbbiekben szeretnék implementálni, bár nagy munkát igényelne, az az, hogy a videókat külön-külön is eltárolja a rendszer. Jelenleg csak bizonyos metrikákat gyűjt ki, de ha teljes videómodellek lennének eltárolva, azokból rengeteg érdekes kimutatást lehetne készíteni. Például arról, hogy mely időszakban feltöltött videók a legnézettebbek, hogyan változott a lájkok száma bizonyos videók között, vagy átlagosan mennyi ideig néznek a nézők egy-egy videót.

9. ÖSSZEGZÉS

Ez a dolgozat célul tűzte ki egy olyan felhőalapú webalkalmazás megvalósítását, amely lehetővé teszi influenszerek követő- és lájkmérő metrikáinak lekérdezését, elemzését, valamint ábrázolását. Kulcsfontosságú szempont volt a rendelkezésre állás biztosítása, valamint az időzített adatlekérések implementálása a manuális lekérdezéseken túl. Örömmel mondhatom, hogy úgy érzem, sikerült teljesítenem ezeket a kitűzött célokat. Az elképzelt végeredmény azonban rengeteget változott a szakdolgozat írása során. Az irodalomkutatás és a hasonló rendszerek vizsgálata során kezdett körvonalazódni, hogy miként szeretném, hogy a webalkalmazás kinézzen, és hogyan valósíthatók meg a kitűzött funkciók gyakorlatban. Ekkor még határozottan úgy tűnt, hogy Instagram influencerek adatait kinyerni teljes mértékben lehetséges Intagram Graph API-n keresztül, ez azonban mire a fejlesztést elkezdtem megváltozott. Ebben a szakaszban az is kezdett körvonalazódni, hogy melyik felhőszolgáltató és annak mely szolgáltatásai lennének a legalkalmasabbak az infrastruktúra kialakítására, valamint mely programnyelv illeszkedne legjobban a backendhez és a frontendhez. Ekkor még React használatát terveztem a frontendhez. A tervezési fázisban igyekeztem meghatározni az adatbázis struktúrát, a fejlesztési menetrendet és a szükséges végpontokat. Azonban az itt megtervezett elemek egyike sem élte túl a fejlesztési szakasszal járó refaktorálást.

Mire elkezdtem a fejlesztést, már szereztem némi tapasztalatot az Angular keretrendszerrel, ezért végül ezt választottam a frontend elkészítéséhez. Sajnos kiderült, hogy ha továbbra is az Instagramról szeretnék adatokat kinyerni, akkor meg kell elégednem azzal, hogy az alkalmazásba bejelentkezett felhasználók csak a saját Instagram-fiókjuk metrikáit érhetik el. Ez nem felelt meg a kitűzött céloknak, ezért alternatívákat kerestem. A választás végül a YouTube-ra esett, ami jó döntésnek bizonyult, mert a Google API-k sokkal kezelhetőbbek és jobban dokumentáltak, mint a Meta API-k. A fejlesztés folyamán rengeteg változás történt az eredeti tervekhez képest. A tervezés menete teljesen másképp alakult, mert meg kellett tanulnom az új technológiák kezelését és az alapjaik elsajátítását. A legnagyobb változások az adatbázis struktúrát érintették: többször kellett új táblákat hozzáadni, ahogy új igények merültek fel az adatok tárolásával és összekapcsolásával kapcsolatban. Ezek a változtatások természetesen az adatbázist kezelő kódot is érintették. A backend oldalon a legnagyobb változást az autentikáció, a felhasználómodell bevezetése és az időzített adatlekérések implementálása jelentette. Az előbbi logikailag nem volt bonyolult, de sok új dolgot kellett megírni és integrálni, míg az utóbbi esetében először meg kellett értenem a technológia működését, majd azt, hogy miként illeszthetem a saját funkcióimhoz.

A tesztelés ugyan nagy küzdelem volt, de elégedettséggel töltött el a végén, hogy láttam a sok zöld pipát a tesztjeim mellett. Itt a kihívást leginkább a rutin, és a szemlélet

hiánya okozta, de idővel ahogy kényelmesebbé vált a Playwright használata, meg úgy általában a különböző tesztek írása könnyebbé vált a folyamat. A legnagyobb áttörés az volt, amikor sikerült elérnem, hogy az Azure Function-ök, ha tesztek hívták meg őket, dedikált teszt adatbázist használjanak. Az E2E tesztek célja a felhasználói folyamatok mint a bejelentkezés, adatlekérés vagy csatorna bekövetése elejétől a végéig történő ellenőrzése volt. Az API-teszteket az Azure Function-ök HTTP-hívásokra adott válaszainak ellenőrzésére használtam, míg az integrációs tesztek célja, annakvizsgálata volt, hogy az egyes komponensek hogyan dolgoznak együtt.

