SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR,
INFORMATIKA SZAK

EventHub

Generatív AI módszereken alapuló eseménygyűjtő és kereső szoftverrendszer

Tudományos Diákköri Konferencia Dolgozat

Témavezető:
Novák Pálma-Rozália,
Egyetemi adjunktus

Végzős hallgatók:
Kukta Norbert-Attila
Tőtős Armand

2025

Kivonat

Sokunk számára ismerős lehet az a helyzet, amikor otthon, vagy akár egy új városban tartózkodva nem tudjuk, hogyan töltsük el a szabadidőnket. Gyakran előfordulhat, hogy a számunkra érdekes eseményekről nem szerzünk tudomást időben, vagy egyáltalán nem is értesülünk róluk. Ennek oka általában az, hogy az eseményekről szóló információk szétszórtan érhetők el, vagy teljesen elkerülnek minket.

Erre a problémára kínál megoldást az EventHub nevű webalkalmazás. Az alkalmazás célja, hogy segítsen a felhasználóknak nyomon követni a különböző eseményeket, így biztosítva, hogy soha ne maradjanak le a számukra fontos programokról. Az alkalmazás könnyen elérhetővé és áttekinthetővé teszi az eseményekről szóló információkat, megkönnyítve a szabadidő eltöltésének tervezését, bárhol is tartózkodjunk.

A rendszer frontendje Next.js/React és TypeScript használatával készült, biztosítva az intuitív felhasználói élményt, míg a backend .NET/C# és PostgreSQL technológiákra épül, hatékony adatkezelést és skálázhatóságot kínálva. Az adatgyűjtést támogató Facebook SDK, Puppeteer és az AI-Scraper kombinációja lehetővé teszi az események pontos és gyors begyűjtését, akár komplex weboldalakról is. Az alkalmazás térképes nézetet, szűrési lehetőségeket, naptár-integrációt és útvonaltervezést kínál, tovább bővítve az alkalmazás funkcionalitásait.

A platform különböző szerepköröket támogat (Guest, User, Organizer, Superadmin), így a vendégek regisztráció nélkül böngészhetnek, míg az eseményszervezők automatizáltan integrálhatják eseményeiket. A kitelepítés GitLab CI/CD pipeline-nal és Docker konténerizációval valósult meg, garantálva a stabil és biztonságos működést (pl. wildcard SSL tanúsítvány).

Kulcsszavak: események, adatgyűjtés, eseménykövetés, mesterséges intelligencia, automatizáció

Tartalomjegyzék

1. Bevezető **5**

2. Elméleti megalapozás **6**
2.1. Szakirodalmi áttekintő . . . 6
2.1.1. Események kezelésének digitalizálása és automatizálása . . . 6
2.1.2. Felhasználói élmény (UX) javítása technológiai eszközökkel . . . 6
2.1.3. Adatbiztonság és adatvédelem . . . 7
2.1.4. Versenytársak elemzése és fejlesztési irányok . . . 7
2.1.5. Inspiráció egy összefogó rendszer kialakításához . . . 7
2.2. Hasonló alkalmazások . . . 7
2.2.1. Mizu.ro . . . 8
2.2.2. VisitMures.com . . . 9
2.2.3. Meetup.com . . . 10
2.3. Összegzés . . . 10

3. A rendszer specifikációja **12**
3.1. Felhasználói követelmények . . . 12
3.1.1. Felhasználói szerepkörök . . . 12
3.1.2. Felhasználói elvárások . . . 14
3.2. Rendszerkövetelmények . . . 14
3.2.1. Funkcionális követelmények . . . 14
3.2.2. Nem funkcionális követelmények . . . 15

4. Felhasznált technológiák **16**
4.1. Frontend . . . 16
4.2. Backend . . . 16
4.2.1. Adatbázis . . . 16
4.3. Importer . . . 16
4.4. AI-Importer . . . 16
4.5. Könyvtárak . . . 17
4.6. Függőségmanagement . . . 17
4.7. Fejlesztői eszközök . . . 17
4.8. AI szolgáltatók . . . 18
4.9. Külső API-k . . . 18
4.10. Authorization server . . . 18

5. Megvalósítás **19**
5.1. Architektúra . . . 19
5.2. Frontend . . . 20
5.2.1. Autentikáció . . . 27
5.3. Backend . . . 30
5.3.1. Rétegzett architektúra . . . 30
5.3.2. Függőségi diagram . . . 34
5.3.3. Miért választottuk a rétegzett architektúrát? . . . 35
5.3.4. .NET Keretrendszer . . . 35
5.3.5. Unit és Integrációs tesztek . . . 36
5.3.6. REST API Konvenciók és Implementáció .NET-ben . . . 37
5.3.7. Middleware használata . . . 38
5.3.8. Machine-to-Machine (M2M) Autentikáció Auth0-val . . . 39
5.4. Adatbázis . . . 42
5.4.1. Code-First megközelítés . . . 43
5.4.2. ORM (Object-Relational Mapping) . . . 43
5.4.3. Adatbázis tervezés . . . 44
5.4.4. Távolságok kiszámítása a Haversine formulával . . . 44
5.4.5. Összegzés . . . 45
5.5. Importer . . . 45
5.5.1. Importer részei . . . 46
5.6. AI-Scraper . . . 49
5.6.1. AI-scraper pontossága . . . 50
5.6.2. AI-Scraper részei . . . 52
5.7. Kitelepítés . . . 54
5.7.1. GitLab CI Pipeline . . . 54
5.7.2. Kitelepítési Folyamat . . . 56
5.7.3. Docker és Docker Compose . . . 56
5.7.4. Kitelepítési Diagram . . . 61
5.7.5. A weboldal elérhetősége . . . 61
5.7.6. Összegzés . . . 62

6. Weboldal bemutatása **64**
6.1. Weboldal kezdeti tervei . . . 64
6.2. Weboldal jelenlegi állapota . . . 65
6.2.1. Főoldal és bejelentkezés . . . 65
6.2.2. Események oldal . . . 66
6.2.3. Eseményoldal . . . 67
6.2.4. Térkép . . . 68
6.2.5. Profil oldalak . . . 69
6.2.6. Kedvelt események . . . 72
6.3. Facebook oldalakkal való összekapcsolás . . . 73

7. Továbbfejlesztési lehetőségek **75**

8. Összefoglaló **76**

Ábrák jegyzéke **78**

Táblázatok jegyzéke **78**

Kódrészletek jegyzéke **78**

Irodalomjegyzék és hivatkozások **80**

1. fejezet

Bevezető

Képzeld el, hogy egy pillanatra megállsz a mindennapi rohanásban, és felteszed a kérdést: hogyan töltjük el valójában az időnket? Az életünk tele van lehetőségekkel, mégis sokszor észrevétlenül elsuhan melletünk egy-egy érdekes esemény, egy inspiráló előadás vagy akár egy közösségi találkozás, mert egyszerűen nem tudunk róla. Az információk szétszórtsága, a rohanó hétköznapok és a modern világ digitális zaja mind hozzájárul ahhoz, hogy lemaradjunk arról, ami igazán számít. Mi lenne, ha létezne egy olyan eszköz, amely összegyűjti és kézbe adja nekünk ezeket az értékes pillanatokat? Itt lép színre az EventHub, egy webalkalmazás, amelynek célja, hogy hidat teremtsen az ember és a körülötte zajló események között, legyen szó egy otthoni délutánról vagy egy ismeretlen városban tett látogatásról.

Mi magunk is gyakran szembesülünk azzal, hogy szabadidőnkben tanácstalanul keresgélünk, mit tehetnénk, ami valóban kikapcsol vagy feltölt minket. Talán mindannyian szembesülünk ezzel az érzéssel: a kíváncsisággal, hogy mi történik körülöttünk, és a tűnődéssel, hogy találkoznánk más emberekkel, de valahogy mindig elkerülik a figyelmünket az értékes programok. Az EventHub éppen erre kínál megoldást. Egy olyan rendszert biztosít, amely nem csupán egy újabb alkalmazás a sok közül, hanem egy útmutató, ami segít eligazodni a lehetőségek sokaságában. De miért is lenne szükségünk minderre? Azért, mert egy egységes, átlátható platform hiányában az események felfedezése időigényes és kaotikus – legyen szó a saját városunkról vagy egy új helyről, ahol éppen barangolunk.

A fő inspiráció egy konkrét igényből fakadt: a Székelyudvarhelyi Művelődési Ház vezetősége, különösen annak elnöke vetette fel, hogy az eseményszervezők és a közösség számára egyaránt könnyebbséget jelentene egy olyan felület, ahol az események automatikusan, központosítva jelennek meg. így nem kellene többé külön platformokon bajlódni a hirdetésekkel, nem kell manuálisan feltölteni minden adatot – az EventHub a Facebook-oldalakról és weboldalakról gyűjti be az információkat, hogy egyetlen kattintással elérhetővé tegye a programokat. Ez a rendszer nemcsak nekünk, felhasználóknak segít, hogy ne maradjunk le semmiről, hanem az eseményszervezőkről is leveszi a terhet, miközben az erdélyi közösségeket még közelebb hozza egymáshoz. Az EventHub nem pusztán egy technikai újítás – egy lehetőség, hogy az időnket tartalmasabban, tudatosabban éljük meg.

2. fejezet

Elméleti megalapozás

2.1. Szakirodalmi áttekintő

A technológiai újítások az eseménykezelésben az utóbbi évtizedekben jelentős mértékben átalakították az iparágat. Thirusanku és Lo Poh Ai (2024) [TA24] a *Technology Innovation in Event Management* című cikkükben az események kezelésében alkalmazott különböző technológiai fejlesztéseket vizsgálják, beleértve az automatizálást, a digitális eszközöket, a virtuális valóságot (VR) és a kiterjesztett valóságot (AR), valamint az adatgyűjtési technológiák fejlődését. A cikkben bemutatott technológiai újítások kulcsfontosságúak az események hatékony tervezéséhez, lebonyolításához és az élmény fokozásához, amely szorosan kapcsolódik az alkalmazásunk fejlesztéséhez.

2.1.1. Események kezelésének digitalizálása és automatizálása

A korábban említett cikk [TA24] egyik központi témája a digitális technológiák, mint például az *API-k*, *web scraping* és az automatizálás szerepe az események kezelésében. Thirusanku és Lo Poh Ai (2024) kiemelik, hogy az események gyors növekedésével párhuzamosan egyre fontosabbá válik az adatok automatizált gyűjtése és feldolgozása.

2.1.2. Felhasználói élmény (UX) javítása technológiai eszközökkel

A felhasználói élmény kulcsfontosságú eleme az eseménykezelő alkalmazások sikerének. Thirusanku és Lo Poh Ai (2024) arra is rámutatnak, hogy a technológiai fejlődés jelentős hatással van arra, hogyan érzékelik a felhasználók az eseményeket. A VR és AR alkalmazások egyre inkább lehetővé teszik a felhasználók számára, hogy gazdagabb és interaktívabb módon böngészhessenek az események között. A fejlesztők az alkalmazás UX tervezése során a felhasználók igényei alapján igyekeztek biztosítani, hogy az alkalmazás intuitív legyen, így a felhasználók könnyedén hozzáférhetnek az összes eseményinformációhoz és kereshetnek a kívánt események között.

2.1.3. Adatbiztonság és adatvédelem

A cikkben említett technológiai újítások mellett kiemelt figyelmet érdemel az adatbiztonság és adatvédelem kérdése, amely a modern eseménykezelő alkalmazások egyik alapvető aspektusa. Thirusanku és Lo Poh Ai (2024) hangsúlyozzák, hogy az adatvédelem fontossága nőtt az internetes és mobilplatformokon történő adatgyűjtés elterjedésével. Az EventHub alkalmazás fejlesztése során a biztonságos adatkezelés kulcsfontosságú volt, különösen a felhasználói adatok védelme érdekében. Az alkalmazás biztosítja, hogy minden adat megfeleljen a adatvédelmi előírásoknak, így a felhasználók bizalommal használhatják az események kezelésére szolgáló platformot.

2.1.4. Versenytársak elemzése és fejlesztési irányok

Thirusanku és Lo Poh Ai (2024) SWOT analízist végeznek a technológiai újítások hatásairól az eseménykezelésre, amely segít az iparági trendek megértésében és az alkalmazásokat érintő lehetőségek és kihívások azonosításában. Az EventHub esetében a versenytársak elemzése segíthet a fejlesztési irányok meghatározásában. A cikkben említett problémák, mint a manuális adatbevitel, a hiányos adatbázisok és a felhasználói élmény korlátozott fejlesztése, mind olyan kihívások, amelyeket az EventHub próbál orvosolni, különösen az automatikus adatgyűjtéssel és a fejlettebb felhasználói élmény biztosításával.

2.1.5. Inspiráció egy összefogó rendszer kialakításához

A cikk inspirációt adhatott az EventHub alkalmazás fejlesztéséhez, különösen az adatgyűjtés automatizálásában, a felhasználói élmény fokozásában és az adatbiztonság biztosításában.

Az EventHub alkalmazás célja az események automatizált gyűjtése különböző forrásokból, azok rendszerezése, valamint a felhasználók számára a könnyebb kereshetőség és naptárhoz való hozzáadás biztosítása. Az adatgyűjtési folyamatban kiemelkedő szerepet játszik a web scraping technológia, amely hatékony és gyors adatkinyerést tesz lehetővé különböző weboldalakról. Az alkalmazás a felhasználói élmény optimalizálására is nagy hangsúlyt fektet, például egyszerűsített felhasználói felülettel és könnyen navigálható eseménylistákkal.

Az EventHub fejlesztése során inspirációt meríthetett a modern eseménykezelő technológiákból, mint például az API-integrációk, a web scraping, az AR/VR alkalmazások, valamint a személyre szabott felhasználói élmény biztosítása. Az alkalmazás sikere az innovatív technológiai megoldások mellett az iparági trendek elemzésére és a felhasználói igények figyelembevételére is épül.

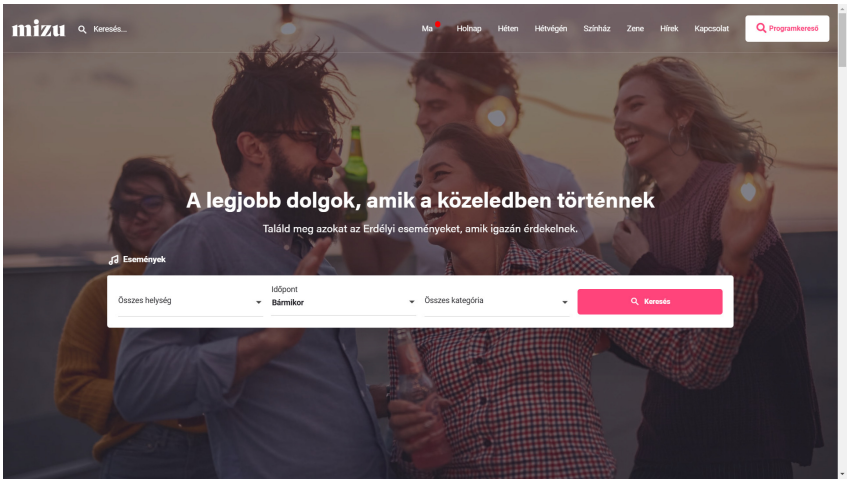
2.2. Hasonló alkalmazások

A projekt keretén belül több hasonló eseménykezelő platformot megvizsgáltunk, hogy azonosítsuk azokat a funkciókat és tulajdonságokat, amelyeket érdemes lenne figyelembe venni az EventHub tervezésekor. Az elemzések közül három példát említünk meg: a Mizu.ro, VisitMures.com és a Meetup.com weboldalakat. Ezek a platformok mind az

eseménykezelés területén használatosak, de pozitív tulajdonságaik mellett számos kihívással, negatívummal is rendelkeznek, amelyeket a projektünkben igyekeztünk megoldani, kiküszöbölni.

2.2.1. Mizu.ro

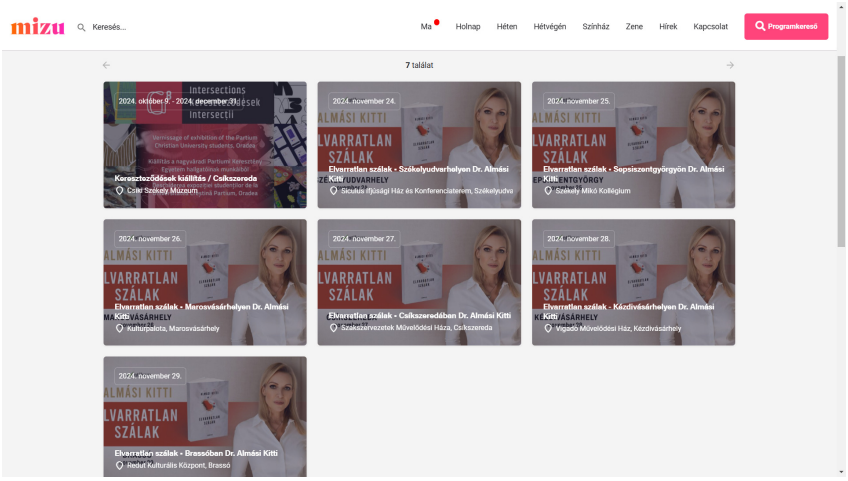
A Mizu.ro egy eseménykezelő oldal, amely különböző eseményeket tartalmaz, megkönnyítve az események böngészését a felhasználók számára. Az oldalon azonban időnként pontatlanságok és késések figyelhetők meg az események adataiban, ami arra utalhat, hogy az információk manuálisan kerülnek feltöltésre. Ez a folyamat időigényes, és nagy figyelmet igényel az eseményszervezők részéről. A manuális feltöltés következményeként az oldalon található események száma nagyon korlátozott. Az előbb említettek miatt az EventHub célja ezzel szemben a teljesen automatizált adatgyűjtés, amely lehetővé teszi, hogy az események emberi beavatkozás nélkül kerüljenek fel a rendszerbe. Az alkalmazásunk a Facebook és más eseményszervező oldalak adatait közvetlenül integrálja, ezáltal folyamatosan frissülő eseménykínálatot biztosít, amely naprakészebb, mint a manuális frissítés.



2.1. ábra. Mizu.ro kezdőlapja és eseménykategóriák

Pozitívumok

- A weboldalon az események számos kategóriára vannak bontva, amely segít a felhasználóknak, hogy könnyebben megtalálják az érdeklődésüknek megfelelő eseményeket (lásd 2.1. ábra).
- Az események helyszínek szerinti csoportosítása lehetővé teszi, hogy a látogatók könnyen megtalálják azokat az eseményeket, amelyek a közelükben vannak, vagy olyan helyszínen, amelyeket szívesen látogatnának. Ez különösen hasznos lehet azok számára, akik időszakosan egy adott városban vagy régióban keresnek programokat.



2.2. ábra. Mizu.ro szerény eseménykínálata

Negatívumok

- A Mizu egyik legnagyobb hátránya az lehet, hogy az eseményeket manuálisan kell feltölteni. Ez jelentős időt igényel igényel az eseményszervezők részéről, és így gyakran előfordulhat, hogy a friss események nem kerülnek időben közzétételre.

- Az oldalon megtalálható események száma viszonylag alacsony (lásd 2.2. ábra), amelyenek oka valószínüleg hasonlóképpen az időigényes adatfeltöltési folyamat lehet. Ez azt eredményezi, hogy az oldalon található eseménykínálat korlátozott, és a felhasználók nem mindig találnak új vagy változatos rendezvényeket. Ez a szűkös eseménykínálat csökkentheti a látogatottságot, és így a platformot kevésbé vonzóvá teheti a felhasználók számára.

2.2.2. VisitMures.com

A VisitMures.com weboldal szintén hasznos inspirációt nyújtott számunkra az EventHub fejlesztése során, hiszen ez az oldal a Marosvásárhelyhez közeli eseményeket gyűjti össze egy jól strukturált felületen. Ugyanakkor pontosan az ilyen példák alapján alakult ki az a célkitűzésünk, hogy az EventHub ne legyen korlátozva egyetlen régióra. Fontosnak tartjuk, hogy a felhasználók ne csak helyi, hanem távolabbi eseményeket is könnyedén megtaláljanak a platformon. Ezt az elképzelést személyes tapasztalatok is motiválták: például sokan vagyunk olyan helyzetben, hogy más városban élünk, mint ahol tanulunk vagy dolgozunk, és emiatt több régió eseményei is érdekelnek bennünket. Az EventHub célja tehát az, hogy minden térséget lefedve egy átfogó, naprakész eseménykínálatot biztosítson, amely alkalmazkodik a modern, mobilis életmódhoz.

Az oldal felhasználóbarát felülete letisztult és rendezett (lásd 2.3. ábra), ezzel segítve a felhasználókat, hogy könnyebben megtalálhassák a keresett eseményeket. Ezt a saját alkalmazásunkban inspirációként szolgált, hogy egy letisztult és egyszerű felhasználóbarát felületet tervezzünk.

2.3. ábra. Visitmures.com eseménykínálata

Pozitívumok

- Átlátható és barátságos felhasználói felület.
- A városközeli események naprakészek és az oldalt folyamatosan frissítik, hogy minden esemény megtalálható legyen.

Negatívumok

- Csak a városközeli események találhatók meg az oldalon.

2.2.3. Meetup.com

A továbbiakban nézzünk meg az első két erdélyi eseményekkel kapcsolatos példával ellentétben egy széleskörben elismert weboldalt. A Meetup.com az emberek közösségi érdeklődéséhez szervez eseményeket és találkozókat. A felhasználók különböző csoportokat és eseményeket hozhatnak létre, valamint csatlakozhatnak hozzájuk. Az EventHub az események gyűjtésére és könnyű elérésére összpontosít, akár regisztráció nélkül is. Ezzel szemben a Meetup inkább a közösségre fókuszál az események gyűjteménye helyett.