A fejlesztés és tesztelés során rengeteget tanultam. Egyértelmű, hogy a jövőben nagyobb figyelmet kell fordítanom a tervezésre és a tesztelésre, mivel ezek határozottan befolyásolják az időt, amelyet debugolással és refaktorálással kell tölteni. Örülök, hogy az infrastruktúra nagyjából úgy működött, ahogy elterveztem, és sikerült egy teljesen serverless backendet létrehozni, amely csak akkor termel költséget, ha ténylegesen fut. Az Azure Function-ök egy shared projekten osztoznak, amely tartalmazza a DbContextet, modelleket, konfigurációkat és repositorykat. Ezzel egy jól működő backendet sikerült megvalósítani.

Összességében hosszú folyamat volt, sokszor frusztráló és nehéz, viszont tagadhatatlanul jutalmazó a végeredményt látni. Még akkor is, ha jól tudom hogy messze van a tökéletestől, és 1000 dolgot lehetne még csiszolni, finomítani rajta, de én hoztam létre elejétől a végéig, minden hibájával, és minden eredményével együtt. Ezúttal szeretném megköszönni konzulensemnek Sipos Miklósnak, hogy segítséget nyújtott, tanácsot adott, időben válaszolt a kérdéseimre, és a szakdolgozati munka folyamatos monitorozását.

10. IRODALOMJEGYZÉK

[1] *InsTrack webalkalmazás*,
https://instrack.app/.

[2] *Keyhole webalkalmazás*,
https://keyhole.co/dashboard.

[3] *NotJustAnalytics webalkalmazás*,
https://www.notjustanalytics.com/.

[4] *Inflact webalkalmazás*,
https://inflact.com/.

[5] V. András, *Virtualizáció a gyakorlatban*,
https://prohardver.hu/teszt/virtualis_infrastruktura/virtualizacio_a_gyakorlatban.html.

[6] *Virtualizáció bemutatása*,
https://www.serco.hu/virtualizacio-elonyei-jobb-mint-a-valodi.

[7] *Konténerizáció bemutatása*,
https://ulx.hu/megoldasok/kontenerek-es-devops/.

[8] *Konténerizáció bemutatása*,
https://www.hwsw.hu/hirek/58930/kontener-kontenerizalas-felho-virtualizacios-szerver-kiszolgalo-docker-devops.html.

[9] *Felhőszolgáltatók definíció*,
https://azure.microsoft.com/hu-hu/resources/cloud-computing-dictionary/what-is-a-cloud-provider.

[10] *Felhőszolgáltatók elényei*,
https://x-com.hu/felhoszolgaltatas/.

[11] *Felhő Szolgáltatási Modellek bemutatása*,
https://www.invitech.hu/techpercek/tanuljunk-a-felho-nyelven.

[12] S. Máté, *AWS bemutatása*,
https://prezi.com/p/p4z5adrhy0vy/az-aws-bemutatasa/.

[13] *AWS, Azure felhőszolgáltatók összehasonlítása*,
https://www.guru99.com/azure-vs-aws.html.

[14] *Keretrendszerek bemutatása*,
https://nofluffjobs.com/hu/log/technologia-hu/keretrendszer-avagy-framework-de-mi-is-az/.

ÁBRAJEGYZÉK

TÁBLAJEGYZÉK

11. MELLÉKLET

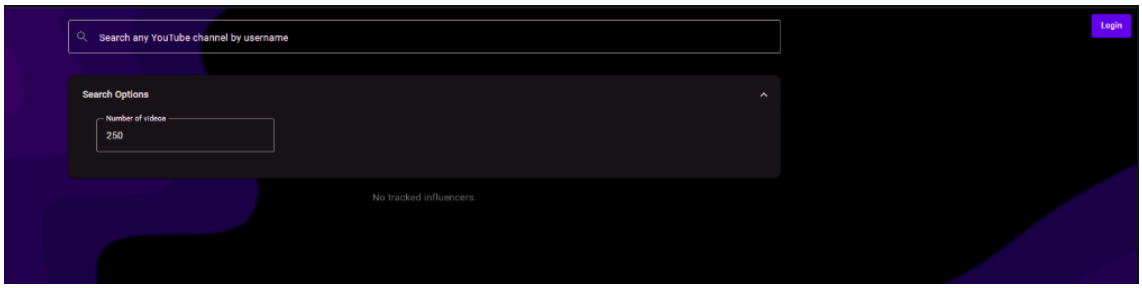
Ezen a linken keresztül lehet elérni a webalkalmazást: https://zealous-dune-091b3c003.5.azurestaticapps
Teszt Google felhasználó: lestertester74@gmail.com PW: LesterTester25$

Keresés menete:

11.1. ábra: A @ kezdetű névvel lehet keresni(@ nélkül) youtube csatornákat. Ebben az esetben-GOATiology.

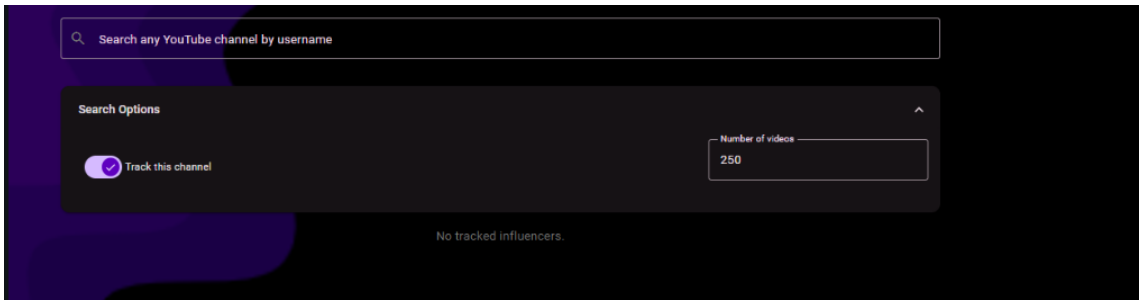
Olyan funkciók mint a csatornák követése, vagy időzitett lekérés beállitása csak bejelentkezés után elérhetőek

Funkciók: Youtube csatornákra a keresőben lehet keresni. Ezt meg lehet tenni bejelentkezés nékül is ilyenkor nincs lehetőség bekövetni a csatornát. 1 és 500 videó között lehet videókat lekérni.



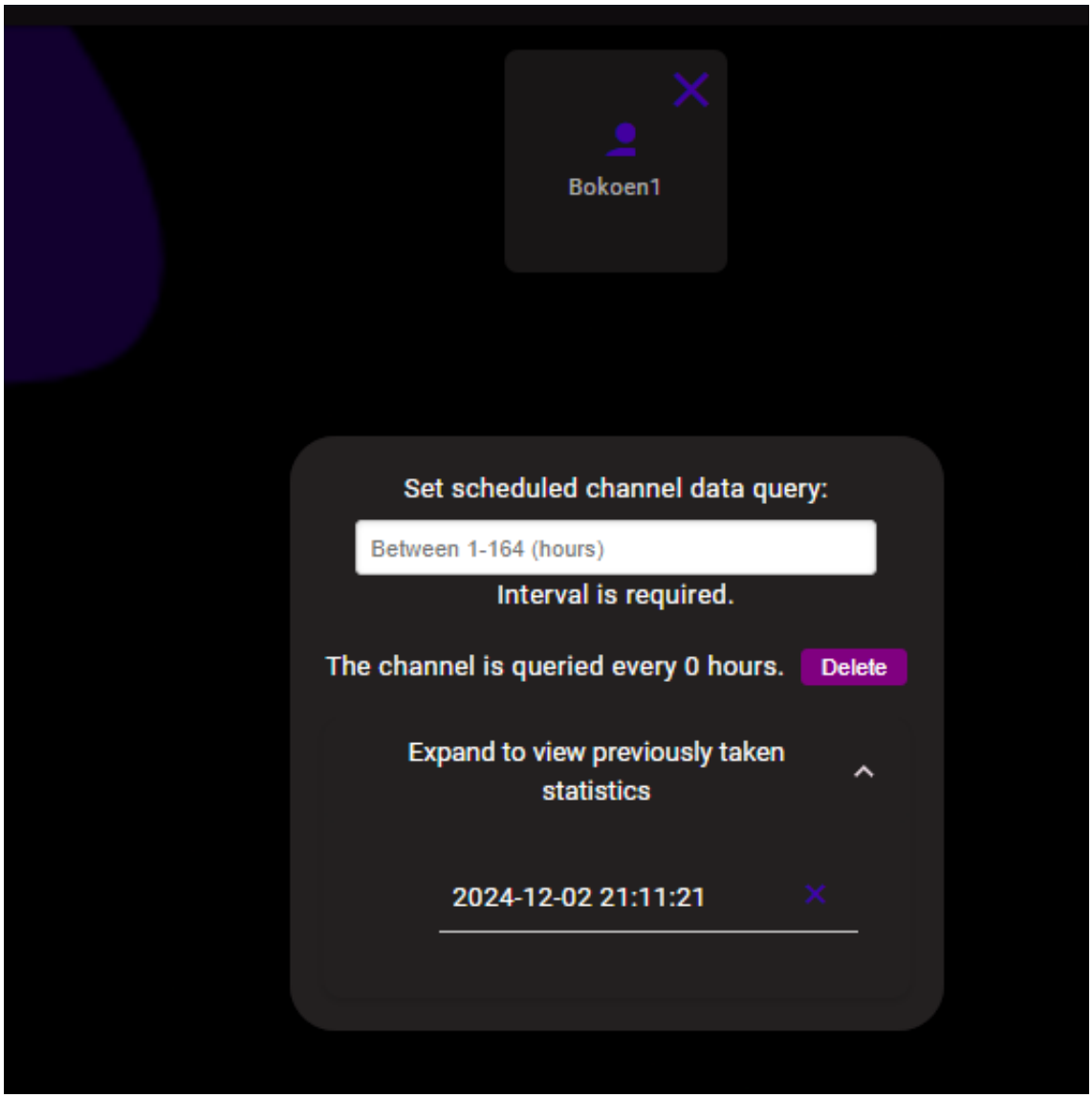
11.2. ábra: Autentikáció nélküli dashboard.