Pozitívumok

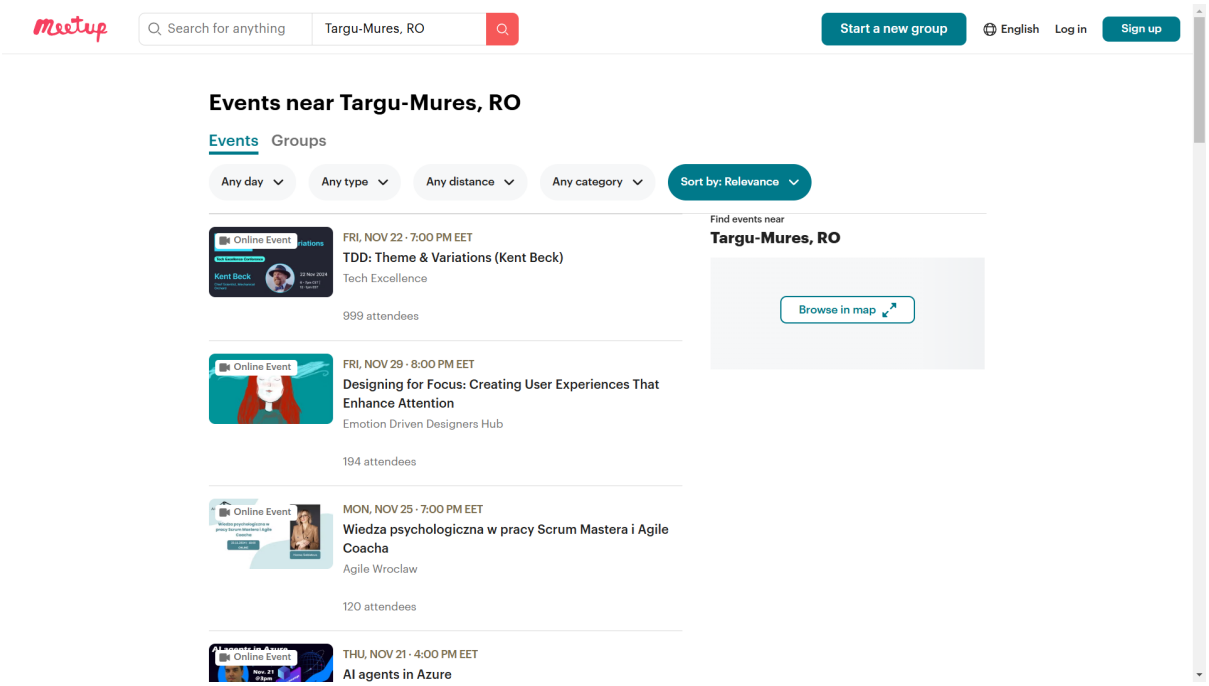
- Erős közösségi interakciók.

Negatívumok

- Csak a platformra regisztrált felhasználók számára elérhető a teljes eseménykínálat.

2.3. Összegzés

Az alábbi 2.1. táblázat összefoglalja a vizsgált weboldalak főbb tulajdonságait:



2.4. ábra. Meetup.com eseménykínálata.

	Mizu.ro	VisitMures.com	Meetup.com
Események frissítése	Valószínűleg manuális	Valószínűleg manuális	Felhasználói feltöltés
Közösségi funkciók	Nem érhető el	Nem érhető el	Erőteljes
Automatikus adatgyűjtés	Nem	Nem	Nem
Eseménykínálat	Korlátozott	Lokális	Széleskörű

2.1. táblázat. Hasonló platformok összehasonlítása

3. fejezet

A rendszer specifikációja

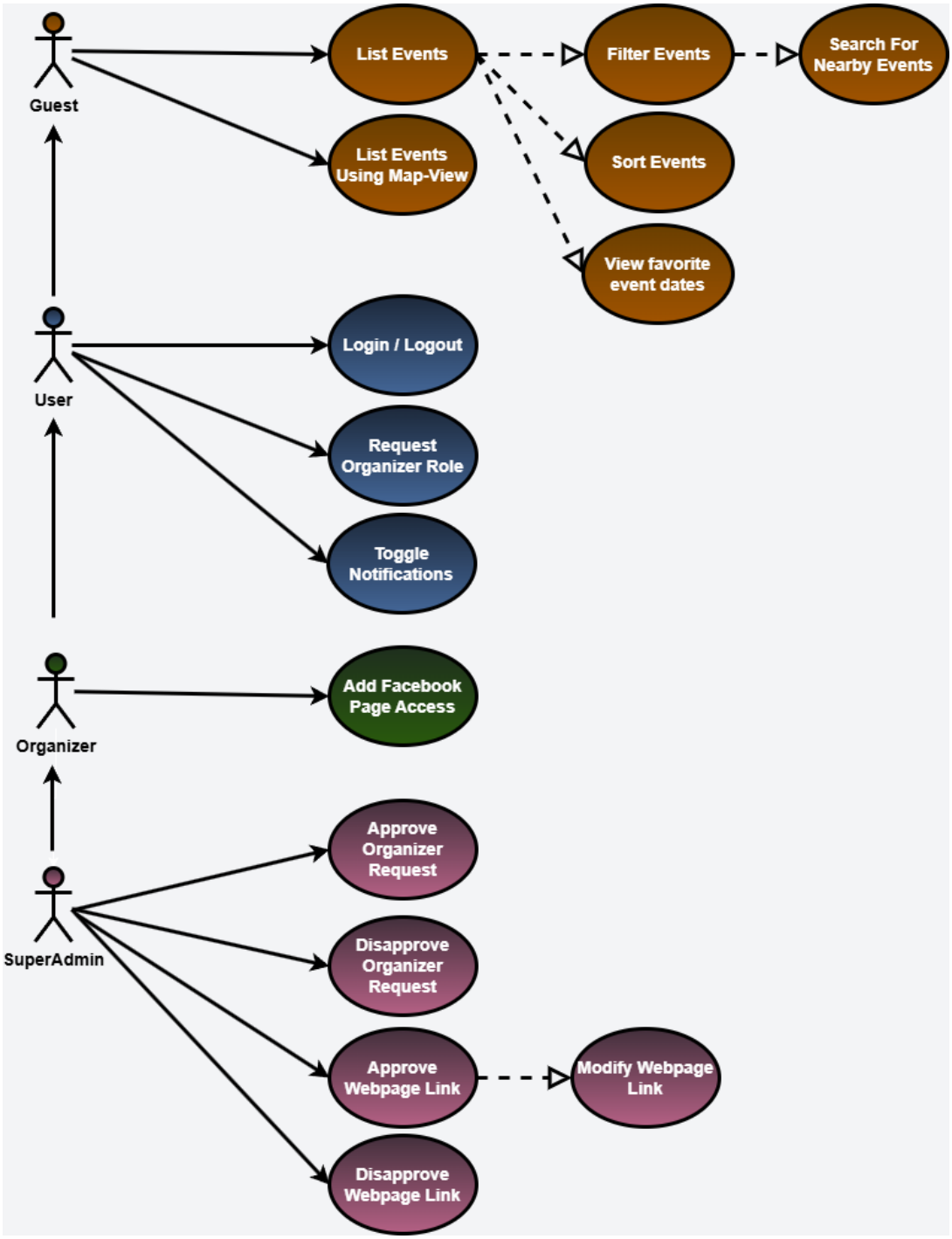
3.1. Felhasználói követelmények

Az EventHub alkalmazás célja, hogy egy felhasználóbarát platformot biztosítson események megtekintésére, azok kategóriák szerinti szűrésére és közvetlen hozzáférésre a releváns eseményekhez. A rendszer támogatja a különböző felhasználói szerepköröket, lehetőséget biztosítva események kezelésére, valamint a szervezők és adminisztrátorok szerepköreinek kialakítására. Az alábbi követelmények biztosítják, hogy az alkalmazás megfeleljen a célfelhasználók igényeinek.

3.1.1. Felhasználói szerepkörök

Az EventHub platformon különböző szerepköröket alakítottunk ki a felhasználók számára, amelyek meghatározzák az elérhető funkcionalitásokat és jogosultságokat (lásd 3.1. ábra).

1. **Guest (vendég)**:
    - Események listázása.
    - Események szűrése különböző paraméterek alapján.
    - Események térképes nézetben történő megtekintése.
    - Legközelebbi események keresése.
    - Útvonaltervezés megtekintett eseményekhez.
    - Események személyes naptárhoz való hozzáadása.
    - Kedvenc események dátumainak megtekintése.
2. **User (felhasználó)**:
    - Bejelentkezés és kijelentkezés.
    - Szervezői szerepkör igénylése.
    - Értesítések be- és kikapcsolása.
    - Facebook oldal hozzáférésének hozzáadása.



3.1. ábra. Használati eset diagram

3. **Organizer (eseményszervező)**:

   - Facebook oldal hozzáférésének megadása, amelyről az események begyűjtődnek.
   - Eseményoldal linkjének megadása, amelyről az események begyűjtődnek.
   - Weboldal link módosítása.

- Weboldal link jóváhagyása/elutasítása.

4. **Superadmin (weboldal adminisztrátor)**:
    - Szervezői szerepkör igények jóváhagyása és elutasítása.
    - Szervezői jogosultságok módosítása.

3.1.2. Felhasználói elvárások

Az alkalmazásnak elvárásokat kell, hogy teljesítsen annak érdekében, hogy a felhasználók teljeskörű kényelemben és biztonságban legyen részük:

- **Felhasználóbarát felület**: Az alkalmazásnak egyszerűen használhatónak és intuitívnak kell lennie.
- **Rugalmas szűrés**: Az események kategóriák vagy távolság alapján történő szűrése legyen gyors és pontos.
- **Integráció**: Az alkalmazás támogassa az események térképes megjelenítését és útvonaltervezést.
- **Biztonság**: Az alkalmazás megfelelő módon kell kezelje a felhasználók adatait, hogy azok ne jussanak ki illetéktelen személyekhez.
- **Pontosság és frissesség**: Az alkalmazás pontos, friss és megfelelő adatot kell szolgáltasson a felhasználóknak az eseményeket illetően, hogy csak valós és megbízható események legyenek elérhetőek az oldalon.

3.2. Rendszerkövetelmények

3.2.1. Funkcionális követelmények

Felhasználók kezelése

- A rendszer biztosítsa a felhasználók bejelentkezését és kijelentkezését Google, Facebook felhasználóval vagy saját e-mail segítségével.
- A felhasználók elérhetik profiloldalukat, ahol igényelhetik a szervezői szerepkört.
- Az eseményszervezők megadhatják Facebook oldalukat vagy egyéb eseményekkel kapcsolatos weboldalakat, ahonnan az események begyűjtődnek.

Események kezelése

- Az események listázása és szűrése különböző paraméterek alapján.
- A legközelebbi események megjelenítése térképes nézetben.
- Útvonaltervezés az események helyszínéhez.

3.2.2. Nem funkcionális követelmények

Biztonság

- Az alkalmazás biztosítsa a jogosultságkezelést és a felhasználói adatok védelmét.

Skálázhatóság

- Az alkalmazás könnyen bővíthető kell legyen és skálázható, annak érdekében, hogy jövőbeli bővítések érdekében ne kelljen nagyobb módosításokat végrehajtani a jelenlegi kódon.

Teljesítmény

- Az alkalmazás gyors válaszidőt biztosít a szűrési és listázási funkciók használata során.

Hálózati és hardver követelmények

- A weboldal eléréséhez a felhasználónak szüksége van egy böngészőt futtatni képes eszközre, valamint megfelelő internetkapcsolatra a weboldal betöltéséhez. A támogatott böngészők közé tartozik [Nex24b] a Google Chrome 64+, a Microsoft Edge 79+, a Mozilla Firefox 67+, az Opera 51+ és a Safari 12+, amelyek valamelyikének használata biztosítja a weboldal megfelelő működését és kompatibilitását.

4. fejezet

Felhasznált technológiák

A technológiai stack azoknak a technológiáknak és eszközöknek az összessége, amelyeket egy szoftver fejlesztéséhez és működtetéséhez használunk. Ez magában foglalhatja a programozási nyelveket, keretrendszereket, adatbázisokat és egyéb szükséges eszközöket.

4.1. Frontend

- **Next.js/React:** Egy React-alapú keretrendszer, amely támogatja a szerveroldali renderelést és a statikus oldalak generálását, így optimalizálva a teljesítményt.
- **JavaScript/TypeScript:** A JavaScript egy típusrendszerrel bővített változata, amely segít a hibák korai felismerésében és a kód olvashatóságának javításában.

4.2. Backend

- **.NET/C#:** Microsoft fejlesztési keretrendszer és nyelv.

4.2.1. Adatbázis

- **PostgreSQL:** Nyílt forráskódú, fejlett relációs adatbázis-kezelő rendszer.

4.3. Importer

- **JavaScript/TypeScript:** Böngészőben és szerveren futó nyelvek, TypeScript típusbiztonságot ad.

4.4. AI-Importer

- **Python:** Sokoldalú, magas szintű programozási nyelv.

4.5. Könyvtárak

- **Crawl4AI:** Weboldalak adatainak gyűjtésére szolgáló eszköz mesterséges intelligencia alkalmazásához.
- **Puppeteer:** Node.js könyvtár, amely lehetővé teszi a Chromium böngésző automatizálását.
- **Facebook SDK:** Eszközkészlet, amely lehetővé teszi a Facebook integrálását webes és mobil alkalmazásokba.
- **Tailwind:** CSS keretrendszer, amely gyors fejlesztést és testreszabhatóságot biztosít.
- **NextAuth:** Hitelesítési és jogosultságkezelési megoldás Next.js alkalmazásokhoz.
- **Leaflet & OpenStreetMap:** Térképes megjelenítéshez használt JavaScript könyvtár és nyílt forráskódú térképszolgáltatás.
- **Xunit:** .NET keretrendszerhez készült egységtesztelő eszköz.
- **I18next:** Lokalizációs könyvtár JavaScript és TypeScript alkalmazásokhoz.
- **Pino:** Nagy teljesítményű naplózó könyvtár Node.js alkalmazásokhoz.
- **Timezonefinder:** Könyvtár, amely a földrajzi koordináták alapján időzónát keres.
- **Pytz:** Python könyvtár, amely lehetővé teszi a különböző időzónák kezelését.

4.6. Függőségmanagement

- **Npm:** A Node.js csomagkezelője JavaScript könyvtárak telepítéséhez.
- **Pip:** A Python csomagkezelője modulok és függőségek kezelésére.
- **NuGet:** A .NET ökoszisztéma csomagkezelője könyvtárak kezeléséhez.

4.7. Fejlesztői eszközök

- **Prettier:** Kódszépítő eszköz, amely egységes formázást biztosít.
- **ESLint:** JavaScript és TypeScript kódelemző eszköz a hibák és rossz gyakorlatok kiszűrésére.
- **Roslyn Analyzers:** .NET kódelemző eszközök a minőség és teljesítmény javítására.
- **GitLab (pipeline & deploy):** CI/CD megoldás automatikus buildeléshez, teszteléshez és telepítéshez.
- **Docker:** Konténerizációs platform alkalmazások izolált futtatására.
- **VS Code & extensions:** Könnyen bővíthető, népszerű fejlesztői környezet.

4.8. AI szolgáltatók

- **Gemini:** Google fejlett AI modellje szöveg- és képgeneráláshoz.
- **Groq:** Nagy teljesítményű AI gyorsító hardver és szoftverplatform.

4.9. Külső API-k

- **Google Maps Geocoder:** Címek és földrajzi koordináták közötti átalakítást végző szolgáltatás.
- **Facebook Graph API:** Facebook-adatok elérésére és kezelésére szolgáló API.
- **OpenStreetMap:** Földrajzi nevek egységesítését és koordináták használatát elősegítő szolgáltatás.

4.10. Authorization server

- **Auth0:** Felhasználóhitelesítési és jogosultságkezelési platform.

5. fejezet

Megvalósítás

5.1. Architektúra

Az alkalmazás megvalósításához microservice architektúra [DM24] mellett döntöttünk, mivel több előnyt is kínált a projekt számára. Elsősorban a modularitás és karbantarthatóság szempontjából előnyös, mivel a különálló szolgáltatások egyszerűbben fejleszthetők. Ugyanakkor a microservice architektúra skálázhatóságot biztosít, lehetővé téve, hogy a rendszer bővíthető legyen és alkalmazkodjon a növekvő terheléshez. Végül a jövőbeli kiegészítések is könnyen megvalósíthatók ebben a keretrendszerben, mivel új szolgáltatások hozzáadása egyszerűen integrálható a meglévő struktúrába.

Az alkalmazás architektúrája öt fő komponensből áll, amelyek a microservice alapú megközelítést követve különálló, de egymással együttműködő részegységeket alkotnak.

1. **Frontend:** A felhasználói felület a Next.js keretrendszerre épül. Ez a komponens felelős az adatok megjelenítéséért és a felhasználói interakciók kezeléséért.

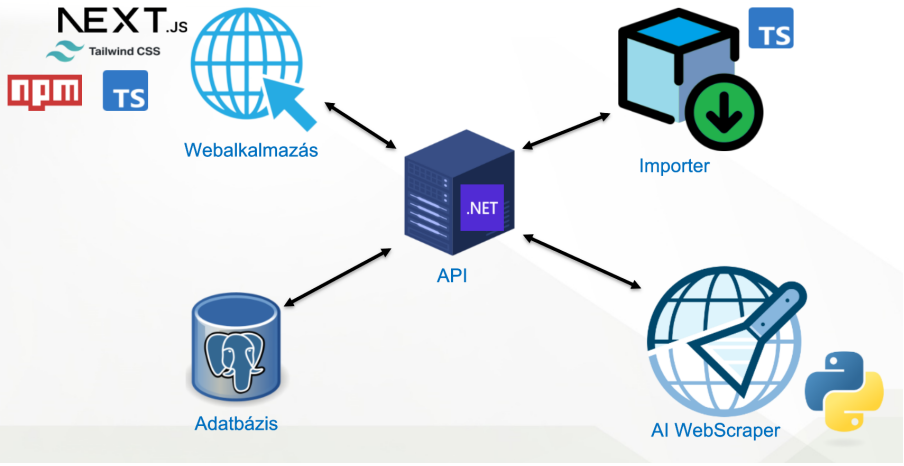
2. **Backend:** A backend egy többrétegű architektúrára épülő komponens, egy .NET alapú szolgáltatás, amely központi elemként biztosítja a folyamatos működést, a frontendre érkező kéréseket feldolgozza és kommunikál az adatbázissal, illetve az adatokat nyújtó szolgáltatásainkkal.

3. **Adatbázis:** Az adatok tárolásáért és kezeléséért a PostgreSQL adatbázis felelős.

4. **Scraper:** Ez az adatgyűjtő komponens két különböző technológiát (Facebook importer és Pupeteer) egyesít az események hatékony begyűjtése érdekében.

5. **AI Importer:** A másik adatgyűjtő komponens egy mesterséges intelligencián alapuló scraper, amely képes automatikusan felfedezni és begyűjteni az eseményeket különböző weboldalakról, anélkül, hogy manuális konfigurációra lenne szükség. Illetve az oldalon nem megfelelően megjelenő eseményadatokat Generatív AI segítségével kigenerálja a hiányzó adatokat.

Az öt komponens együttműködve biztosítja a rendszer megfelelő működését.



5.1. ábra. Architektúra diagram

5.2. Frontend

A projekt frontendje a Next.js [Nex24a] keretrendszerre épül, amely a React könyvtár alapjaira építve biztosít teljesítménybeli és fejlesztési előnyöket. A React egy komponensalapú JavaScript-könyvtár, amely egy virtuális DOM segítségével optimalizálja a felhasználói felület frissítését, így gyors és hatékony renderelést biztosít. A component tree struktúra lehetővé teszi az újrafelhasználható és jól szervezett UI-elemek létrehozását, ami átláthatóbbá és fenntarthatóbbá teszi a kódbázist.

Bár a React önmagában széleskörű könyvtár, a Next.js további előnyöket nyújt, amelyek miatt ezt választottuk a frontend alapjául:

- SSG és SSR támogatás: A Next.js támogatja a statikus oldalgenerálást (SSG) és a szerveroldali renderelést (SSR), amelyek jelentősen javítják az oldalbetöltési sebességet és a keresőoptimalizálást (SEO). Ez különösen fontos dinamikus tartalmak esetén.
- Gyors betöltés és optimalizálás: A beépített képkezelés (next/image), automatikus kódtördelés (code-splitting) és prefetching mechanizmusok segítenek csökkenteni az oldal betöltési idejét.

Az alkalmazást TypeScript segítségével írtuk, mivel a fordítási időben végzett típusellenőrzés segít megelőzni a hibákat és csökkenti a futásidejű problémákat.

Az alkalmazás részeit a következők szerint különítettük el:

- **Components:** Az egyes oldalakhoz tartozó komponensek kaptak helyet, mint például a Filter, Footer és Header. A nagyobb, összetettebb komponensek további alkomponensekre bontottuk, hogy kisebb és újrahasznosítható elemeket kapjunk, például a LocationFilter és CategoryFilter. Ez a megoldás megkönnyítette számunkra a fejlesztést és karbantartást.

  Példa pár komponensre:

- **Pages:** Ez tartalmazza az alkalmazás különböző oldalait, amelyek a webalkalmazás navigációs struktúráját alkotják. Minden oldal külön fájlban van, és a Next.js automatikusan kezeli az útvonalakat, így minden fájl egy-egy URL végpontot képvisel. Az oldalak logikáját és megjelenését itt építjük fel, figyelembe véve a komponensek és fordítások használatát. Ezen a struktúrán keresztül biztosítjuk, hogy az alkalmazás könnyen navigálható legyen, miközben minden oldal jól elkülöníthető és jól karbantartható marad. Ez a struktúra a Next.js Pages Router rendszerére épül, amely automatikusan generálja az útvonalakat a fájlok elhelyezkedése alapján. Az egyszerű .tsx fájlok oldalakat hoznak létre, míg a dinamikus útvonalakat szögletes zárójelben megadott fájlnevekkel ([id].tsx) lehet kezelni. Például egy events/[event-id].tsx fájl lehetővé teszi, hogy az adott esemény egyedi azonosítója alapján jelenjen meg a megfelelő tartalom. Az alkalmazás a következő oldalakkal rendelkezik:
  - *Main*: Az alkalmazás kezdőoldala, amellyel a felhasználók először találkoznak.
  - *Events*: Az események listáját és részletes információit tartalmazó oldal, ahol a felhasználók böngészhetnek és kereshetnek események között.
  - *EventDetail*: Az események saját oldala, ahol az adott esemény részletes információi, például leírás, időpont, helyszín és egyéb fontos adatok találhatók.
  - *Map*: Egy térkép, amely az események helyszíneit ábrázolja, segítve a felhasználókat a helyek felfedezésében.
  - *FavoriteEvents*: A felhasználók kedvenc eseményeit tartalmazó oldal.
  - *Profile*: A felhasználók saját profiljukat kezelhetik ezen az oldalon, ahol kérvényezhetnek eseménykezelő szerepkört és bekapcsolhatják majd az értesítéseket.
  - *AdminProfile*: Az eseménykezelő és Superadmin profiloldala, ahol az adminisztrátorok kezelhetik és feltölthetik az eseményeiket, összekapcsolhatják facebook oldalukkal vagy megadhatják saját weboldalukat, amelyről az események jönnek majd.
  - *PendingAdmins*: Az adminisztrátori jóváhagyásra váró felhasználók listája, ahol a Superadmin elfogadhatja az eseménykezelő szerepkör kéréseket.
  - *PendingLinks*: A eseményekkel kapcsolatos weboldalak linkjeinek jóváhagyását teszi lehetővé.
- **Public:** Tartalmazza az alkalmazás statikus fájljait, amelyek közvetlenül elérhetők a weboldalról. Ide tartoznak a képek, ikonok és egyéb fájlok.
- *Filter:* Lehetővé teszi események keresését név, kategória, dátum, helyszín és földrajzi elhelyezkedés (távolság) alapján.
- *Paginator:* Megjeleníti az aktuális oldal számát, az elérhető oldalak listáját, és lehetővé teszi az előző/következő oldalra léptetést. Az oldalváltáskor automatikusan az oldal tetejére görget. Ha túl sok oldal van, „...” jelölést használ a felesleges számok helyett (lásd 5.1. kódrészlet).

5.1. kódrészlet. Paginator részlet

```
const getPageNumbers = () => {
  const pages = [];
  const maxPagesToShow = 8;

  if (totalPages <= maxPagesToShow) {
    for (let i = 1; i <= totalPages; i++) {
      pages.push(i);
    }
  } else {
    const range = 2;
    const start = Math.max(1, currentPage - range);
    const end = Math.min(totalPages, currentPage + range);

    if (start > 1) pages.push(1);
    if (start > 2) pages.push("...");

    for (let i = start; i <= end; i++) {
      pages.push(i);
    }

    if (end < totalPages - 1) pages.push("...");
    if (end < totalPages) pages.push(totalPages);
  }

  return pages;
};
```

- **Interfaces:** Itt határoztuk meg azokat az interfészeket, amelyek biztosítják a típusosságot és a kód átláthatóságát. Ez a megoldás különösen hasznos, mivel segít minimalizálni a hibalehetőségeket, miközben elősegíti a könnyen karbantartható kód írását.
- **Locales:** Itt találhatók a nyelvi fájlok, amelyek tartalmazzák a magyar, román és angol nyelvekhez tartozó fordításokat. A nyelvi fájlokban külön vannak kezelve a különböző oldalak. Ezáltal minden nyelv esetében az egyes oldalakhoz kapcsolódó szövegek könnyen elérhetőek és módosíthatóak.
- **Models:** Ebben találhatók az adatmodellek, amelyek meghatározzák a különböző entitásokat, például az eseményeket és a helyszíneket. A modellek biztosítják az adatok konzisztens kezelését, valamint egységes formátumban való elérhetőségét a különböző komponensek számára.
- **Services:** A services tartalmazza az alkalmazás különböző külső API-khoz, a saját általunk implementált API-hoz és más szolgáltatásokhoz való kapcsolódását. Ez lehetővé teszi, hogy az adatkezelést külön rétegekre bontsuk.
  - Az *adminProfileService* az adminisztrátori profil kezeléséhez kapcsolódó különböző műveleteket tartalmazza. Itt található funkciók segítségével az eseménykezelők kezelhetik a weboldalukat (hozzáadhatnak és törölhetnek) és a Facebook oldalukat (összekapcsolhatják a Facebook oldalukat az EventHub weboldallal és ki is törölhetik onnan).

– Az *eventService* az események kezeléséhez szükséges műveleteket tartalmazza, beleértve az események lekérdezését, szűrést, kategóriák kezelését, valamint a kedvenc események hozzáadását vagy eltávolítását. Például a *fetchEventsWithFilters* szűrt eseményeket kérdez le különböző paraméterek (pl. név, dátum, helyszín) alapján, valamint oldalszámozással és paginálással segít a nagy adatmennyiség kezelésében (lásd 5.2. kódrészlet).

5.2. kódrészlet. Események lekérdezése szűrési paraméterekkel

```
    export const fetchEventsWithFilters = async (
     ...
): Promise<{ items: Event[]; totalPages: number }> => {
  try {
    const queryParams = new QueryParamsBuilder()
      .addParam("pageNumber", pageNumber)
      .addParam("pageSize", pageSize)
      .addParam("eventName", filters.name)
      .addParam("category", filters.category)
      .addParam("locationName", filters.location)
      .addParam("maxRange", filters.rangeValue)
      .addParam("latitude", filters.lat)
      .addParam("longitude", filters.lng)
      .addDateParam("startDate", filters.startDate)
      .addDateParam("endDate", filters.endDate, true)
      .addParam("sortBy", filters.sortBy)
      .addParam("descending", filters.descending)
      .addParam("favoriteOnly", filters.favoriteOnly)
      .addParam("togglePastDates", filters.togglePastDates)
      .build();

    const data = await getData<
      null,
      { items: Event[]; totalPages: number }
    >(
      `${apiEndpoints.data}/Exporter/events?${queryParams}`,
      200,
      undefined,
      true,
    );

    if (!data) {
      throw new Error(t("errors.failedToFetchEventsWithFilters"));
    }

    return data;
  } catch (error) {
    toast.error(t("errors.failedToFetchEventsWithFilters"));
    throw error;
  }
};
```

– A *facebookService* az alkalmazás Facebook integrációját biztosítja, lehetővé teszi a felhasználók számára a Facebook fiókkal való összekapcsolást és az autentikációs tokenek kezelését.

  - Az *icsService* az eseményekhez tartozó iCalendar fájlokat generál a strategy pattern [GHJV95] segítségével, amelyek lehetővé teszik az események naptárba történő importálását.
  - A *pendingAdminService* az adminisztrátorok és weboldal linkek állapotának kezelésére szolgáló funkciókat tartalmaz.

- **Styles:** A Styles mappa a projekt vizuális megjelenését tartalmazza. A *global.css* fájl általában az alapértelmezett stílusokat foglalja magában, amelyek az alkalmazás minden oldalára és komponensére kiterjednek.

- **Types:** A projektben használt típusdefiníciókat és enumokat tartalmazza.

- **Utils:** Olyan segédfüggvényeket és konstansokat tartalmaz, amelyek az alkalmazás különböző részein újra felhasználhatóak. Például endointok, stringek, útvonalak, dátum formázó mapper, autentikációval kapcsolatos segédmetódusok és builder pattern [GHJV95] a paramétereknek. A builder pattern, hogy egy segédosztály, amely segít URL lekérdezési paraméterek könnyű létrehozásában. Az *addParam* metódussal egyszerű értékeket, míg az *addDateParam*-al dátumokat adhatunk hozzá. A *build* metódus pedig visszaadja az összeállított paramétereket egy URL-kompatibilis formában (lásd 5.3. kódrészlet).

5.3. kódrészlet. QueryParamsBuilder osztály

```
 export class QueryParamsBuilder {
private params: URLSearchParams;

constructor() {
    this.params = new URLSearchParams();
}

addParam(
    key: string,
    value: string | number | boolean | null | undefined,
): this {
    if (value !== null && value !== undefined) {
        this.params.append(key, value.toString());
    }
    return this;
}

addDateParam(key: string, date: Date | null | undefined, isEndDate = false): this {
    if (date) {
        const adjustedDate = new Date(date);
        adjustedDate.setUTCHours(
            isEndDate ? 23 : 0,
            isEndDate ? 59 : 0,
            isEndDate ? 59 : 0,
            isEndDate ? 999 : 0,
        );
        adjustedDate.setUTCDate(adjustedDate.getUTCDate() + 1);
        this.params.append(key, adjustedDate.toISOString());
    }
    return this;
```

```
    }

    build(): string {
        return this.params.toString();
    }
}
```

Szintén az *utils* mappában helyezkedik el az *apiUtils* fájl, amely az API-hívások kezelésére szolgál. Ez az összes HTTP-metódust egységes kezelésére szolgál, tartalmazza a query felépítését, *fetch wrappereket*, amelyek biztosítják a HTTP kérések egyszerű és konzisztens kezelését és magasabb szintű API hívásokat (lásd 5.4. és 5.5 kódrészletek).

5.4. kódrészlet. Fetch wrapper

```
export const getData = async <Body, Response>(
    path: string,
    expectedHttpStatus: number,
    body?: Body,
    json?: boolean,
    queryParams?: QueryParams,
) => {
    return fetchWithMethod<Body, Response>(
        FetchMethod.GET,
        path,
        expectedHttpStatus,
        body,
        json,
        queryParams,
    );
};

export const postData = async <Body, Response>(
    path: string,
    expectedHttpStatus: number,
    body?: Body,
    json?: boolean,
    queryParams?: QueryParams,
) => {
    return fetchWithMethod<Body, Response>(
        FetchMethod.POST,
        path,
        expectedHttpStatus,
        body,
        json,
        queryParams,
    );
};
```

5.5. kódrészlet. Metódusok az API hívásokhoz

```
const fetchWithMethod = async <Body = undefined, Response = any>(
    fetchMethod: FetchMethod,
    path: string,
    expectedHttpStatus: number,
    body?: Body,
    json = true,
    queryParams?: QueryParams,
): Promise<Response | null> => {
    const queryString = buildQueryString(queryParams);
    const url = `${path}${queryString}`;
    const accesToken = getItemFromLocalStorage("accessToken");

    const headers: HeadersInit = {
        ...( json ? { "Content-Type": "application/json" } : {}),
        ...( accesToken ? { Authorization: `Bearer ${accesToken}` } : {}),
    };

    try {
        const response = await fetch(url, {
            method: fetchMethod,
            headers: headers,
            body: body
                ? json
                    ? JSON.stringify(body)
                    : (body as string)
                : null,
        });

        if (response.status === 401) {
            return null;
        }

        if (response.status !== expectedHttpStatus) {
            return null;
        }

        const responseBody = response.headers
            .get("content-type")
            ?.startsWith("application/json")
            ? await response.json()
            : await response.text();

        return responseBody as Response;
    } catch (fetchError) {
        toast.error("Fetch error");
        return null;
    }
};
```

- **Kofigurációs fájlok:** A Konfigurációs fájlok tartalmazzák a projekt működéséhez és fejlesztői környezetéhez szükséges beállításokat. Ezek segítenek a kódminőség, a buildelési folyamatok és a stílus egységesítésében. Mint például: *tsconfig, tailwind.config, .editorconfig, .eslintrc.*

- **i18n:** Az i18n (internationalization) a Next.js-ben lehetőséget biztosít arra, hogy többnyelvű alkalmazásokat építsünk, és könnyedén kezelhessük a különböző nyelvi verziókat. Az i18n segítségével a tartalom, például szövegek, helyi nyelv szerint jeleníthetők meg, így az alkalmazás egyszerre több nyelvet is támogathat. A Next.js beépített i18n támogatásával könnyen beállíthatjuk az alapértelmezett nyelvet és a támogatott nyelveket, emellett fordításokat kezelhetünk különböző fájlokban, ami lehetővé teszi, hogy a felhasználók az általuk preferált nyelven használhassák az alkalmazást.

- **Middleware:** Lehetőséget biztosít arra, hogy a kérés és a válasz között közvetlenül, mielőtt a kérés elérné az alkalmazás logikáját, vagy miután a kérés végrehajtódott, egységesen hajtsunk végre egyedi műveleteket, például autentikációt, naplózást, vagy a kérések átirányítását, mindezt anélkül, hogy módosítanánk a tényleges oldalak vagy API route-ok kódját, és lehetővé teszi az útvonalak kezelését, a kérések szűrését vagy módosítását globálisan vagy specifikus útvonalakra vonatkozóan.

5.2.1. Autentikáció

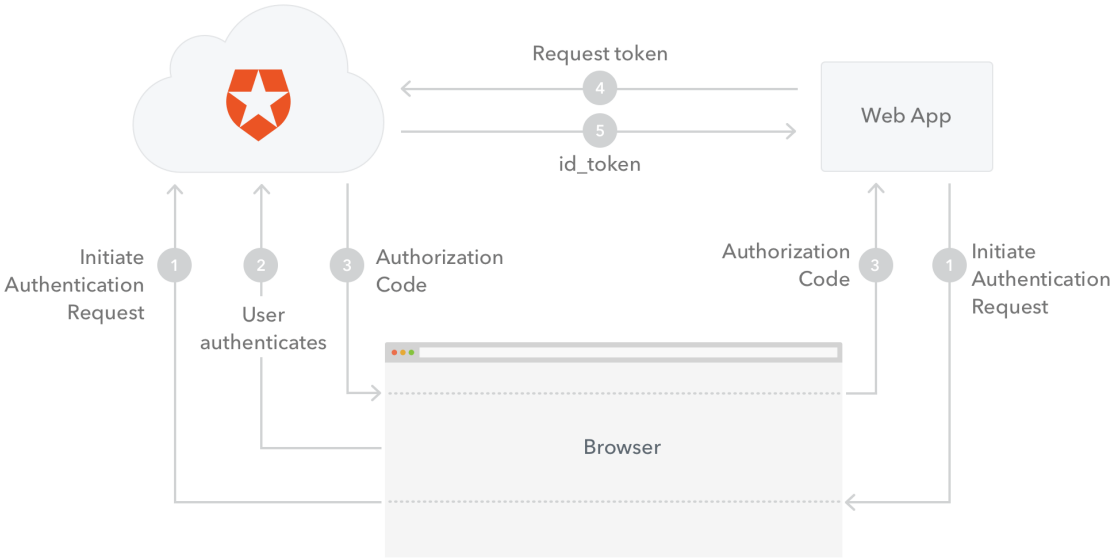
Az EventHub alkalmazás az OAuth 2.0 [Boy12] szabvány alapján végzi az autorizációt. Az OAuth 2.0 egy nyílt protokoll, amely lehetővé teszi a felhasználói adatok biztonságos megosztását egy harmadik fél számára anélkül, hogy a felhasználó jelszavát felfedné. A protokoll különböző autorizációs folyamatokat, úgynevezett „flow”-kat kínál, amelyek különböző típusú alkalmazásokhoz és felhasználási esetekhez illeszkednek. Az EventHub alkalmazásban a authorization code flow és a client credentials flow kerülnek alkalmazásra. Az authorization code flow a felhasználói bejelentkezés esetén használatos, míg a client credentials flow akkor, amikor az alkalmazás saját nevében kér hozzáférést.

Az authorization code flow során a felhasználó először az Auth0 felületére irányítódik, ahol bejelentkezik, majd az Auth0 egy kódot küld vissza az alkalmazásnak, amelyet az alkalmazás felhasználva egy hozzáférési tokent kap. A client credentials flow esetében az alkalmazás közvetlenül kérhet hozzáférést saját nevében, felhasználói beavatkozás nélkül, egy kliensazonosító és secret segítségével.

Az Auth0 [Aut24a] egy identitáskezelő szolgáltatás, amely lehetővé teszi az alkalmazások számára, hogy biztonságosan kezeljék a felhasználói hitelesítést és autorizációt. Az Auth0 biztosítja a különböző hitelesítési protokollok, mint például az OAuth 2.0, SAML és OpenID Connect támogatását, és lehetővé teszi külső hitelesítési szolgáltatók integrálását. Az Auth0 biztosítja a felhasználói adatok biztonságos kezelését, a jogosultságok ellenőrzését és a hozzáférési tokenek generálását, amelyek az alkalmazások számára lehetővé teszik az erőforrásokhoz való biztonságos hozzáférést.

Az EventHub alkalmazásban a felhasználók hitelesítése és szerepköreik kezelése az Auth0 szolgáltatásával lett megoldva (lásd 5.2. ábra). Az Auth0 biztosítja a különböző szerepkörök (guest, user, organizer, Superadmin) pontos és biztonságos kezelését. A hitelesítési folyamat integrálására a Next.js keretrendszerben a NextAuth könyvtárat használtuk, amely kapcsolódik az Auth0-hoz. Az autentikációs folyamat során JSON Web Tokenek kerülnek felhasználásra, amelyek lehetővé teszik a felhasználók hitelesítését és a jogosultságaik biztonságos kezelését.

- **Authorization Server:** Az Auth0 szolgáltatás felelős a felhasználók azonosításáért, a hozzáférési tokenek kiadásáért, valamint a különböző szerepkörök kezeléséért. Az Auth0 támogatja a JSON Web Tokenek (JWT) használatát, amelyek aláírásuk révén biztosítják a felhasználói jogosultságok hitelességét, és így bárki láthatja a tokenben tárolt adatokat, de azok módosítása vagy hamisítása csak a megfelelő titkosító kulccsal lehetséges.
- **Client Application:** A Next.js keretrendszer a NextAuth könyvtárat használja, amely az Auth0-tól származó hitelesítési információkat kezeli. A NextAuth ellenőrzi, hogy van-e bejelentkezett felhasználó. Ha nincs, akkor átirányítja az Auth0-hoz a bejelentkezéshez. Ezt követően az Auth0-tól kapott autorizációs kódot JWT tokenre cseréli, amelyet a kliens alkalmazás továbbküld a resource servernek.
- **Resource Server:** A .NET alapú backend csak érvényes hozzáférési tokenekkel rendelkező kéréseket fogad el. A tokeneket az Auth0 állítja elő, és a backend ezeket ellenőrzi a felhasználói jogosultságok hitelesítéséhez.
- **Resource Owner:**A felhasználók (resource owners) a hitelesítési folyamat során az Auth0 segítségével igazolják magukat. A sikeres hitelesítés után hozzáférést kapnak az alkalmazás erőforrásaihoz, a szerepkörüktől függően.



5.2. ábra. Autentikációs folyamat

A hitelesítési folyamatot a szekvenciadiagram mutatja be részletesen (lásd 5.3. ábra). A folyamat lépései a következők:

1. **Login kérés:** A felhasználó bejelentkezési kérését a Next.js keretrendszeren alapuló felhasználói felület (UI) fogadja.

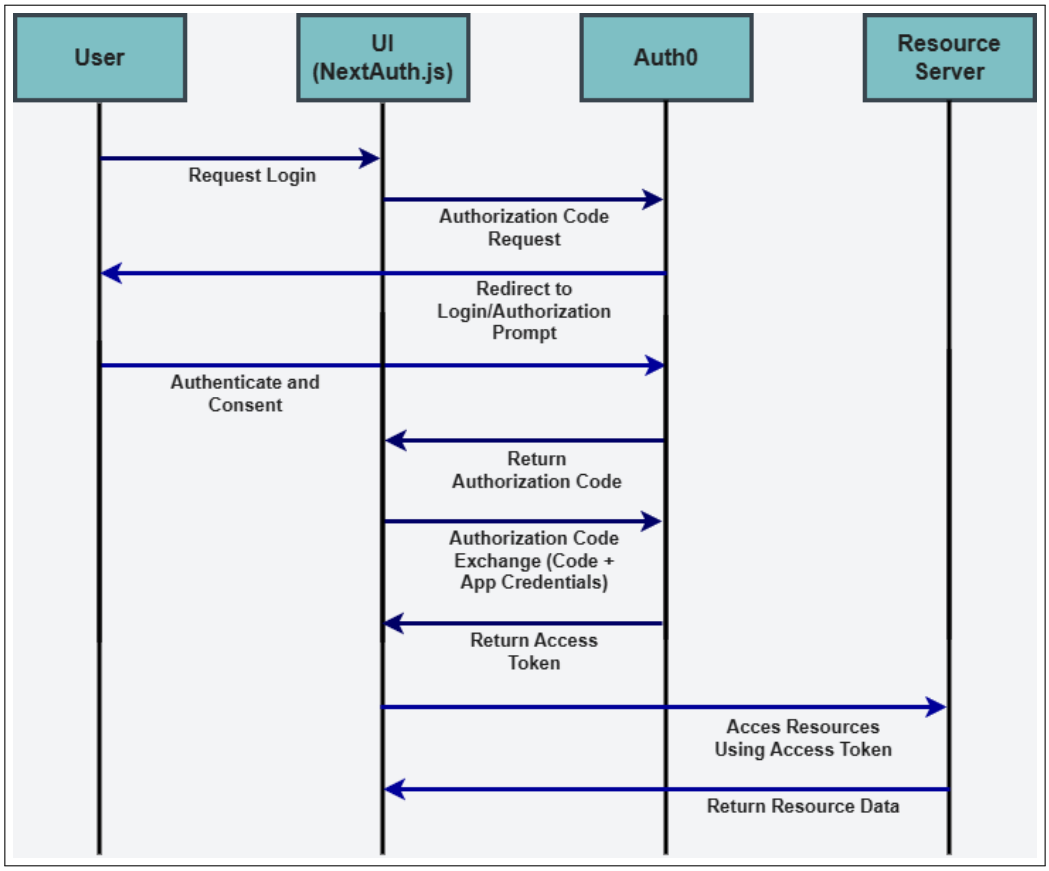
2. **Átirányítás az Auth0 SPA-hoz:** A NextAuth.js könyvtár használatával az alkalmazás átirányítja a felhasználót az Auth0 által biztosított egyszeri oldalas alkalmazáshoz (SPA).

3. **Hitelesítési válasz:** A sikeres hitelesítés után nyerünk egy generál Auth0-specifikus JWT tokent.

4. **Forrásokhoz való hozzáférés:** A felhasználói felület megkapja a tokent az Auth0-tól, amellyel biztonságos módon kérhet hozzáférést az EventHub erőforrásaihoz. A .NET alapú backend validálja a kapott tokent, és csak az érvényes kéréseket szolgálja ki.

5. **Erőforrások visszaadása:** A backend a jogosultságok ellenőrzése után visszaadja a kért adatokat a felhasználói felületnek.

Ez a folyamat garantálja, hogy csak hitelesített és jogosultsággal rendelkező felhasználók érhessék el az alkalmazás különböző erőforrásait. A NextAuth.js és az Auth0 szoros integrációja biztosítja az autentikáció és az autorizáció egyszerű és biztonságos kezelését.



5.3. ábra. Az EventHub alkalmazás hitelesítési folyamata

5.3. Backend

A projekt backendje egy .NET alapú API, amely a rendszer központi kommunikációs pontjaként szolgál. Az API felelős az adatok kezeléséért, a kérések feldolgozásáért, valamint a frontend és az adatbázis közötti kommunikációért. Az API-t több rétegre osztottuk, amelyek mindegyike specifikus feladatokat lát el. Az alábbiakban részletesen bemutatjuk ezeket a rétegeket, valamint kiemeljük, hogy miért választottuk a rétegzett architektúrát (layered architecture) és miért tekinthető a .NET ideális választásnak egy ilyen rendszer fejlesztéséhez.