Bejelentkezés után már be lehet húzni a tracking slide-ot:



11.3. ábra: Sikeres autentikáció utáni dashboard.

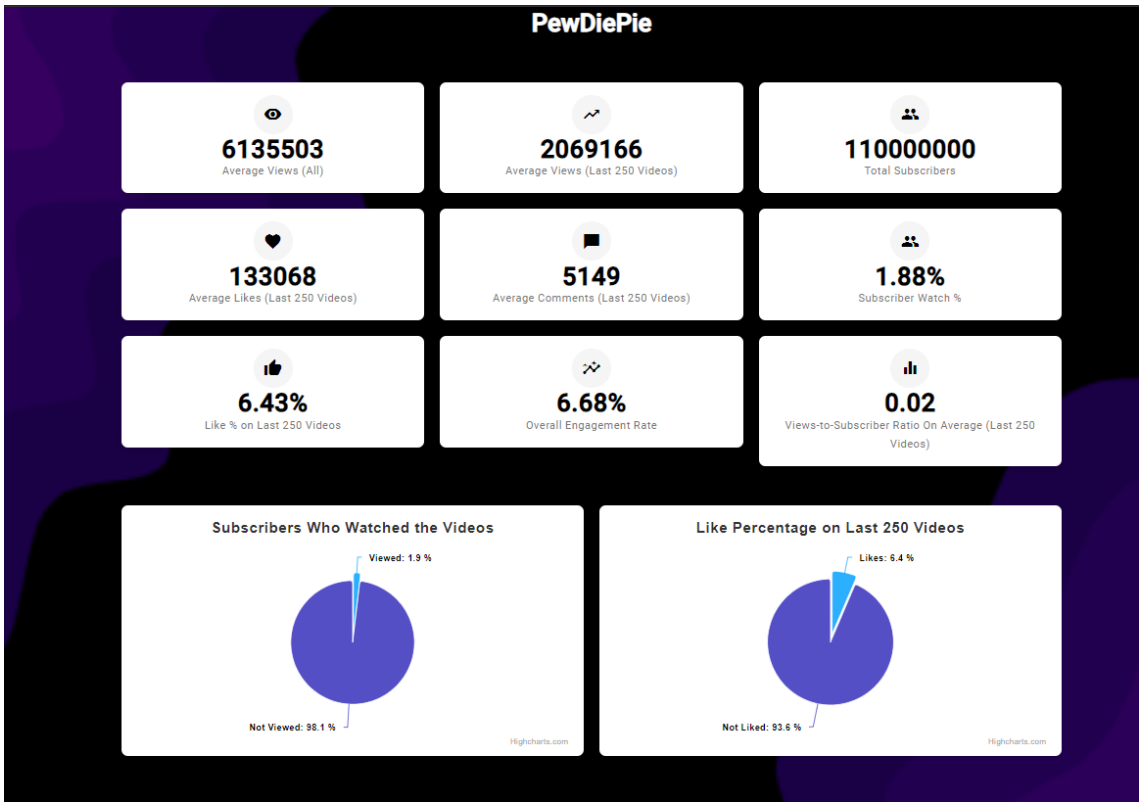
Követett felhasználó akciók:



11.4. ábra: Követett csatorna csempe.

Miután rákerestünk egy csatornára a slide behúzásával, és visszatérünk a dashboardra a csatornát szimbolizáló csempe látatóvá válik. Rákattintva a képen látható kép fogad. Itt beállíthatunk időzített keresést, vagy a dátumra kattintva megnézhetünk egy régebben készült statisztikát. Egyszerre 1 (én alkalmazásomba belpett) felhasználóhoz tartozó 1 követett csatornához 1 időzített lekérés tartozhat, mielőtt újat indítanál ki kell törölni az előzőt.

Statisztika oldal:



11.5. ábra: Statisztika oldal.