5.3.1. Rétegzett architektúra

A projekt rétegzett architektúrát (layered architecture) [SM09] követ, amely a következő főbb rétegekből áll. Minden réteg egy specifikus feladatkörrel rendelkezik, és az elkülönített feladatok lehetővé teszik a kód modularitását, könnyű karbantarthatóságát és skálázhatóságát. Az alábbiakban részletesen bemutatjuk az egyes rétegek szerepét és feladatait:

- **Kontrollerek (Controllers)**: A kontrollerek a rendszer bemeneti pontjai, amelyek felelősek a HTTP kérések fogadásáért és a válaszok küldéséért. Ezek a komponensek közvetlenül kapcsolódnak a felhasználói felülethez vagy más service-ekhez, amelyek szintén használják az API-t, és az alábbi feladatokat látják el:
  - Kérések fogadása és érvényesítése (pl. adatbeviteli validáció).
  - A kérések továbbítása a szolgáltatási rétegnek a megfelelő üzleti logika feldolgozásához.
  - A szolgáltatási rétegtől kapott eredmények alapján HTTP válaszok generálása.
  - Hibakezelés és státuszkódok visszaadása a kliensnek.
  - **Middleware**: A middleware a kérések és válaszok közötti egyéni feldolgozásért felelős réteg. Ezek a komponensek a kérés feldolgozási folyamatában (request pipeline) helyezkednek el, és a kérések vagy válaszok módosítására, ellenőrzésére vagy kiegészítésére szolgálnak. A middleware feladatai közé tartozik:
    * **Autentikáció és autorizáció kezelése**:
      · A middleware felelős a felhasználók azonosításáért (autentikáció) és a hozzáférési jogosultságok ellenőrzéséért (autorizáció).
      · JWT tokenek ellenőrzése: A middleware ellenőrzi a JWT tokenek érvényességét, beleértve a signature validálását, a token lejáratának ellenőrzését, valamint a claim-ek (pl. issuer, audience, roles) ellenőrzését.
      · A tokenek aláírását a JWK (JSON Web Key) segítségével validálja, amelyet az identitásszolgáltató (pl. Auth0, Azure AD) által biztosított végpontról lehet lekérni.

    - **Hibakezelés és kivételek központosított kezelése**:
        - A middleware kezeli a kezeletlen kivételeket, és központosított hibaválaszokat generál.
        - Például, ha egy kivétel dobódik a kérés feldolgozása során, a middleware rögzíti a hibát, és egy standardizált hibaválaszt küld vissza a kliensnek (pl. 500 Internal Server Error).
    - **Naplózás és metrikák gyűjtése**:
        - A middleware naplózza a kérések és válaszok részleteit, amely segít a hibakeresésben és a rendszer teljesítményének monitorozásában.
        - Metrikákat gyűjt a kérések számáról, válaszidőkről és hibákról, amelyek felhasználhatók a rendszer állapotának elemzéséhez.
    - **Kérések vagy válaszok módosítása**:
        - A middleware módosíthatja a kéréseket vagy válaszokat, például fejlécék hozzáadásával vagy adatok kinyerésével.
        - Például egy middleware hozzáadhat egy egyedi fejlécet a válaszhoz, vagy kinyerhet adatokat a kérésből (pl. felhasználói adatok a JWT tokenből).

    A middleware célja, hogy központosított és újrafelhasználható logikát biztosítson a kérések és válaszok feldolgozásához, miközben csökkenti a kód ismétlődését a kontrollerekben és szolgáltatásokban.

A kontrollerek célja, hogy a kérések feldolgozása minél egyszerűbb és átláthatóbb legyen, miközben az üzleti logika a szolgáltatási rétegben marad.

- **Szolgáltatások (Services)**: A szolgáltatási réteg az alkalmazás üzleti logikájának központja. Ez a réteg felelős az adatok feldolgozásáért, az üzleti szabályok alkalmazásáért és a különböző műveletek koordinálásáért. A szolgáltatások feladatai közé tartozik:
    - Az üzleti logika implementálása (pl. adatátalakítás, számítások, szabályok alkalmazása).
    - Az adatelérési réteggel való kommunikáció az adatok lekéréséhez vagy mentéséhez.
    - Több adatforrás vagy szolgáltatás összehangolása egyetlen művelet során.
    - Tranzakciók kezelése, hogy az adatok konzisztensek maradjanak.

    A szolgáltatási réteg célja, hogy a kontrollerek és az adatelérési réteg között közvetítsen, és az üzleti logikát elkülönítse a felhasználói felülettől és az adatbázis rétegtől.

- **Adatelérési réteg (Repositories)**: Az adatelérési réteg felelős az adatbázis kommunikációért és az adatok perzisztens tárolásáért. Ez a réteg absztrakciót biztosít az adatbázis műveletek felett, így a szolgáltatási rétegnek nem kell közvetlenül az adatbázissal kommunikálnia. Az adatelérési réteg feladatai:
    - Adatok lekérése az adatbázisból (pl. lekérdezések végrehajtása).

- Adatok mentése, frissítése vagy törlése az adatbázisban.
- Adatbázis-specifikus logika implementálása (pl. összetett lekérdezések, tranzakciók).
- Az adatbázis függőségek elrejtése a szolgáltatási réteg elől.

Az adatelérési réteg célja, hogy az adatbázis műveleteket egyszerű és újrafelhasználható módon biztosítsa, miközben elrejti az adatbázis-specifikus részleteket.

- **Modellek (Models)**: A modellek az alkalmazás adatainak reprezentációjáért felelősek. Ezek az osztályok vagy struktúrák tartalmazzák az adatokat, amelyeket a rendszer feldolgoz és tárol. A modellek feladatai:
  - **Az adatok struktúrájának definiálása**: A modellek definiálják az adatok struktúráját, például entitásokat vagy DTO-kat (Data Transfer Objects).
  - **Adatvalidációk és korlátozások jelölése**: A modellek annotációk formájában tartalmaznak validációs szabályokat (pl. kötelező mezők, adattípusok, hosszkorlátozások). Ezeket a szabályokat a keretrendszer (ASP.NET Core) értelmezi és végrehajtja a kérések feldolgozása során.
  - **Az adatok közvetítése**: A modellek segítenek az adatok továbbításában a különböző rétegek között (pl. kontrollerek és szolgáltatások között).

  A modellek célja, hogy az adatok egységes és jól definiált struktúrával rendelkezzenek, és könnyen átadhatóak legyenek a rendszer különböző részei között.

A rétegzett architektúra előnyei közé tartozik a kód modularitása, a könnyű karbantarthatóság és a skálázhatóság. Az egyes rétegek elkülönítése lehetővé teszi, hogy a fejlesztők egymástól függetlenül dolgozzanak a különböző rétegeken, és a hibajavítás vagy funkcionalitások bővítése egyszerűbbé válik. Emellett a rétegzett architektúra támogatja a SOLID tervezési alapelvek betartását, különösen a **Single Responsibility Principle (SRP)** és a **Dependency Inversion Principle (DIP)** elveit.

A többrétegű architektúra megvalósítása

A rétegzett architektúra megvalósításakor különös figyelmet fordítottunk a rétegek közötti függetlenség biztosítására. Ennek érdekében az alábbi módszereket alkalmaztuk:

Interfészek használata

A rétegek közötti függőségeket interfészek segítségével oldottuk meg. Minden repository és service rendelkezik saját interfésszel, amelyek meghatározzák a rétegek közötti kommunikációt. Például:

- Az adatelérési rétegben (Repository) minden repository implementál egy közös interfészt, amely meghatározza az adatbázis műveleteket (pl. `IRepository<T>`).
- A szolgáltatási rétegben (Service) a szolgáltatások szintén interfészeken keresztül kommunikálnak az adatelérési réteggel, így a szolgáltatási réteg nem függ közvetlenül az adatelérési réteg konkrét implementációjától.

Ez a megközelítés lehetővé teszi, hogy a rétegek egymástól függetlenül fejlődjenek, és a kód könnyen karbantartható legyen.

Dependency Injection (DI)

A rétegek közötti függőségek kezelésére a **Dependency Injection (DI)** mintát alkalmaztuk. A .NET keretrendszer beépített támogatást nyújt a DI-hez, amely lehetővé teszi, hogy a függőségeket a rendszer injektálja a konstruktorokon keresztül. Ennek főbb előnyei:

- **Könnyű tesztelhetőség**: A függőségek injektálásával egyszerű mockolni a különböző rétegeket unit tesztek során.
- **Laza csatolás**: A rétegek közötti függőségek nem közvetlenek, hanem interfészeken keresztül történnek, ami növeli a rendszer rugalmasságát.
- **Konfigurálhatóság**: A függőségek regisztrálása és konfigurálása a program indításakor történik, így a kód tiszta és moduláris marad.

Függőségek regisztrálása és konfigurálása

A .NET keretrendszerben a függőségek regisztrálása a `Program.cs` fájlban történik. Például:

- A szolgáltatások és repository-k regisztrálása:

```
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IUserRepository, UserRepository>();
```

- Az adatbázis kontextus regisztrálása:

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseNpgsql(Configuration.GetConnectionString("DefaultConnection")));
```

A `AddScoped` metódus biztosítja, hogy a függőségek élettartama a kérés élettartamához kötődik, így minden kéréshez új példány jön létre.

Dependency Inversion Principle (DIP)

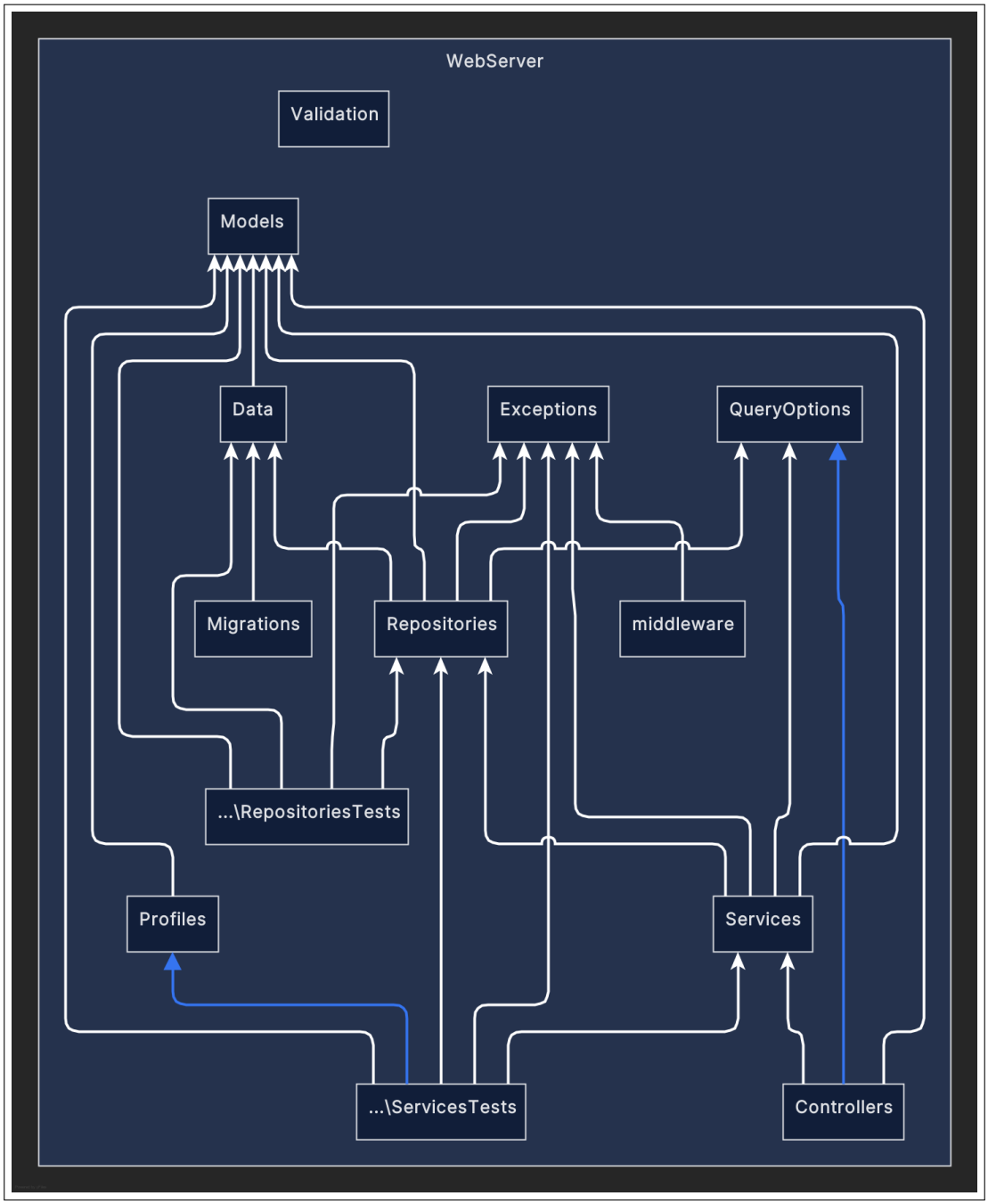
A **Dependency Inversion Principle (DIP)** betartásával a magas szintű modulok (pl. szolgáltatások) nem függenek az alacsony szintű moduloktól (pl. adatelérési réteg), hanem mindkettő absztrakciókon (interfészeken) keresztül kommunikál. Ez növeli a rendszer rugalmasságát és tesztelhetőségét. Például:

- A szolgáltatási réteg csak az `IUserRepository` interfészt ismeri, nem pedig a konkrét `UserRepository` osztályt.
- Az adatelérési réteg implementálja az `IUserRepository` interfészt, így a szolgáltatási réteg nem függ közvetlenül az adatelérési réteg részleteitől.

Összességében a rétegzett architektúra megvalósítása során az interfészek és a Dependency Injection használata lehetővé teszi, hogy a rendszer rugalmas, könnyen karbantartható és skálázható legyen, miközben betartja a SOLID tervezési alapelveket.

5.3.2. Függőségi diagram

A következő diagram (lásd 5.4. ábra). a backend rétegek közötti függőségeket ábrázolja. A diagram segít megérteni, hogyan kommunikálnak egymással a különböző komponensek, és hogyan épül fel a rendszer architektúrája.



5.4. ábra. Függőségek

A diagramon látható, hogy a **Kontrollerek** a felhasználói felületről érkező kéréseket fogadják, és továbbítják a **Szolgáltatások** rétegének. A **Szolgáltatások** az üzleti logikát végzik, és az **Adatelérési réteg** segítségével kommunikálnak az adatbázissal. A

Modellek az adatok egységes reprezentációjáért felelősek, míg a **Middleware** a kérések és válaszok közötti egyéni feldolgozásért felelős.

5.3.3. Miért választottuk a rétegzett architektúrát?

A rétegzett architektúra számos előnnyel jár, amelyek közül kiemeljük a következőket:

- **Szeparáció és modularitás**: A rétegek elkülönítése lehetővé teszi, hogy a különböző feladatokat egymástól függetlenül fejlesszük és karbantartsuk. Ez növeli a kód modularitását és átláthatóságát.
- **Könnyű karbantarthatóság**: A rétegek közötti egyértelmű határok miatt a hibajavítás és a funkcionalitások bővítése egyszerűbbé válik.
- **Dependency Inversion Elv (DIP)**: A rétegzett architektúra lehetővé teszi a Dependency Inversion Elv alkalmazását, amely szerint a magas szintű modulok (pl. szolgáltatások) ne függjenek az alacsony szintű moduloktól (pl. adatelérési réteg), hanem mindkettő függjön absztrakcióktól. Ez növeli a rendszer rugalmasságát és tesztelhetőségét.
- **Skálázhatóság**: A rétegek függetlensége lehetővé teszi, hogy a rendszer egyes részeit külön-külön skálázzuk az igényeknek megfelelően.

5.3.4. .NET Keretrendszer

A .NET keretrendszer [Mic24b] megfelelő választásnak bizonyult a rétegzett architektúra implementálásához, és számos előnnyel jár:

- **Modularitás és rugalmasság**: A .NET támogatja a moduláris fejlesztést, amely lehetővé teszi a különböző rétegek egymástól független fejlesztését és karbantartását.
- **Erős típusosság és biztonság**: A .NET erős típusossága és memóriakezelése biztonságosabbá teszi a kódot, és csökkenti a hibák kockázatát.
- **Széleskörű eszköztár és támogatás**: A .NET gazdag eszköztárral és könyvtárakkal rendelkezik, amelyek segítik a fejlesztést, például Entity Framework az adateléréshez, ASP.NET Core a webes API-khoz, és AutoMapper az adatleképezésekhez.
- **Platformfüggetlenség**: A .NET Core (és most már a .NET 5/6/7/8) platformfüggetlen, ami azt jelenti, hogy az alkalmazás futtatható Windows, Linux és macOS rendszereken is. Ez nagyobb rugalmasságot biztosít a fejlesztés és üzemeltetés során.

5.3.5. Unit és Integrációs tesztek

A projektben a tesztek írásához a **xUnit** [xUn24] keretrendszert használjuk, amely a .NET környezetben széles körben elterjedt tesztelési eszköz. A tesztek két fő kategóriába sorolhatók:

- **Unit tesztek**: Ezek a tesztek az egyes metódusok vagy osztályok működését ellenőrzik, függetlenül a rendszer többi részétől. A unit tesztek célja, hogy a kód egyes részei helyesen működjenek, és a hibákat gyorsan azonosítsák. Például egy szolgáltatási rétegben lévő metódus működését teszteljük, hogy helyesen dolgozza-e fel az adatokat.
- **Integrációs tesztek**: Ezek a tesztek a rendszer különböző részei közötti interakciót ellenőrzik. Például egy API végpont tesztelése, hogy helyesen kommunikál-e az adatbázissal, vagy hogy a különböző rétegek (pl. kontrollerek és szolgáltatások) megfelelően működnek-e együtt.

A projektben az integrációs tesztek során kiemelt figyelmet kapott az **in-memory adatbázis** használata. Az in-memory adatbázis lehetővé teszi, hogy a tesztek során ne legyen szükség külső adatbázisra, hanem a memóriában létrehozott adatbázison keresztül történjen az adatkezelés. Ez a megközelítés számos előnnyel jár:

- **Gyors tesztelés**: Mivel az adatbázis a memóriában fut, a tesztek végrehajtása sokkal gyorsabb, mint egy külső adatbázis használata esetén.
- **Egyszerű beállítás**: Az in-memory adatbázis könnyen beállítható és elindítható, így nincs szükség bonyolult adatbázis konfigurációkra vagy külső függőségekre.
- **Ismételhetőség**: Minden tesztfuttatás során az adatbázis ismét létrejön, így a tesztek egymástól függetlenül futnak, és nincs szükség az adatbázis állapotának manuális visszaállítására.

A .NET környezetben a **Entity Framework Core** támogatja az in-memory adatbázis használatát, amely lehetővé teszi, hogy a **DbContext** osztályt közvetlenül a memóriában lévő adatbázison keresztül teszteljük. Ez különösen hasznos, mivel a .NET nem támogatja a **DbContext** osztályok közvetlen mockolását (pl. Moq keretrendszerrel). Az in-memory adatbázis használata tehát nemcsak praktikusabb, hanem hatékonyabb is, mivel a valós adatbázis működését szimulálja anélkül, hogy külső függőségekre lenne szükség.

Például egy integrációs teszt során az alábbi lépéseket követjük:

1. Létrehozunk egy in-memory adatbázist a teszt kezdetén.
2. Betöltjük a szükséges tesztadatokkal az adatbázist.
3. Végrehajtjuk a tesztelendő API hívást vagy szolgáltatási metódust.
4. Ellenőrizzük, hogy az eredmények megfelelnek-e az elvárt kimeneteknek.
5. Az adatbázis automatikusan törlődik a teszt végén, így nincs szükség manuális tisztításra.

Ez a megközelítés lehetővé teszi, hogy egyszerre teszteljük a unit és integrációs teszteket, hiszen az in-memory adatbázis mindkét esetben használható. Így a tesztelési folyamat hatékonyabbá válik, és a hibák gyorsabban azonosíthatók.

5.3.6. REST API Konvenciók és Implementáció .NET-ben

A REST (Representational State Transfer) [Fie24] egy architektúra stílus, amelyet széles körben használnak webes API-k fejlesztéséhez. A REST API-k célja, hogy egyszerű, skálázható és karbantartható módon biztosítsák a kommunikációt a kliens és a szerver között. A REST alapelveit követve az API-k erőforrásokon alapulnak, és HTTP metódusokkal (GET, POST, PUT, DELETE stb.) végzik a műveleteket.

REST Alapelvek

A REST API-k fejlesztése során az alábbi alapelveket kell követni:

- **Erőforrás-alapú architektúra**: A REST API-kban minden entitás (pl. felhasználók, események) egy erőforrásként van kezelve, amelyet egy URI (Uniform Resource Identifier) azonosít. Például egy felhasználó erőforrás elérése a `/users` URI-n keresztül történik.
- **HTTP Metódusok használata**: A REST API-kban a HTTP metódusok használatával végezzük a műveleteket az erőforrásokon:
  - **GET**: Erőforrás lekérése vagy lista lekérése (pl. `GET /events` vagy `GET /events/1`).
  - **POST**: Új erőforrás létrehozása (pl. `POST /organizers`).
  - **PUT**: Erőforrás teljes frissítése (pl. `PUT /locations/1`).
  - **PATCH**: Erőforrás részleges frissítése (pl. `PATCH /events/1`).
  - **DELETE**: Erőforrás törlése (pl. `DELETE /events/1`).
- **Állapotmentesség (Stateless)**: A REST API-k állapotmentesek, ami azt jelenti, hogy minden kérés tartalmazza az összes szükséges információt a szerver számára a kérés feldolgozásához. A szerver nem tárol állapotot a klienssel kapcsolatban.
- **Reprezentációk (Representations)**: Az erőforrások különböző formátumokban (pl. JSON, XML) reprezentálhatók. A leggyakrabban használt formátum a JSON, amely könnyen olvasható és feldolgozható.

REST API Konvenciók .NET-ben

A .NET keretrendszer, különösen az ASP.NET Core, kiválóan támogatja a REST API-k fejlesztését. Az alábbiakban bemutatjuk a REST API-k .NET-ben történő implementációjának legfontosabb konvencióit és gyakorlatait:

- **Controller-ek és Route-ok**: A REST API végpontokat a .NET-ben controllerekkel valósítjuk meg. Minden controller egy adott erőforráshoz kapcsolódik, és a route-ok segítségével határozzuk meg, hogy melyik HTTP metódus melyik metódust hívja meg. Például:
  - `[HttpGet("users")]` egy GET kérést kezel, amely visszaadja az összes felhasználót.

  - `[HttpGet("users/{id}")]` egy GET kérést kezel, amely egy adott felhasználót ad vissza.
  - `[HttpPost("users")]` egy POST kérést kezel, amely új felhasználót hoz létre.

- **DTO-k (Data Transfer Objects)**: A REST API-kban gyakran használunk DTO-kat az adatok továbbítására. A DTO-k segítenek elkülöníteni az API réteget az adatbázis rétegtől, és lehetővé teszik az adatok formázását. Például egy felhasználó létrehozásakor a kliens egy `UserDTO` objektumot küld, amelyet a szerver validál és feldolgoz.

- **HTTP Státuszkódok**: A REST API-kban fontos a megfelelő HTTP státuszkódok használata, hogy a kliens tisztában legyen a kérés eredményével. Például:
  - `200 OK`: A kérés sikeresen lefutott, és a válasz tartalmazza a kért adatokat.
  - `201 Created`: Az erőforrás sikeresen létrejött, és a válasz tartalmazza az új erőforrás azonosítóját.
  - `400 Bad Request`: A kérés érvénytelen vagy hiányos adatokat tartalmaz.
  - `404 Not Found`: A kért erőforrás nem található.
  - `500 Internal Server Error`: Szerverhiba történt a kérés feldolgozása során.

- **Swagger/OpenAPI Dokumentáció**: Az API-k dokumentációja elengedhetetlen a fejlesztők számára. A .NET-ben a Swagger/OpenAPI segítségével automatikusan generálhatunk API dokumentációt, amely tartalmazza az összes végpontot, paramétereket és válaszokat. Ez lehetővé teszi a fejlesztők számára, hogy könnyedén teszteljék és megértsék az API-t.

- **Validáció és Hibakezelés**: A REST API-kban fontos a bemeneti adatok validációja és a hibák központosított kezelése. A .NET-ben a beépített validációs attribútumok (pl. `[Required]`, `[StringLength]`) segítségével ellenőrizhetjük a bemeneti adatokat, és a hibákat egy központosított hibakezelő middleware segítségével kezelhetjük.

Összegzés

A REST API-k fejlesztése során fontos betartani az alapelveket és konvenciókat, hogy az API könnyen használható, skálázható és karbantartható legyen. A .NET keretrendszer kiváló eszközöket biztosít a REST API-k fejlesztéséhez, beleértve a controller-eket, DTO-kat, validációt és hibakezelést. A Swagger/OpenAPI dokumentációval pedig könnyedén dokumentálhatjuk és tesztelhetjük az API-t, ami növeli a fejlesztés hatékonyságát.

5.3.7. Middleware használata

A projektben több egyéni middleware-t is használunk, amelyek a kérések és válaszok közötti feldolgozásért felelősek. A .NET keretrendszerben a middleware-ek a kérések feldolgozási folyamatában (request pipeline) helyezkednek el, és sorban egymás után hajtódnak végre. A kérés a pipeline-on keresztül halad, és minden middleware feldolgozhatja

a kérést vagy a választ, mielőtt továbbítja a következő middleware-nek vagy a végpontnak (pl. controllernek).

Két fő middleware-t emelünk ki:

- **Authorization Middleware**: Ez a middleware felelős azért, hogy a bejövő kérésekben lévő JWT tokeneket ellenőrizze, és a felhasználó jogosultságait ellenőrizze. Ha a token érvényes, a middleware továbbítja a kérést a következő rétegnek, ellenkező esetben hibát dob. Ez a middleware általában a pipeline elején helyezkedik el, hogy a jogosultságok ellenőrzése a kérés feldolgozása előtt történjen.
- **User Data Extraction Middleware**: Ez a middleware a JWT tokenből kinyeri a felhasználó adatait (pl. felhasználói azonosító, szerepkörök), és ezeket a kérés kontextusába helyezi. Ez lehetővé teszi, hogy a későbbi rétegekben ne kelljen újra feldolgozni a token tartalmát. Ez a middleware általában az Authorization Middleware után következik, hogy a felhasználó adatai már elérhetők legyenek a további feldolgozás során.

Ezen kívül a projektben egy harmadik, egyedi middleware-t is használunk, amely a hibakezelésért felelős:

- **Error Handling Middleware**: Ennek a middleware feladata, hogy kezeletlen kivételeket elkapjon, és ezeket a hibákat hozzáadja a HTTP válaszhoz. A middleware a kivételek részletes információit (pl. hibaüzenet, státuszkód) tartalmazó választ küld vissza a kliensnek. Ennek köszönhetően a controller rétegben nincs szükség külön kivételkezelésre, mivel a middleware központilag kezeli az összes hibát. Ez a middleware általában a pipeline végén helyezkedik el, hogy minden más middleware és végpont által dobott kivételt el tudjon kapni.

A middleware-ek sorrendje a .NET pipeline-ban a következőképpen épül fel:

1. **Authorization Middleware**: Jogosultságok ellenőrzése.
2. **User Data Extraction Middleware**: Felhasználói adatok kinyerése a tokenből.
3. **Egyéb middleware-ek**: További feldolgozások (pl. naplózás, adatellenőrzés).
4. **Controller réteg**: A kérés végleges feldolgozása.
5. **Error Handling Middleware**: Kezeletlen kivételek kezelése és hibaválaszok generálása.

Ez a struktúra lehetővé teszi, hogy a hibakezelés központosított legyen, és a kód tisztább, karbantarthatóbb maradjon.

5.3.8. Machine-to-Machine (M2M) Autentikáció Auth0-val

Az alkalmazásban az importer szolgáltatásoknak szükségük van biztonságos hozzáférésre a backend API-hoz, hogy az események adatait feltölthessék. Ennek érdekében Auth0 Machine-to-Machine (M2M) [Aut24b] autentikációt használunk, amely lehetővé teszi, hogy az importer szolgáltatások biztonságosan kommunikáljanak a backenddel. Az M2M autentikáció során az importer szolgáltatások hozzáférési tokeneket kapnak, amelyekkel hitelesítik magukat a backend API felé.

Auth0 M2M Autentikáció Beállítása

Az Auth0 M2M autentikáció beállításához az alábbi lépéseket követtük:

1. **Auth0 Alkalmazás Létrehozása**: Az Auth0 felületén létrehoztunk egy új alkalmazást, amelyet Machine-to-Machine (M2M) típusúként jelöltünk meg. Ez az alkalmazás reprezentálja az importer szolgáltatásokat, és lehetővé teszi, hogy azok hozzáférési jogosultságokat kapjanak a backend API-hoz. Az alkalmazás létrehozása során megadtuk az importer szolgáltatások nevét és leírását, valamint hozzárendeltünk egy ügyfél-azonosítót (Client ID) és egy titkos kulcsot (Client Secret), amelyeket az importer szolgáltatások használnak a tokenek kéréséhez.

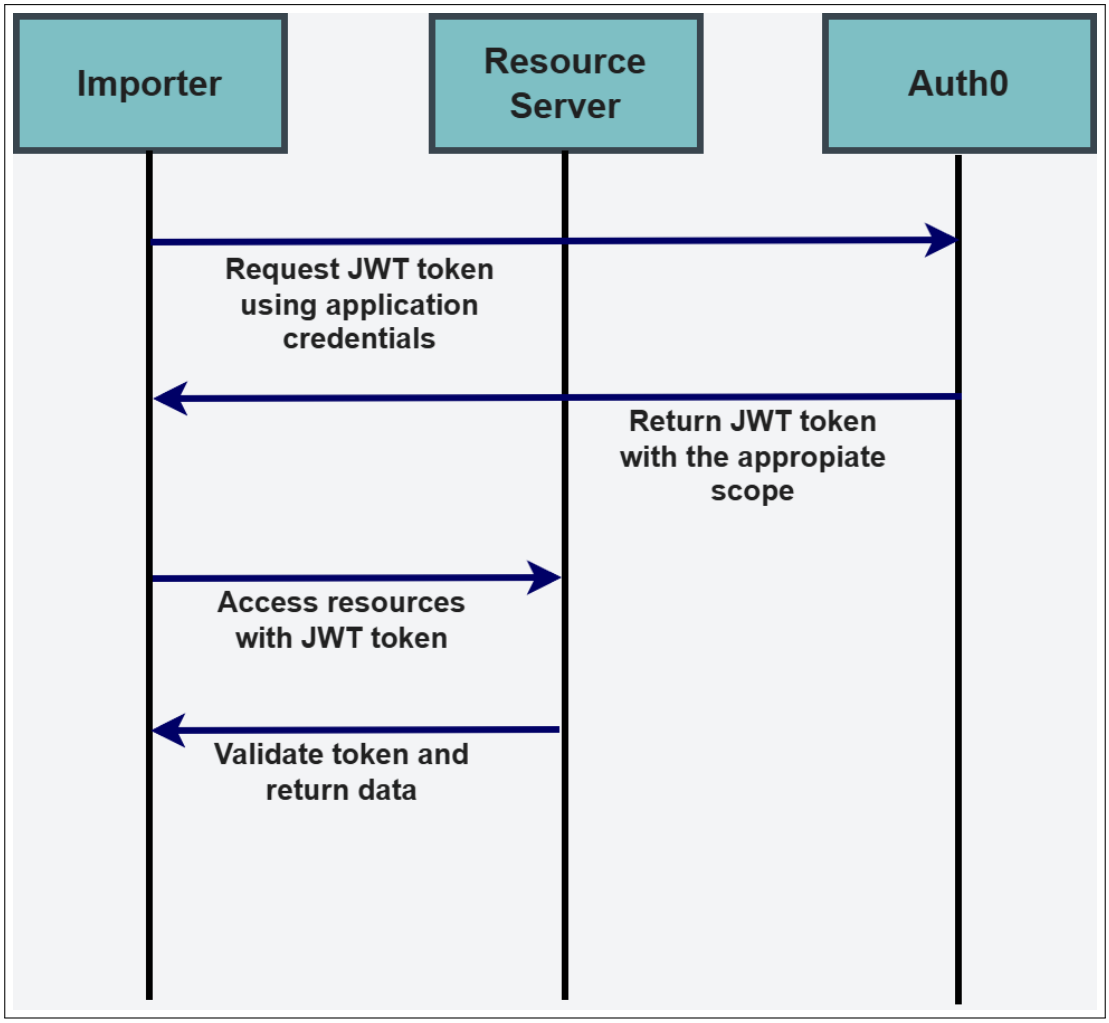
2. **API Regisztrálása Auth0-ban**: A backend API-t regisztráltuk Auth0-ban, hogy az importer szolgáltatások hozzáférhessenek az API végpontokhoz. Az API regisztrálása során megadtuk az API azonosítóját (audience), amelyet a tokenek kérésénél használunk. Az audience értéke egyedi, és biztosítja, hogy a tokenek csak az adott API-hoz legyenek érvényesek.

3. **Scope-ok Beállítása**: A backend API-hoz tartozó scope-okat definiáltuk, amelyek meghatározzák, hogy az importer szolgáltatások milyen műveleteket hajthatnak végre. Például a `import:events` scope lehetővé teszi az események importálását, míg a `read:events` scope csak az események olvasását engedélyezi. A scope-ok segítségével korlátozhatjuk az importer szolgáltatások hozzáférését, és biztosíthatjuk, hogy csak azokhoz a műveletekhez férjenek hozzá, amelyekre valóban szükségük van.

4. **Client Credentials Flow**: Az importer szolgáltatások a Client Credentials Flow segítségével kérnek hozzáférési tokeneket az Auth0-tól. Ez a folyamat olyan esetekben használatos, amikor egy alkalmazásnak (jelen esetben az importer szolgáltatásoknak) kell kommunikálnia egy másik alkalmazással (a backend API-val) anélkül, hogy felhasználói beavatkozásra lenne szükség. A folyamat során az importer szolgáltatások elküldik az ügyfél-azonosítójukat és a titkos kulcsukat Auth0-nak, és cserébe hozzáférési tokeneket kapnak.

5. **Token Kérése**: Az importer szolgáltatások a következő lépésekkel kérnek hozzáférési tokeneket (lásd 5.5. ábra):

   - Az importer szolgáltatás elküld egy HTTP POST kérést az Auth0 token végpontjára, amely tartalmazza az ügyfél-azonosítót, a titkos kulcsot, az API audience értékét, valamint a kért scope-okat.
   - Az Auth0 ellenőrzi a kérést, és ha minden adat helyes, visszaküld egy hozzáférési tokent, amely tartalmazza a kért scope-okat és egy lejárati időt.
   - Az importer szolgáltatás ezt a tokent használja a backend API-hoz intézett kérések során. A token minden API kérésnél az Authorization fejlécben kerül elküldésre.



5.5. ábra. Client Credentials Flow folyamata

.NET Konfigurációk

A backend API-ban a következő konfigurációkat állítottuk be az Auth0 M2M autentikáció támogatásához:

- **Auth0 Domain és Audience**: Az Auth0 domain és az API audience értékeit a konfigurációs fájlban tároljuk. Ezek az értékek szükségesek a JWT tokenek validálásához, mivel a tokenek érvényességét az Auth0 domain és az audience alapján ellenőrizzük.
- **JWT Token Validáció**: A .NET keretrendszer beépített JWT hitelesítési mechanizmusát használjuk a tokenek validálásához. A JWT hitelesítés konfigurálása során megadjuk az Auth0 domain-t és az API audience-t, valamint beállítjuk a tokenek élettartamának és aláírásának ellenőrzését. Ez biztosítja, hogy csak az Auth0 által kiadott és érvényes tokenek legyenek elfogadva.
- **Authorize Attribútum**: Az API végpontok védelméhez az `[Authorize]` attribútumot használjuk. Ez az attribútum biztosítja, hogy csak azok a kérések legyenek

elfogadva, amelyek érvényes hozzáférési tokent tartalmaznak. A tokenekben található scope-ok alapján további jogosultsági ellenőrzéseket is végezhetünk, hogy biztosítsuk, hogy az importer szolgáltatások csak azokhoz a műveletekhez férjenek hozzá, amelyekre jogosultak.

Token Biztonsága

A tokenek biztonságos kezelése érdekében a következő intézkedéseket alkalmaztuk:

- **Token élettartam**: A tokenek élettartamát korlátoztuk, hogy minimalizáljuk a tokenek visszaélése esetén fellépő kockázatokat. Az Auth0-ban beállítottuk a tokenek élettartamát (pl. 1 óra), így a tokenek rövid időn belül lejárnak, és új tokeneket kell kérniük az importer szolgáltatásoknak.
- **HTTPS használata**: Minden kommunikáció HTTPS protokollon keresztül történik, hogy biztosítsuk az adatok titkosságát és integritását. Ez különösen fontos a tokenek továbbítása során, mivel így megelőzzük a tokenek ellopását vagy visszaélését.
- **Token titkosítás**: A tokenek titkosítva kerülnek továbbításra. Az Auth0 által kiadott tokenek JWT (JSON Web Token) formátumban kerülnek elküldésre, amelyek digitálisan aláírtak. Ez biztosítja, hogy a tokenek tartalma ne legyen módosítható, és csak az Auth0 által kiadott tokenek legyenek érvényesek.
- **Scope korlátozás**: Az importer szolgáltatások csak azokhoz a scope-okhoz férnek hozzá, amelyekre szükségük van. Ez minimalizálja a tokenek visszaélésének kockázatát, mivel az importer szolgáltatások nem férnek hozzá olyan műveletekhez, amelyekre nincs jogosultságuk.

Összegzés

Az Auth0 Machine-to-Machine (M2M) autentikáció bevezetése lehetővé tette, hogy az importer szolgáltatások biztonságosan kommunikáljanak a backend API-val. A .NET keretrendszer beépített JWT hitelesítési mechanizmusa és az Auth0 által biztosított token kezelés segítségével biztosítottuk a rendszer biztonságát és skálázhatóságát. A tokenek biztonságos kezelése, a scope-ok korlátozása és a HTTPS használata további rétegeket adtak a rendszer védelmének, így az alkalmazás magas szintű biztonságot nyújt mind az adatok, mind a felhasználók számára.

5.4. Adatbázis

A projektben a **PostgreSQL** [Pos24] relációs adatbázist használjuk, amely egy nyílt forráskódú, hatékony és skálázható adatbázis-kezelő rendszer. A PostgreSQL kiváló választásnak bizonyult a projekt számára, mivel támogatja a komplex lekérdezéseket, tranzakciókat, és rugalmasan kezelhető a sémák és adattípusok terén. Az adatbázis tervezésénél és kezelésénél a **Code-First** megközelítést alkalmaztuk, amely lehetővé teszi, hogy az adatbázis sémáját közvetlenül a C# osztályokból generáljuk.

5.4.1. Code-First megközelítés

A Code-First megközelítés során az adatbázis sémáját a C# osztályok (entitások) alapján hozzuk létre. Ez a módszer lehetővé teszi, hogy a fejlesztők az alkalmazás logikájára koncentráljanak, miközben az adatbázis sémája automatikusan generálódik az osztályok alapján. A Code-First megközelítés főbb előnyei:

- **Gyors fejlesztés**: Az adatbázis sémája automatikusan generálódik, így nincs szükség manuális SQL szkriptek írására.
- **Könnyű karbantarthatóság**: Az adatbázis változtatásai közvetlenül a C# osztályok módosításával végezhetők el.
- **Verziókövetés**: Az adatbázis sémája verziókövetéssel nyomon követhető, mivel az entitások a forráskód részei.
- **Platformfüggetlenség**: A Code-First megközelítés lehetővé teszi, hogy az adatbázis sémája bármilyen támogatott adatbázisrendszeren létrejöjjön, beleértve a PostgreSQL-t is.

A Code-First megközelítés során az adatbázis migrációkat használjuk a sémaváltoztatások kezelésére. A migrációk lehetővé teszik, hogy az adatbázis sémája fokozatosan fejlődjön az alkalmazás változásaival együtt, anélkül, hogy az adatok elvesznének vagy sérülnének.

5.4.2. ORM (Object-Relational Mapping)

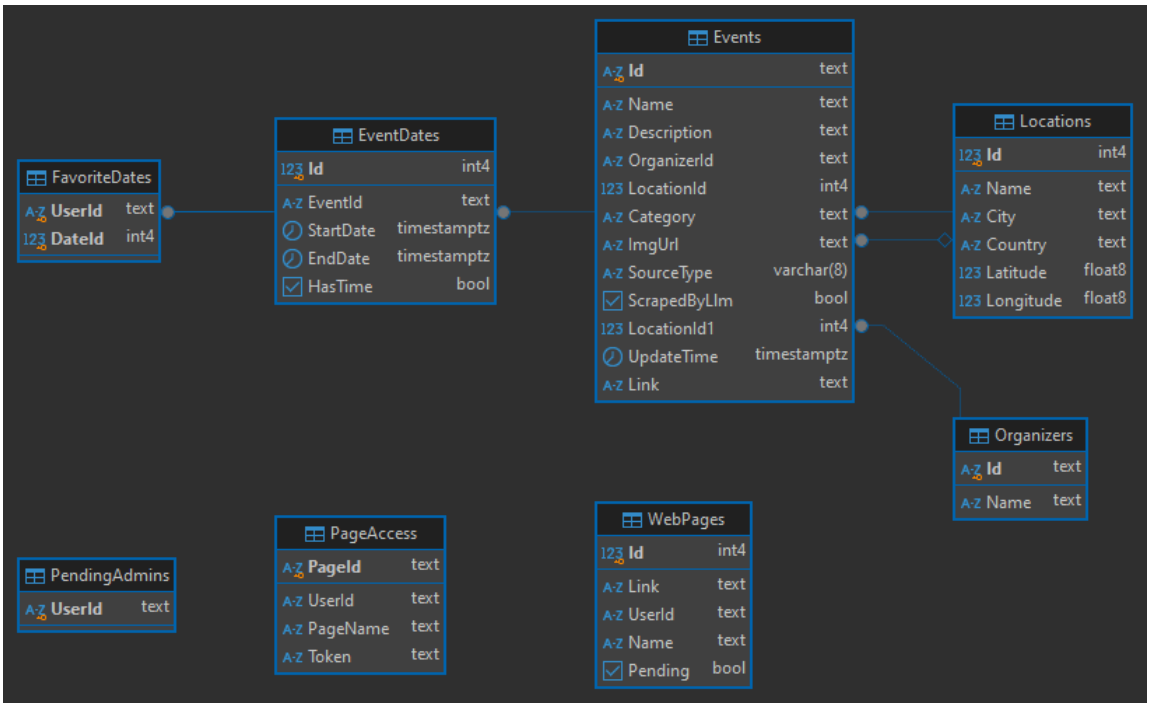
Az adatbázis kezeléséhez a .NET keretrendszer beépített ORM (Object-Relational Mapping) eszközét, az **Entity Framework Core**-t használjuk [Mic24a]. Az Entity Framework Core egy hatékony és rugalmas ORM, amely lehetővé teszi, hogy a fejlesztők objektum-orientált módon dolgozzanak az adatbázissal, anélkül, hogy közvetlenül SQL lekérdezéseket kellene írniuk. Az Entity Framework Core főbb jellemzői és előnyei:

- **Entitások és adatbázis táblák közötti leképezés**: Az Entity Framework Core automatikusan leképezi a C# osztályokat (entitásokat) az adatbázis táblákra, és fordítva.
- **LINQ támogatás**: A LINQ (Language Integrated Query) segítségével a fejlesztők közvetlenül C# nyelven írhatnak lekérdezéseket, amelyeket az Entity Framework Core SQL lekérdezésekké alakít.
- **Migrációk**: Az Entity Framework Core támogatja az adatbázis migrációkat, amelyek lehetővé teszik a sémaváltoztatások könnyű kezelését és nyomon követését.
- **Tranzakciók és konkurencia kezelése**: Az Entity Framework Core beépített támogatást nyújt a tranzakciók kezelésére és a konkurencia problémák megelőzésére.
- **Adatbázis függetlenség**: Az Entity Framework Core több adatbázisrendszert támogat, beleértve a PostgreSQL-t, így a kód átvihető más adatbázisokra is.

5.4.3. Adatbázis tervezés

Az adatbázis tervezésénél a következő alapelveket követtük:

- **Normalizálás**: Az adatbázis sémáját normalizáltuk, hogy elkerüljük az adatredundanciát és biztosítsuk az adatok konzisztenciáját.
- **Kapcsolatok**: Az entitások közötti kapcsolatokat (pl. egy-egy, egy-több, több-több) az Entity Framework Core segítségével definiáltuk, hogy az adatok közötti kapcsolatok jól reprezentálva legyenek.
- **Indexelés**: Az adatbázisban indexeket hoztunk létre a gyakran használt lekérdezések teljesítményének javítása érdekében.
- **Adatbiztonság**: Az adatbázis hozzáférését szigorúan korlátoztuk, és a bizalmas adatokat titkosítottuk.



5.6. ábra. Az adatbázis sémája és a táblák közötti kapcsolatok

5.4.4. Távolságok kiszámítása a Haversine formulával

Az alkalmazásban az események közötti földrajzi távolságok meghatározásához a **Haversine formulát** alkalmazzuk [Sin84]. Ez a módszer pontos eredményt nyújt a Föld görbületét figyelembe véve, amikor két pont távolságát számítjuk ki szélességi és hosszúsági koordináták alapján.

A formula matematikai alakja:

$$a = \sin\check{s}\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin\check{s}\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = R \cdot c$$

ahol:

- ϕ_1 és ϕ_2 a két pont szélességi koordinátái (radiánban),
- $\Delta\phi$ és $\Delta\lambda$ a koordináták különbségei,
- R a Föld sugara (átlagosan 6371 km),
- d a két pont közötti távolság.

A számításokat a backend szolgáltatási rétegében valósítjuk meg, amely lehetővé teszi:

- Események szűrését felhasználó által megadott hely és sugár alapján,
- Közelben lévő események hatékony azonosítását,
- Távolság alapú rendezési lehetőségeket.

A módszer különösen hatékony nagy adatmennyiségek esetén.

5.4.5. Összegzés

A PostgreSQL relációs adatbázis és az Entity Framework Core ORM együttes használata lehetővé teszi, hogy az adatbázis kezelése hatékony, rugalmas és könnyen karbantartható legyen. A Code-First megközelítés és a migrációk segítségével az adatbázis sémája folyamatosan fejlődhet az alkalmazás változásaival együtt, miközben az adatok integritása és biztonsága biztosított marad. Az adatbázis végleges formája az 5.6 ábrán látható, amely szemlélteti a táblák szerkezetét és a köztük lévő kapcsolatokat.

5.5. Importer

Az EventHub folyamatosan naprakész és pontos információszolgáltatása érdekében többféle adatgyűjtési módszert alkalmazunk.

- **Facebook Graph API:** Az első adatgyűjtési módszerünk a Facebook Graph API [Fac24a], amely lehetővé teszi, hogy közvetlenül importáljuk az eseményeket a Facebook oldalakról. Ez a módszer hatékonyan és könnyen hozzáférhetővé teszi az adott oldalak eseményeit, biztosítva a gyakori információfrissítést.
- **Puppeteer Scraper:** A második módszerünk a Puppeteer Scraper [Pup24] technológiára épül, amely a weboldalak HTML struktúrája alapján gyűjti össze az adatokat. Ez a megközelítés lehetővé teszi, hogy az oldalak tartalmát kinyerjük, így biztosítva az események részletes és pontos megjelenítését, de minden egyes weboldalra manuálisan kell beállítanunk azok struktúráját.

Ez a komponens TypeScript-ben íródott, ami segít a hibák felismerésében és a kód karbantartását könnyebbé teszi. Emellett még használatba vettük az ESLin-t és Prettier eszközöket, hogy egységesítsük a stílust és a potenciális hibákat kiszűrjük.

5.5.1. Importer részei

Main és naplózás

A main fájl az események begyűjtéséért és mentéséért felelős, két fő műveletet kezelve:

- Puppeteer által gyűjtött események
- Facebook importált események

Párhuzamosan végrehajtja mindkét műveletet, majd elküldi az eseményeket a backendnek. Ahhoz, hogy a folyamat csak naponta egyszer fusson le, időzítjük egyetlen időpontra. Emellett a *pino* logger eszközzel naplózza a folyamatokat (lásd 5.6. kódrészlet).

Scrapers

Ebben a részben vannak megvalósítva a Puppeterrel írt scraperek a különböző eseménnyel kapcsolatos weboldalakra (Például: Tomcsa Sándor Színház).

5.6. kódrészlet. Importer main függvény

```
async function main() {
  try {
    const [tomcsaTheaterEvents, facebookEvents] = await Promise.all([
      scrapeTomcsaTheater(),
      importFacebookEvents(),
    ]);

    const eventManagerService = new EventManagerService();
    await eventManagerService.saveFacebookEvents(facebookEvents);

    await eventManagerService.saveScrapedEvents(tomcsaTheaterEvents);
  } catch (error) {
    logger.error('Error during main execution:', error);
  }
}
```

5.7. kódrészlet. Scraper interface

```
export interface Scraper {
    initScraper(): Promise<void>;
    scrapePeriod(startDate: Date, endDate: Date): Promise<void>;
    getScrapedEvents(): Event[];
    closeScraper(): Promise<void>;
}
```

Példa a székelyudvarhelyi Tomcsa Sándor Színház weboldalának scrapeléséről: Az importer először elindít egy böngészőt, majd adott időszakra végigmegy az eseményeket listázó oldalakon, és begyűjti az események nevét, időpontját, helyszínét, képét és linkjét. Ha egy esemény részleteit is meg kell szerezni, akkor megnyitja az adott esemény oldalát, és kiolvassa a leírást. Az összegyűjtött adatokat elmenti, majd bezárja a böngészőt.

A megközelítés hátránya, hogy minden egyes oldalra külön scrapert kell írni a megfelelő működés és az adatok pontossága érdekében.

Importers

Jelenleg egyetlen importerünk van a Facebook Importer, amely a Facebook Graph API segítségével gyűjti be az eseményeket. Ez az importer Facebookról gyűjt eseményeket a megadott access tokenek segítségével (lásd 5.8. kódrészlet). Az eseményeket egy adott időponttól kezdve kéri le, majd az adatokat egy egységes formátumba alakítja. Ha egy esemény nem online, akkor normalizálja a városnevet, és ha nincsenek koordinátái, a város földrajzi adatait is hozzá rendeli. A lekérések között egy másodperces várakozás van, hogy elkerüljük a túl sok kérés miatti korlátozásokat. Utolsó lépésként pedig az összes eseményt összevonja.

5.8. kódrészlet. Facebook impotert események lekérése

```
export class FacebookImporter implements Importer {
  private async fetchEventsWithToken(accessToken: string): Promise<Event[]> {
    const currentTime = new Date().toISOString();
    const queryParams: QueryParams = {
        fields :
          'name,start_time,end_time,description,place,cover,
              category,updated_time,is_online,event_times',
        since: currentTime,
        access_token: accessToken,
    };

    const eventsUrl = `${urls.facebook}/me/events/${buildQueryString(queryParams)}`;
```

Mappers

Az *EventMapper* osztály felelős azért, hogy a Facebook API-ból származó eseményadatokat egy általánosabb Event modellformátumba alakítja, hogy a backend API, jól tudja értelmezi az adatokat. Az esemény helyszínének meghatározása során először ellenőrzi, hogy az esemény online-e, ebben az esetben a város mező „Online" lesz, és nem rendel hozzá földrajzi koordinátákat. Ha az esemény nem online, akkor a város és az ország értékét a place objektumból vagy a helyszín nevéből próbálja meghatározni, ha pedig egyik sem áll rendelkezésre, akkor a helyszín neve „Unknown" lesz. A kezdő- és befejezési időpontokat az API válaszából nyeri ki és formázza őket ISO formátumra. Ha az eseménynek több időpontja van mindegyik időponthoz külön bejegyzést készít. Az esemény kategóriája egy előre meghatározott kategória mapper segítségével egy általánosabb kategóriarendszerbe kerül leképezésre, ha nincs megfelelő kategória, akkor az eredeti Facebook kategória marad. Az eseményhez tartozó egyéb mezők, például a borítókép URL-je, az esemény linkje, a frissítési időpont és a szervező neve is beállításra kerülnek. Az így elkészült Event objektum normalizált formában tartalmazza az esemény adatait.

Utils

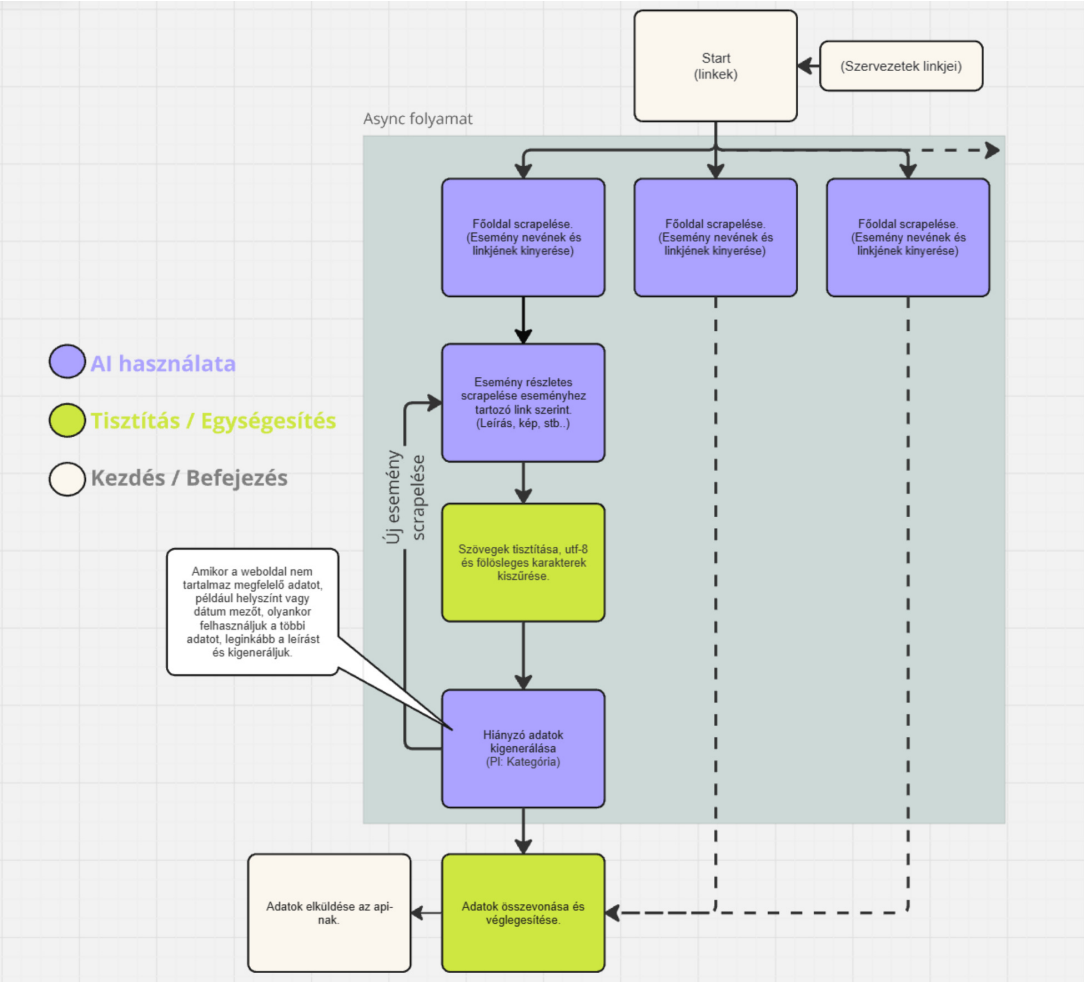
Használunk egy *buildQueryString* függvényt, amely objektumból URL-lekérdezési stringet generál.

Service

- **CityNameService:** Egy városnév normalizáló szolgáltatást valósít meg. Ha a bemenet üres vagy „Unknown", visszaadja ugyanazt az értéket. Egyébként lekérdezi az OpenStreetMap API-t [Ope25], hogy pontosabb városnevet és koordinátákat szerezzen. Ha talál egyezést, visszaadja a normalizált nevet és a földrajzi koordinátákat. Hibás válasz vagy hiba esetén az eredeti városnevet adja vissza.
- **EventManagerService:** Ez a rész elküldi az eseményeket a backendnek, külön a Facebookos és weboldalról nyert eseményeket.
- **PageAccessService:** A backendtől megkapjuk a Facebook oldalakhoz tartozó, access tokeneket.

5.6. AI-Scraper

AI Alapú Adatgyűjtő Eszköz: A harmadik módszerünk az adatgyűjtésre egy mesterséges intelligencián alapuló webes adatgyűjtő eszköz, amely automatikusan képes felfedezni és scrape-elni weboldalakat anélkül, hogy előre konfigurálnánk az oldalspecifikus beállításokat. Ez a Pythonban íródott AI scraper intelligensen alkalmazkodik a különböző weboldalak szerkezetéhez, így hatékonyan és precízen gyűjti össze a szükséges adatokat. Az adatgyűjtési folyamatot az open-source Large Language Model (LLM) alapú crawl4ai[Cra24] támogatja, melyet a Google Gemini nevű LLM-je egészít ki, amely az oldalak inteligens scrapelését segíti elő, illetve továbbá használjuk a Groq llama-3.3-70b-versatile LLM-jét is az adatok kiegészítéséhez.



5.7. ábra. AI Scraper diagram

A 5.7. ábra diagramja alapján az AI Scraper működésének részletes bemutatása a következő lépéseken keresztül szemlélteti, hogyan biztosítja folyamatosan az eseményeket.

1. **Indítás:** A folyamat azzal kezdődik, hogy a szervezetek linkjeit begyűjtjük a backendtől. Ezek a linkek szolgálnak a további scrape-elési műveletek alapjául.

2. **Főoldal scrape-elése (Esemény nevek és linkek összegyűjtése):** Első lépésként a főoldalról begyűjtjük a szükséges eseményekhez tartozó neveket és linkeket,

úgy hogy minden linken async módon futtatjuk le az adatgyűjtést. Ezek a linkek lehetnek eseményekhez vagy további aloldalakhoz vezető hivatkozások.

3. **Események részletes oldalainak scrape-elése:** Az előző lépésben gyűjtött esemény linkek és nevek alapján az események részletes oldalait scrape-eljük. Itt további adatokat szerzünk, mint például leírás, képek, és egyéb fontos információk.

4. **Szövegek tisztítása, karakterek kiszűrése:** Miután az adatokat összegyűjtöttük, következik a szövegek tisztítása. Az adatok UTF-8 kódolása történik, és a felesleges karakterek kiszűrése. Ez a lépés biztosítja, hogy az adatok megfelelő formátumban kerüljenek feldolgozásra.

5. **Adatok generálása:** Amikor az adott weboldal nem tartalmazza az összes szükséges adatot (pl. helyszín, dátum), a meglévő adatok alapján az AI generálja ki a hiányzó információkat. Ez lehet például egy dátum létrehozása a leírás alapján, ha benne található. Illetve geokódoljuk a nevet, hogy megkapjuk az esemény pontos földrajzi helyzetét, ezt a folyamatot a Google Maps platformmal végeztük el, mivel ez a szolgáltatás képes név alapján is keresni, nemcsak helyszíni cím alapján. Ezáltal sokkal pontosabb adatokat kapunk, mint más geokódoló szolgáltatások esetében, amelyek gyakran csak hely címekkel dolgoznak.

6. **Adatok összevonása és véglegesítése:** Miután nincs több eseményoldal link és minden szükséges adat rendelkezésre áll, azokat összevonjuk és véglegesítjük a rendszer számára.

7. **Adatok elküldése az API-nak:** Az utolsó lépés az adatok elküldése az API-nak, amely aztán gondoskodik az adatok továbbításáról és megjelenítéséről az EventHub weboldalon.

5.6.1. AI-scraper pontossága

Az adatok pontossága nagyon fontos szempont az alkalmazást illetően, hiszen nem tartalmazhatnak hamis információk az események.

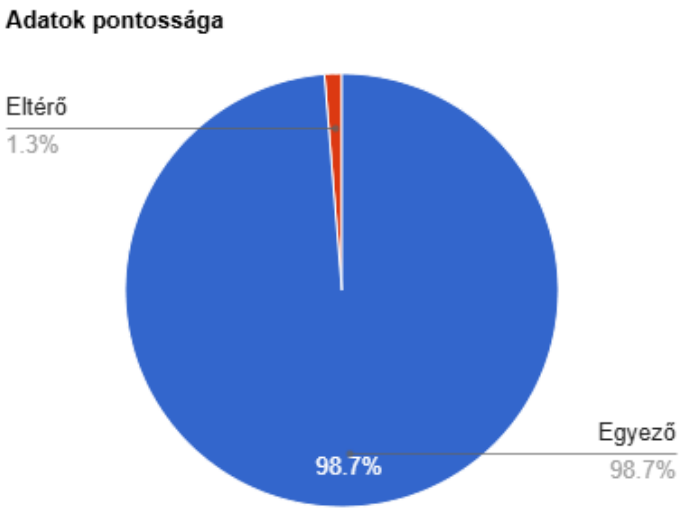
Különböző oldalak eseményeivel teszteltük az AI-scrapert, hogy mennyire pontos a működése. A teszteket úgy végeztük el, hogy lefuttattuk a scrapert, amelynek a végeredményét manuálisan ellenőriztük és vizsgáltuk át minden egyes eseményre.

Az eredmény 4 különböző weboldalra és 390 adatmezőre 98.72 százalékos pontosságot ért el (lásd 5.8. ábra).

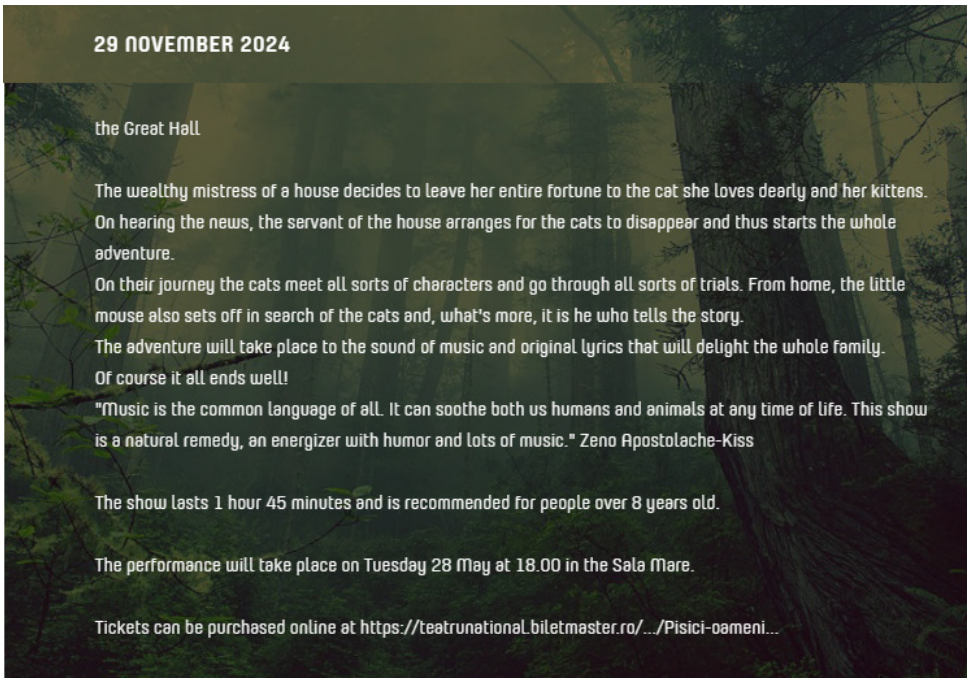
A hibát egyszerű emberi tévedések okozhatják, mint például egy elírás vagy több eltérő dátum használata egyetlen eseményhez (lásd 5.9. ábra). Az AI helyesen kiszámolta az esemény idejét és végét a leírás alapján (lásd 5.9. kódrészlet), de hiába ha az oldalon nem egyértelmű a hónap és a nap.

5.9. kódrészlet. AI-scraper által kinyert dátum

```
"start_date": "2024-05-28T15:00:00Z",
"end_date": "2028-05-28T16:45:00Z",
```



5.8. ábra. Adatok pontossága



5.9. ábra. Több eltérő kezdődátum

Egy másik emberi hiba lehet a helytelen fogalmazás, mint például egy eseményhez azt a dátumot adni hogy: „2025-s szilveszteri buli”, amely a 2025 év végi szilvesztert jelenti, nem a 2024 év végit, pedig ez az esemény a 2024 év végén volt kitéve.

5.6.2. AI-Scraper részei

Scraper

Egy aszinkron webes eseménygyűjtő modult valósít meg. Egy adott URL-ről gyűjti össze az eseményeket, majd részletes adatokat szerez az egyes eseményekhez. Az adatokat egy előre létrehozott és promptolt LLM-alapú stratégiával (lásd 5.10. ábra) kinyeri, majd a kapott tartalmat JSON formátumba alakítja és feldolgozza. Ezután további műveletekkel (pl. hiányzó adatok generálása, geokódolás, időpontfrissítés) teszi teljessé az eseményeket. Az eredmény egy egységes formátumú eseménylista.

5.10. kódrészlet. AI scraper kinyerési stratégia

```
main_llm_extraction_strategy = LLMExtractionStrategy(
    provider=f'{PROVIDER}/{SCRAPER_MODEL}',
    api_token=os.getenv('GEMINI_API_KEY'),
    schema=Event.model_json_schema(),
    extraction_type="schema",
    instruction=EXTRACTION_INSTRUCTION
)

sub_extraction_strategy = LLMExtractionStrategy(
    provider=f'{PROVIDER}/{SCRAPER_MODEL}',
    api_token=os.getenv('GEMINI_API_KEY'),
    schema=DetailPage.model_json_schema(),
    extraction_type="schema",
    instruction=LINK_INSTRUCTION
)
```

Geocoding

A geokódolás az események földrajzi helyének frissítéséért felel. A helyadatokat a Google Maps API segítségével határozza meg az esemény helyszíne alapján (ország, város, hely neve), majd frissíti az esemény szélességi és hosszúsági koordinátáit.

Ezen belül az időt is egységes időzónába alakítja, ezt a földrajzi koordináták alapján állapítja meg. Ha az időpont éjfélre esik (00:00:00), akkor az eredeti időpont marad, különben az időzónának megfelelő helyi időre alakítja, majd UTC-re konvertálja.

Generator

Az adatok generálása Groq API segítségével történik, amely segít kiegészíteni a hiányzó eseményadatokat. Szöveges promptokat használ, majd a válaszban kapott generált adatokat elmenti az adott eseményhez.

Generálható adatok:

- Kezdési és befejezési dátum.
- Helyszín neve, városa és országa.
- Esemény kategóriája.

Parser

Itt a JSON-re való átalakítás történik, amely során a bemeneti stringet alakítja át strukturált adattá.

Main

A main fájlban történik a scraper indítása és a backendel való komunikáció. Először megkapja a backendől a weboldalak URL-jét, amelyekről az adatot le kell szedni, a scrapertől megkapja az adatokat majd elküldi őket a backendnek. Mindez időzítve van, hogy milyen időpontokban fusson csak le (lásd 5.11. ábra).

5.11. kódrészlet. AI-scraper ütemezése

```
logger = logging.getLogger(__name__)

async def main():
    logger.info("Starting scraper...")
    all_events = []

    urls = get_urls()

    tasks = [extract_events_from_link(url) for url in urls]
    results = await asyncio.gather(*tasks)

    for events in results:
        all_events.extend(events)

    post_events(all_events)
    logger.info("Scraper finished.")

time_var = os.environ.get('AI_SCRAPER_SCHEDULE', '23:20')

schedule.every().day.at(time_var).do(lambda: asyncio.run(main()))

while True:
    schedule.run_pending()
    time.sleep(60)
```

Mindegyik importáló módszer JSON formátumban küldi el az adatokat az API-nak, amely berakja őket az adatbázisba.

5.7. Kitelepítés

A projekt szoftverfejlesztési folyamatának egyik legfontosabb része a kitelepítés (deployment), amely során a fejlesztett alkalmazásokat és szolgáltatásokat éles környezetbe helyezzük. A kitelepítési folyamat során a GitLab CI/CD (Continuous Integration/Continuous Deployment) rendszert használjuk [Git24a], amely lehetővé teszi a folyamatok automatizálását és a kód minőségének folyamatos ellenőrzését. A következőkben részletesen bemutatjuk a kitelepítési folyamatot, a használt technológiákat és az architektúra részleteit.

5.7.1. GitLab CI Pipeline

A GitLab CI/CD (Continuous Integration/Continuous Deployment) egy olyan eszköz, amely lehetővé teszi a szoftverfejlesztési folyamatok teljes körű automatizálását. A folyamatot a `.gitlab-ci.yml` fájlban definiáljuk, amelyben különböző szakaszokra (stages) osztjuk a folyamatot. A fő szakaszok a következők:

- **Preparation (Előkészítés)**: A folyamat első lépése, ahol a szükséges környezeti változók és függőségek beállításra kerülnek. Ebben a szakaszban a rendszer ellenőrzi, hogy a kód minden szükséges függőséggel rendelkezik-e, és hogy a környezeti változók (például adatbázis kapcsolati sztringek, API kulcsok) helyesen vannak-e konfigurálva. A `variables` blockban definiált változók globálisan elérhetőek a folyamat során, és a különböző feladatokban felhasználhatók.
- **Build (Fordítás)**: A forráskód átalakítása futtatható konténerképekké. A build folyamat során a rendszer:
  - Létrehoz egy Docker image-t a forráskódból a `Dockerfile` definíció alapján
  - Az image-eket a GitLab Container Registry-ben tárolja verziózottan
  - Automatikusan verziócímkézést alkalmaz (pl. commit hash, dátum alapján)
  - Többfázisú buildet használ a végső image méretének minimalizálásához

  A build folyamat során a rendszer ellenőrzi:
  - A `Dockerfile` szintaxisának helyességét
  - Az image-ek sebezhetőségeit (vulnerability scanning)
  - A függőségek licenceinek megfelelőségét

  A sikeres build eredménye minden esetben standard Docker/OCI image formátumban áll elő, amely bármely konténer-orchestrációs platformon futtatható.
- **Tests (Tesztek)**: Az alkalmazás automatikus tesztelése, amely magában foglalja az egységteszteket, integrációs teszteket és egyéb ellenőrzéseket. A tesztek során a rendszer ellenőrzi, hogy az alkalmazás minden funkciója helyesen működik-e, és hogy nincsenek-e regressziós hibák. A tesztek eredményeit a rendszer naplózza, és ha valamelyik teszt sikertelen, a folyamat leáll, és a hibák kijavítása után újra kell indítani a folyamatot. A tesztek során a rendszer különböző környezetekben is futtatja a kódot, hogy biztosítsa a kompatibilitást és a stabilitást.

- **Deploy (Kitelepítés)**: Az alkalmazás éles környezetbe történő telepítése. A telepítés során a rendszer a korábban létrehozott container image-eket használja, és a Docker konténerek segítségével telepíti az alkalmazást. A telepítés során a rendszer ellenőrzi, hogy az alkalmazás helyesen indul-e el, és hogy az összes szükséges szolgáltatás (például adatbázis, API-k) elérhető-e. A telepítés során a rendszer a `docker-compose.yml` fájlban definiált szolgáltatásokat indítja el, és ellenőrzi, hogy azok helyesen működnek-e.

A folyamatot a GitLab Runner végzi [Git24b], amely egy olyan eszköz, amely a CI/CD feladatokat végrehajtja. A Runner lehet közös (shared) vagy specifikus (specific), és Docker konténerben fut, hogy biztosítsa az izolációt és a konzisztenciát. A Runner a `gitlab-ci.yml` fájlban definiált feladatokat hajtja végre, és a folyamat során létrehozott kimenetek (például bináris fájlokat, naplókat) a GitLab rendszerében tárolja. A Runner konfigurációja a `.gitlab-ci.yml` fájlban történik, ahol megadhatjuk, hogy milyen környezetben fusson a folyamat, és milyen feltételek alapján hajtsa végre a feladatokat.

A GitLab CI/CD folyamat során a következő kulcsfontosságú elemeket használjuk:

- **Stages (Szakaszok)**: A folyamatot különböző szakaszokra osztjuk, amelyek sorrendben futnak le. Minden szakaszban egy vagy több feladat (job) futhat, és a szakaszok sorrendje meghatározza a folyamat logikáját. Például a `build` szakasz csak akkor fut le, ha a `preparation` szakasz sikeresen befejeződött.

- **Jobs (Feladatok)**: Minden szakaszban egy vagy több feladat fut, amelyek a folyamat egy-egy lépését hajtják végre. Például a `build` szakaszban a `dotnet build` parancsot futtató feladat fordítja le a kódot. A feladatok során a rendszer naplózza a folyamatot, és ha valamelyik feladat sikertelen, a folyamat leáll, és a hibák kijavítása után újra kell indítani a folyamatot.

- **Termékfájlok**: A folyamatt során létrehozott bináris fájlokat, naplókat és egyéb kimeneteket a rendszer tárolja, hogy a későbbi szakaszokban felhasználhassuk őket. Például a `build` szakaszban létrehozott bináris fájlokat a `deploy` szakaszban használjuk fel az alkalmazás telepítéséhez.

- **Rules (Szabályok)**: A feladatok futtatásának feltételeit a `rules` szakaszban definiáljuk. Például a `build` feladat csak akkor fut le, ha a `CI_COMMIT_REF_NAME` változó értéke nem `develop`. Ez lehetővé teszi, hogy a folyamatot csak bizonyos feltételek teljesülése esetén hajtsuk végre, például csak a fő ágon vagy egy adott környezetben.

A GitLab CI/CD folyamat során a rendszer folyamatosan ellenőrzi a kód minőségét, és a sikeres tesztelés után az alkalmazás éles környezetbe kerül telepítésre. A folyamat során a rendszer naplózza a folyamatot, és a hibák esetén értesítést küld a fejlesztőknek, hogy a hibákat minél hamarabb kijavíthassák. A GitLab CI/CD folyamat lehetővé teszi a szoftverfejlesztési folyamatok hatékony automatizálását, és biztosítja, hogy a kód mindig stabil és készen áll a telepítésre.

5.7.2. Kitelepítési Folyamat

A kitelepítési folyamat során a következő lépéseket hajtjuk végre:

1. **Forráskód kezelés és verziókövetés**: A kódot a GitLab verziókövető rendszerében kezeljük, ahol minden változtatás követhető és ellenőrizhető. A változtatások a `git push` parancs segítségével kerülnek feltöltésre a tárolóba.

2. **Automatizált tesztelés**: Minden változtatás automatikus tesztelési folyamaton megy keresztül, amely magában foglalja az egységteszteket és az integrációs teszteket. A tesztek sikeres lefutása után a kód továbbhalad a következő szakaszba.

3. **Build folyamat**: A sikeres tesztelés után a kód fordításra kerül, és a szükséges bináris fájlok előállításra kerülnek. A fordítást a `dotnet build` parancs végzi, amely a `Release` konfigurációban fordítja le a kódot.

4. **Kitelepítés automatizált folyamata**: A verzió közvetlenül az éles környezetbe kerül telepítésre teljesen automatizált folyamat keretében. A telepítést Docker konténerek segítségével valósítjuk meg, amely biztosítja:

   - Az alkalmazás izolált futását
   - Skálázhatóságot
   - Konzisztens működést minden környezetben

5.7.3. Docker és Docker Compose

A projektben a Docker és a Docker Compose technológiákat használjuk az alkalmazások konténerizálására és kezelésére. Ezek a technológiák lehetővé teszik, hogy az alkalmazásokat és azok függőségeit elkülönített, könnyen reprodukálható környezetben futtassuk. A Docker konténerek segítségével biztosítható, hogy az alkalmazás minden környezetben (fejlesztői gépen, tesztkörnyezetben, éles környezetben) ugyanúgy működik, miközben a Docker Compose segítségével több konténer egyszerre kezelhető, és azok közötti kapcsolatokat is könnyedén definiálhatjuk.

Docker

A Docker egy nyílt forráskódú platform, amely lehetővé teszi alkalmazások konténerizálását. A konténerek olyan könnyű, hordozható és önálló egységek, amelyek tartalmaznak mindent, ami az alkalmazás futtatásához szükséges: kódot, futási időt, rendszerkönyvtárakat és beállításokat. A Docker konténerek izoláltak egymástól és a gazdagéptől, de ugyanazon az operációs rendszeren futnak, így hatékonyabban használják a rendszererőforrásokat, mint a hagyományos virtuális gépek.

A Docker működése a következő főbb lépésekből áll:

- **Dockerfile**: A Dockerfile egy szöveges konfigurációs fájl, amely meghatározza a Docker konténer összeállításának lépéseit. Tartalmazza:
  - Az alaprendszerkép kiválasztását (`FROM` utasítás)

  - A szükséges függőségek telepítését (`RUN`, `COPY` utasítások)
  - Az alkalmazás indítási parancsát (`CMD` vagy `ENTRYPOINT`)

- **Rétegek (Layers)**: Minden Dockerfile utasítás új réteget hoz létre:
  - Minden réteg csak az előzőhöz képest történt változtatásokat tárolja
  - A rétegek immutábilisak és gyorsítótárazva vannak
  - Optimalizálás: gyakran változó utasításokat (pl. `COPY`) a Dockerfile végére érdemes helyezni
  - Többszintű build (*multi-stage build*) segítségével csökkenthető a végső image mérete

- **Image létrehozása**: A Dockerfile feldolgozása során a rendszer:
  - Létrehozza a rétegeket a definiált utasítások alapján -et készít ezekből a rétegekből
  - Ez az image szolgál sablonként a konténerek példányosításához

- **Docker Image**: A Docker image egy csak olvasható, többrétegű (layered) csomagolási egység [Doc24], amely tartalmazza:
  - Az alkalmazás futtatásához szükséges kódot
  - Függőségeket és könyvtárakat
  - Rendszerkonfigurációkat
  - Futtatókörnyezetet

  Az image-ek forrása:
  - **GitLab Container Registry**: Privát tárolónk, ahová a CI/CD folyamat automatikusan feltölti a buildelt image-eket
  - **Docker Hub**: Nyilvános repository közös image-ekhez (pl. alaprendszerképek)

  Verziókezelés:
  - Minden image egyedi címkével (tag) ellátva
  - Verziószámozás a `git commit` hash alapján
  - `latest` tag a legutóbbi sikeres buildhez

- **Docker Container**: A Docker container egy futó példánya a Docker image-nek. A konténer egy izolált környezetben fut, és tartalmazza az alkalmazás futtatásához szükséges összes erőforrást. A konténerek könnyedén indíthatók, leállíthatók és törölhetők, és több konténer is futhat egyszerre ugyanazon a gépen, anélkül, hogy egymás működését befolyásolnák.

A Docker főbb parancsai:

- `docker build -t <image_name> .`: Létrehoz egy Docker image-et a Dockerfile alapján.
- `docker run <image_name>`: Elindít egy konténert a megadott image-ből.
- `docker ps`: Kilistázza az aktuálisan futó konténereket.
- `docker stop <container_id>`: Leállít egy futó konténert.
- `docker rm <container_id>`: Töröl egy konténert.

Docker Compose

A Docker Compose egy olyan eszköz, amely lehetővé teszi több Docker konténer egyszerre történő kezelését és azok közötti kapcsolatok definiálását. A Docker Compose segítségével egyetlen konfigurációs fájlban (`docker-compose.yml`) definiálhatjuk az összes szükséges szolgáltatást, és azok kapcsolatait. Ez különösen hasznos olyan alkalmazások esetében, ahol több szolgáltatás (például webalkalmazás, adatbázis, üzenetsor) együttműködésre van szükség.

A Docker Compose működése a következő főbb lépésekből áll:

- **Szolgáltatások definiálása**: A `docker-compose.yml` fájlban definiáljuk az összes szükséges szolgáltatást, és azok konfigurációját. Minden szolgáltatáshoz megadhatjuk, hogy milyen image-et használjon, milyen portokon fusson, milyen környezeti változókat használjon, és hogyan kapcsolódjon más szolgáltatásokhoz.
- **Hálózati kapcsolatok**: A Docker Compose automatikusan hoz létre hálózatokat a szolgáltatások között, így azok kommunikálhatnak egymással. A hálózati kapcsolatokat a `networks` szakaszban definiálhatjuk, és a szolgáltatások között a szolgáltatás nevével lehet hivatkozni (például `db` szolgáltatás elérése a `webapp` szolgáltatásból).
- **Környezeti változók**: A `environment` szakaszban definiálhatjuk a szolgáltatások által használt környezeti változókat. Ezek a változók lehetnek statikus értékek, vagy akár külső fájlokból is betölthetők (például `.env` fájl).
- **Konténerek indítása és leállítása**: A Docker Compose segítségével egyetlen paranccsal indíthatjuk és leállíthatjuk az összes szolgáltatást. A `docker-compose up` parancs elindítja az összes szolgáltatást, míg a `docker-compose down` parancs leállítja és törli azokat.

A Docker Compose főbb parancsai:

- `docker-compose up`: Elindítja az összes szolgáltatást a `docker-compose.yml` fájlban definiálva.
- `docker-compose down`: Leállítja és törli az összes szolgáltatást.
- `docker-compose logs`: Megjeleníti a szolgáltatások naplóit.
- `docker-compose ps`: Kilistázza az aktuálisan futó szolgáltatásokat.

Példa Docker Compose konfiguráció

A projektben a következő `docker-compose.yml` fájlt használjuk:

5.12. kódrészlet. Docker Compose konfiguráció

```
services:
  eventhub-web:
    image: r.edu.codespring.ro/esemenykereso/ui:latest
    environment:
      AUTH0_CLIENT_ID: ${AUTH0_CLIENT_ID}
      AUTH0_CLIENT_SECRET: ${AUTH0_CLIENT_SECRET}
      AUTH0_ISSUER: ${AUTH0_ISSUER}
      AUTH0_DOMAIN: ${AUTH0_DOMAIN}
      AUTH_SECRET: ${AUTH_SECRET}
      AUTH_TRUST_HOST: ${AUTH_TRUST_HOST}
      NEXTAUTH_URL: "https://eventhub.test.edu.codespring.ro"
      NEXT_PUBLIC_FACEBOOK_APP_ID: ${NEXT_PUBLIC_FACEBOOK_APP_ID}
      FACEBOOK_APP_SECRET: ${FACEBOOK_APP_SECRET}
      NEXT_PUBLIC_FACEBOOK_REDIRECT_URI:
          ${NEXT_PUBLIC_FACEBOOK_REDIRECT_URI}
      NEXT_PUBLIC_BACKEND_BASE_URL: "https://api.eventhub.test.edu.codespring.ro/"
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.eventhub_web.rule=Host(`eventhub.test.edu.codespring.ro`)"
      - "traefik.http.routers.eventhub_web.service=eventhub_web"
      - "traefik.http.routers.eventhub_web.entrypoints=https"
      - "traefik.http.services.eventhub_web.loadbalancer.server.port=3000"
    networks:
      - proxy
    restart: unless-stopped

  eventhub-importer:
    image: r.edu.codespring.ro/esemenykereso/importer:latest
    environment:
      IMPORT_SCHEDULE: "*/10 * * * *"
      BACKEND_BASE_URL: ${BACKEND_BASE_URL}
      IMPORTER_AUTH0_AUDIENCE: ${IMPORTER_AUTH0_AUDIENCE}
      IMPORTER_AUTH0_CLIENT_ID: ${IMPORTER_AUTH0_CLIENT_ID}
      IMPORTER_AUTH0_CLIENT_SECRET: ${IMPORTER_AUTH0_CLIENT_SECRET}
    networks:
      - proxy
    restart: no

  eventhub-database:
    image: postgres:latest
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    networks:
      - proxy
    restart: unless-stopped

  eventhub-api:
    image: r.edu.codespring.ro/esemenykereso/api:latest
```

```
    depends_on:
      - eventhub-database
    environment:
      ASPNETCORE_ENVIRONMENT: Development
      CONNECTION_STRING: ${CONNECTION_STRING}
      ALLOWED_ORIGINS: https://eventhub.test.edu.codespring.ro
      AUTH0_AUDIENCE: ${AUTH0_AUDIENCE}
      AUTH0_DOMAIN: ${AUTH0_DOMAIN}
      AUTH0_ISSUER: ${AUTH0_ISSUER}
      DATABASE_HOST: ${DATABASE_HOST}
      AUTH0_CLIENT_ID: ${AUTH0_CLIENT_ID}
      AUTH0_CLIENT_SECRET: ${AUTH0_CLIENT_SECRET}

    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.eventhub_api.rule=Host(`api.eventhub.test.edu.codespring.ro`)"
      - "traefik.http.routers.eventhub_api.service=eventhub_api"
      - "traefik.http.routers.eventhub_api.entrypoints=https"
      - "traefik.http.services.eventhub_api.loadbalancer.server.port=5140"
    networks:
      - proxy
    restart: unless-stopped

  eventhub-ai-importer:
    image: r.edu.codespring.ro/esemenykereso/ai-importer:latest
    environment:
      AI_SCRAPER_SCHEDULE: "14:00"
      IMPORTER_AUTH0_AUDIENCE: ${IMPORTER_AUTH0_AUDIENCE}
      IMPORTER_AUTH0_CLIENT_ID: $(IMPORTER_AUTH0_CLIENT_ID)
      IMPORTER_AUTH0_CLIENT_SECRET: ${IMPORTER_AUTH0_CLIENT_SECRET}
      BACKEND_BASE_URL: ${BACKEND_BASE_URL}
      GROQ_MODEL_NAME: ${GROQ_MODEL_NAME}
      GROQ_API_KEY: ${GROQ_API_KEY}
      GEMINI_PROVIDER: ${GEMINI_PROVIDER}
      GEMINI_MODEL: ${GEMINI_MODEL}
      GEMINI_API_KEY: ${GEMINI_API_KEY}
      GOOGLE_MAPS_API_KEY: ${GOOGLE_MAPS_API_KEY}
      IMPORTER_AUTH0_URL: ${IMPORTER_AUTH0_URL}
    networks:
      - proxy
    restart: no

networks:
  proxy:
    external: true
```

Ebben a konfigurációban két szolgáltatás van definiálva:

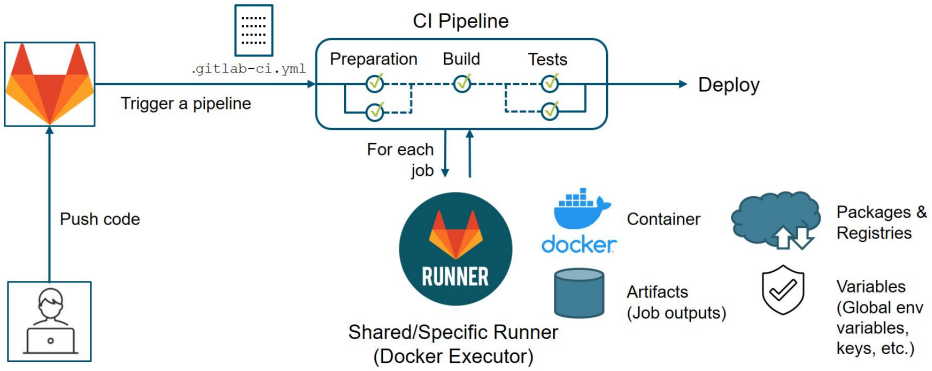
- **webapp**: Az alkalmazás fő komponense, amely a 5140-es porton fut. A szolgáltatás függ az adatbázis szolgáltatástól (`db`), és a `build: .` utasítás alapján a helyi Dockerfile-ból készít image-et. A szolgáltatás környezeti változókat használ, amelyeket a `environment` szakaszban definiálunk.
- **db**: Az adatbázis szolgáltatás, amely PostgreSQL-t használ, és a 5432-es porton érhető el. Az adatbázis környezeti változóit a `environment` szakaszban definiáljuk,

és a szolgáltatás a `mynetwork` hálózaton keresztül kommunikál a `webapp` szolgáltatással.

A Docker Compose segítségével az alkalmazás könnyedén indítható és leállítható, és a szolgáltatások közötti kapcsolatokat is könnyedén kezelhetjük. Ez lehetővé teszi, hogy az alkalmazás minden környezetben ugyanúgy működjön, és a fejlesztők ne kelljen külön konfigurálniuk az egyes szolgáltatásokat.

5.7.4. Kitelepítési Diagram

A kitelepítési folyamatot a következő diagram szemlélteti:



5.10. ábra. Kitelepítési Diagram

A diagramon látható, hogy a CI Pipeline során a kód először előkészítésre, majd fordításra és tesztelésre kerül. A sikeres tesztelés után a kód telepítésre kerül, és a Docker konténerek segítségével fut az éles környezetben.

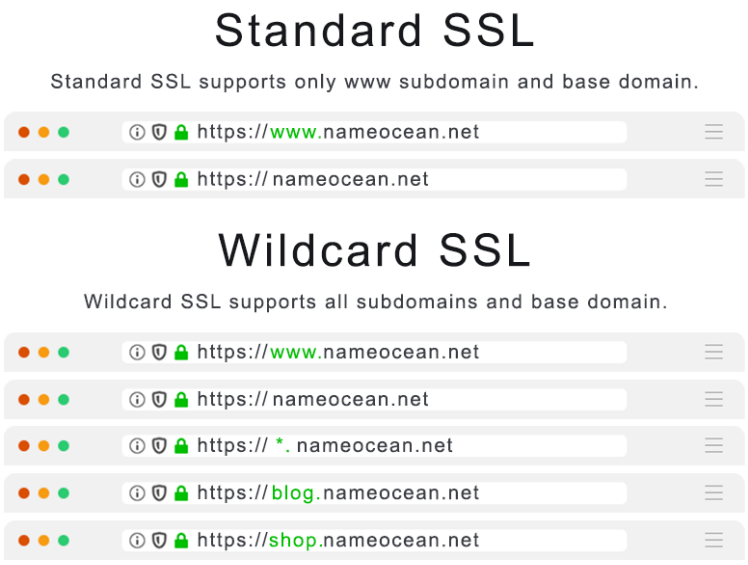
5.7.5. A weboldal elérhetősége

A weboldal a következő linken érhető el:
https://eventhub.test.edu.codespring.ro

Az alkalmazás által használt aldomain:
https://eventhub.test.edu.codespring.ro/hu/map

A rendszer biztonságos elérhetőségét egy wildcard tanúsítvány biztosítja. Ez egy speciális SSL/TLS tanúsítvány, amely egy domain és annak összes aldomainjére érvényes. Például a **.test.edu.codespring.ro* tanúsítvány érvényes a *eventhub.test.edu.codespring.ro* domainre és a hozzá tartozó */hu/main* aldomainre is.

A 5.11. ábrán látható az alap SSL és a wildcard SSL közötti különbség. Az alap SSL tanúsítvány csak egy konkrét domainre vonatkozik (például *www.nameocean.net*), míg a wildcard SSL az összes aldomainre kiterjed (például *blog.nameocean.net*, *shop.nameocean.net*).
A wildcard tanúsítvány előnyei:



5.11. ábra. Standard SSL és Wildcard SSL összehasonlítása

- **Biztonság**: Az összes forgalmat titkosítva továbbítja a domain és az aldomain között, biztosítva a biztonságos adatkapcsolatot.
- **Költséghatékonyság**: Egy wildcard tanúsítvány használata helyettesíti az egyedi tanúsítványok beszerzését minden aldomainhez.
- **Egyszerűség**: Új aldomain létrehozásakor a tanúsítvány automatikusan érvényes lesz további konfiguráció nélkül.

5.7.6. Összegzés

A kitelepítési folyamat során a GitLab CI/CD rendszert használjuk a szoftverfejlesztési folyamatok automatizálására. A folyamat a következő főbb lépésekből áll:

- **Forráskód kezelés és verziókövetés**: A kódot GitLab-ban tároljuk, ahol minden változtatás nyomon követhető.
- **Automatizált tesztelés**: Az alkalmazás automatikusan tesztelésen megy keresztül, beleértve az egységteszteket és integrációs teszteket.
- **Build folyamat**: A forráskód fordítása és a bináris fájlok előállítása a `dotnet build` paranccsal történik.
- **Kitelepítés**: Az alkalmazást Docker konténerek segítségével telepítjük éles környezetbe, miután az tesztkörnyezetben sikeresen ellenőrzésen ment keresztül.

A Docker és Docker Compose technológiák segítségével az alkalmazásokat konténerizált formában futtatjuk, ami biztosítja az izolációt és a skálázhatóságot. A Docker Compose segítségével több konténer egyszerre kezelhető, és azok közötti kapcsolatokat

is könnyedén definiálhatjuk. A rendszer biztonságos elérhetőségét egy wildcard SSL tanúsítvány biztosítja, amely költséghatékony és egyszerű megoldást kínál a domain és aldomainek védelmére.

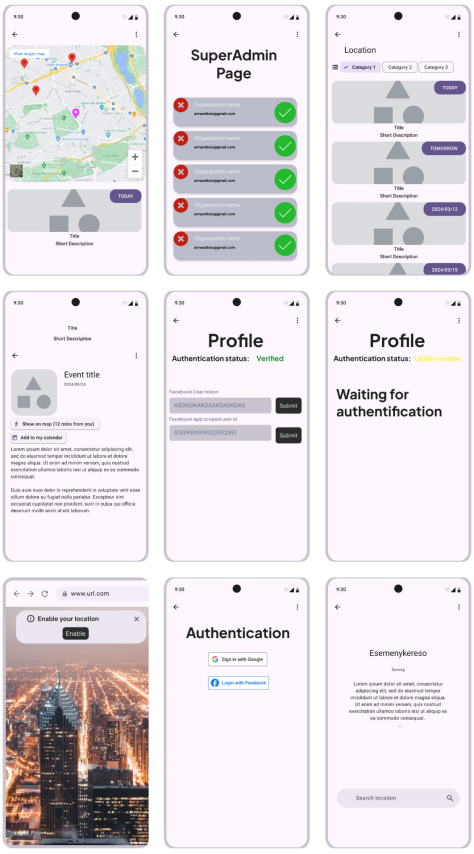
A folyamat során a kód minőségét folyamatosan ellenőrizzük, és a sikeres tesztelés után az alkalmazás éles környezetbe kerül telepítésre. Ez a megközelítés lehetővé teszi a hatékony és megbízható szoftverfejlesztést és -kiszolgálást.

6. fejezet

Weboldal bemutatása

6.1. Weboldal kezdeti tervei

A weboldal kezdeti tervei alapozták meg azt az irányt, amelyet folyamatosan követtünk a weboldalunk elkészítéséhez. Az alábbiakban tekintsünk meg pár fontos koncepciót, amelyek a weboldal fejlődésének az alapját képezték.

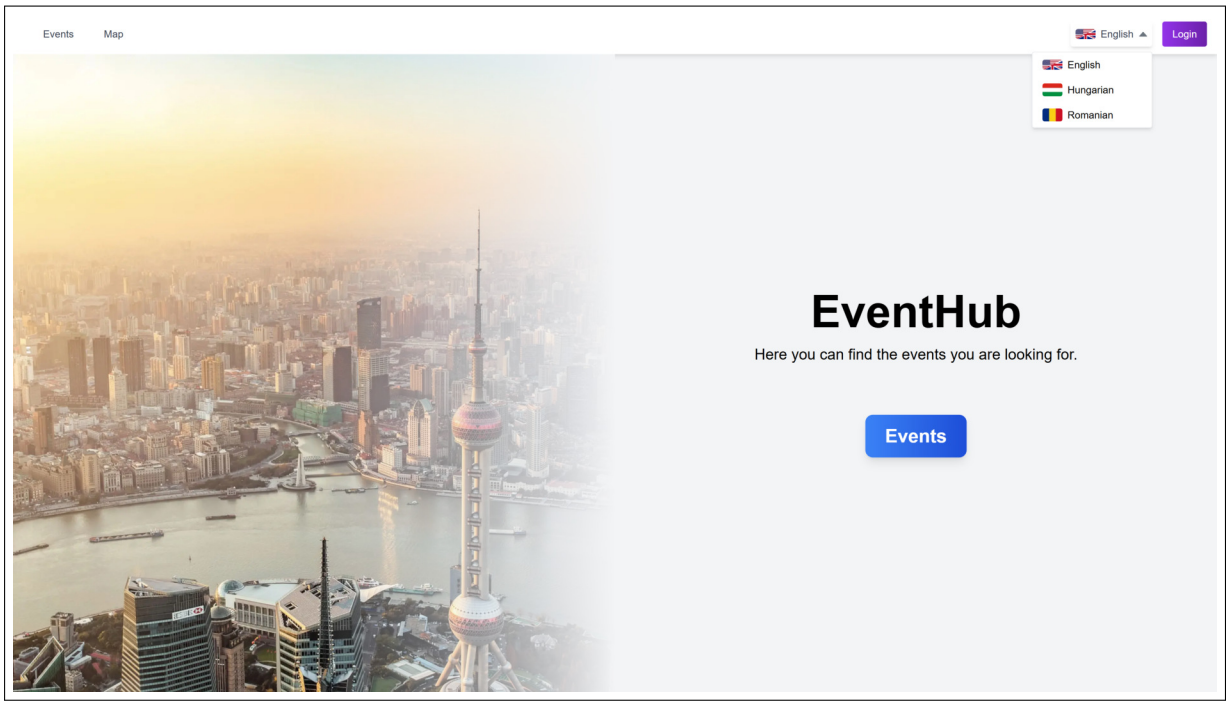


6.1. ábra. A weboldal kezdeti tervei

A tervek célja egy átlátható és felhasználóbarát felület kialakítása volt, amely az események kezelésére és megjelenítésére szolgál.

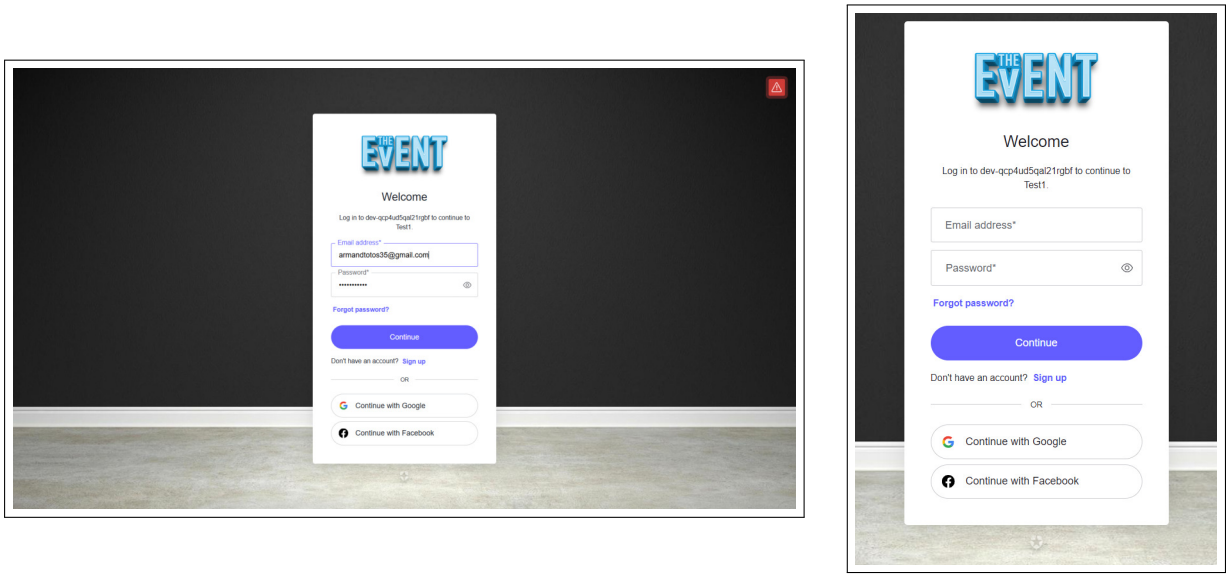
6.2. Weboldal jelenlegi állapota

6.2.1. Főoldal és bejelentkezés



6.2. ábra. Főoldal

Az 6.2. ábrán látható a főoldal, amellyel először találkozik a felhasználó.

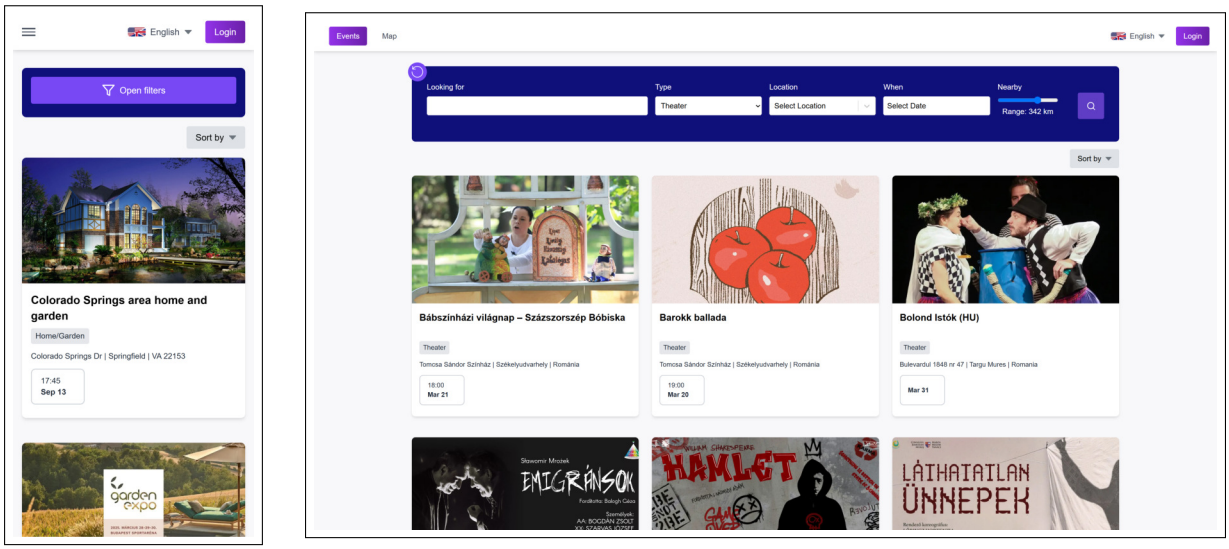


6.3. ábra. Bejelentkezés

Az 6.3. ábrán látható a bejelentkezési panel, amely többféle bejelentkezési lehetőséget kínál a felhasználóknak. A bejelentkezés révén elérhetők a szerepkörökhöz kötött

funkciók, például az admin jogosultságok és eseménykedvelések, valamint a későbbiekben érkező értesítések.

6.2.2. Események oldal

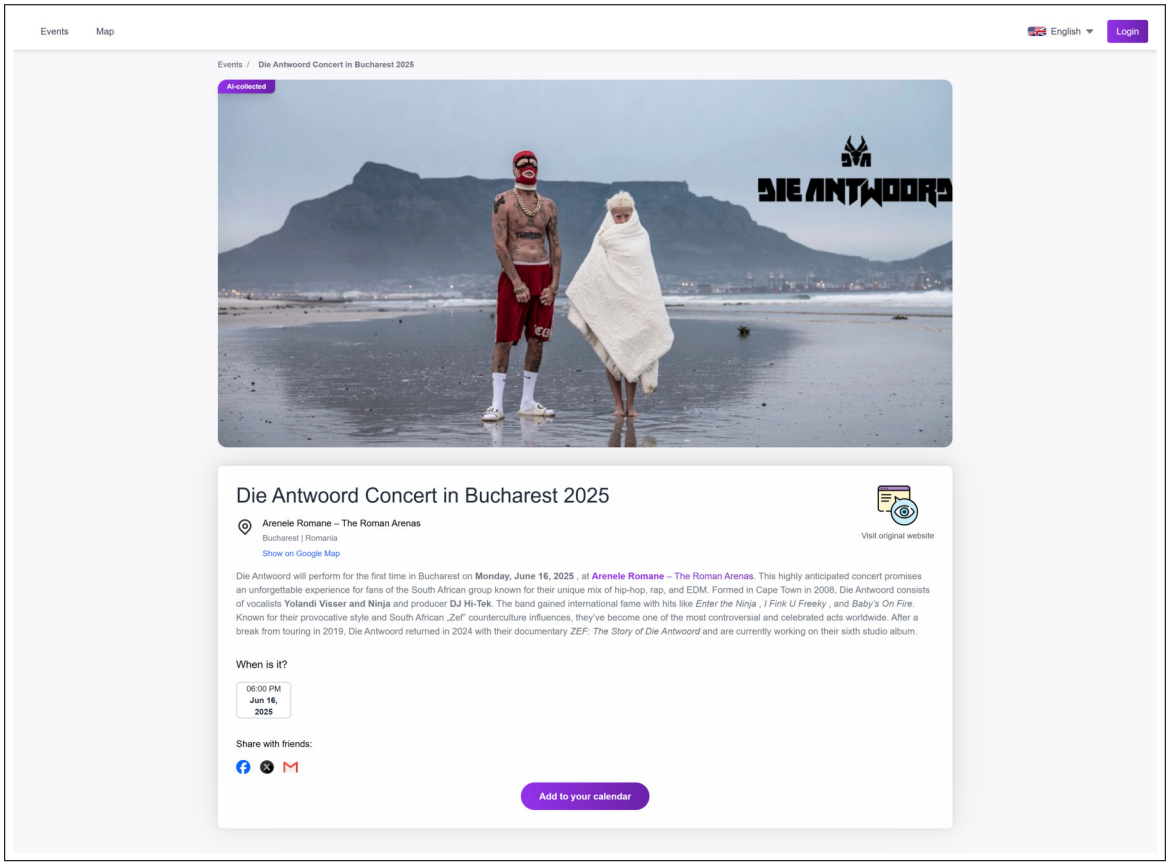


6.4. ábra. Események oldal

Az 6.4. ábrán láthatjuk az események listáját, minden felhasználó saját időzónája szerint beállítva. Itt lehetőségünk van szűrni, kategorizálni, hogy minden felhasználó saját igényéhez illő eseményeket böngészhessen.

Az imént látható eseménykártyák kattinthatóak, amellyel megnyithatjuk az események részletesebb oldalát.

6.2.3. Eseményoldal

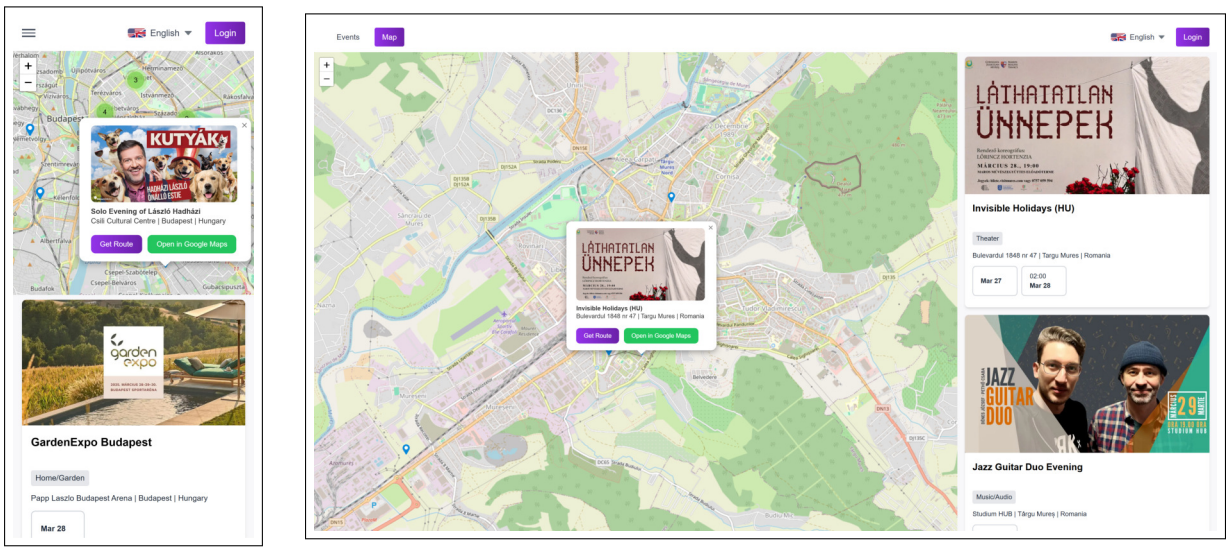


6.5. ábra. Esemény oldal

Az 6.5. ábrán látható egy esemény részletesebb leírása, illetve egy naptárhoz adás gomb, amely megnyomása után a felhasználó hozzá tudja adni az eseményt (több időpont esetén, a felhasználó ki kell válassza, hogy melyik időpontot szeretné) bármely általa használt naptárhoz. Az „Eredeti oldal megtekintése" gomb segítségével megtekinthetjük az esemény eredeti oldalát, amely lehet Facebook vagy más weboldal. A gomb felirata és ikonja az eredeti oldal típusától függően változik.

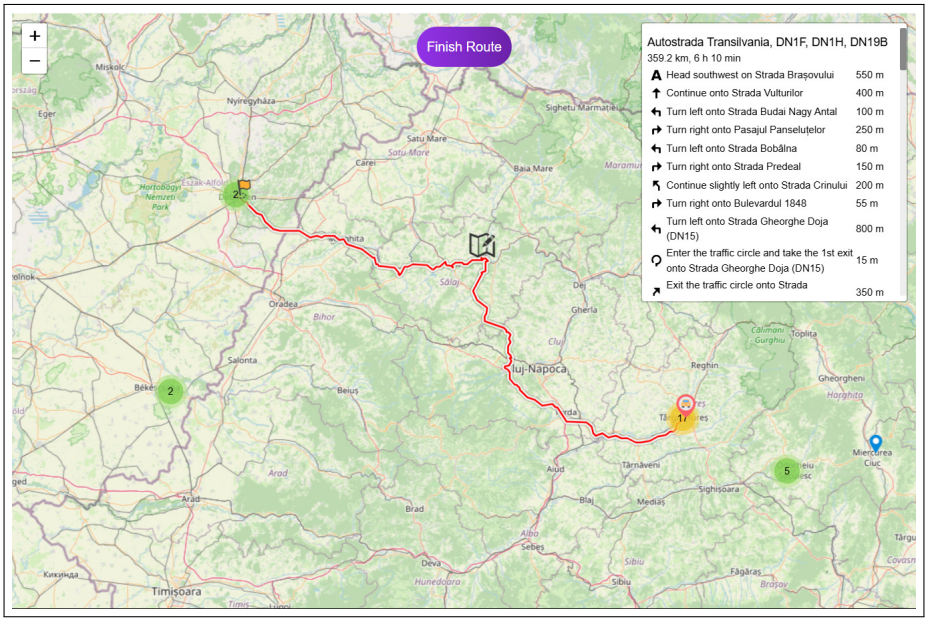
6.2.4. Térkép

Fontosnak tartottuk, hogy az események a térképen is megjelenjenek ezért létrehoztunk egy térképnézetet, ahol a felhasználók könnyebben megtalálhatják az általuk keresett eseményeket (lásd 6.6. ábra).



6.6. ábra. Térkép nézet

A térképünk egy útvonal tervezőt is tartalmaz, hogy a felhasználó láthassa, hogy hogyan juthat el az adott eseményhez (lásd 6.7. ábra).

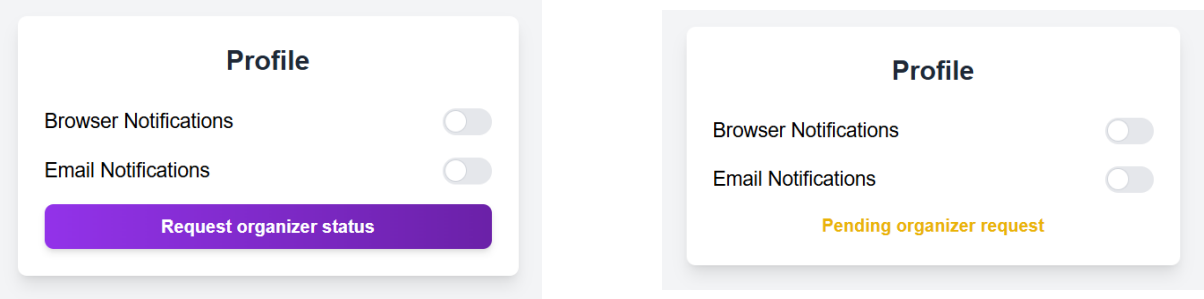


6.7. ábra. Útvonaltervezés

6.2.5. Profil oldalak

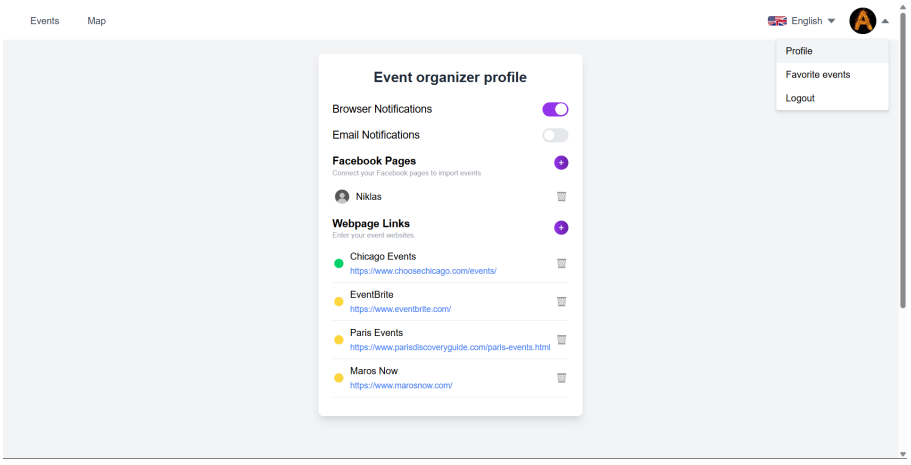
A szerepkörök többféle profilt követelnek meg a megfelelő működés érdekében:

- **User Profil:** Az oldal lehetőséget ad arra, hogy a felhasználó eseményszervezői státusz kérelmet indítson, amely után jelezzük számára, hogy „elbírálás alatt” áll (lásd 6.8. ábra).



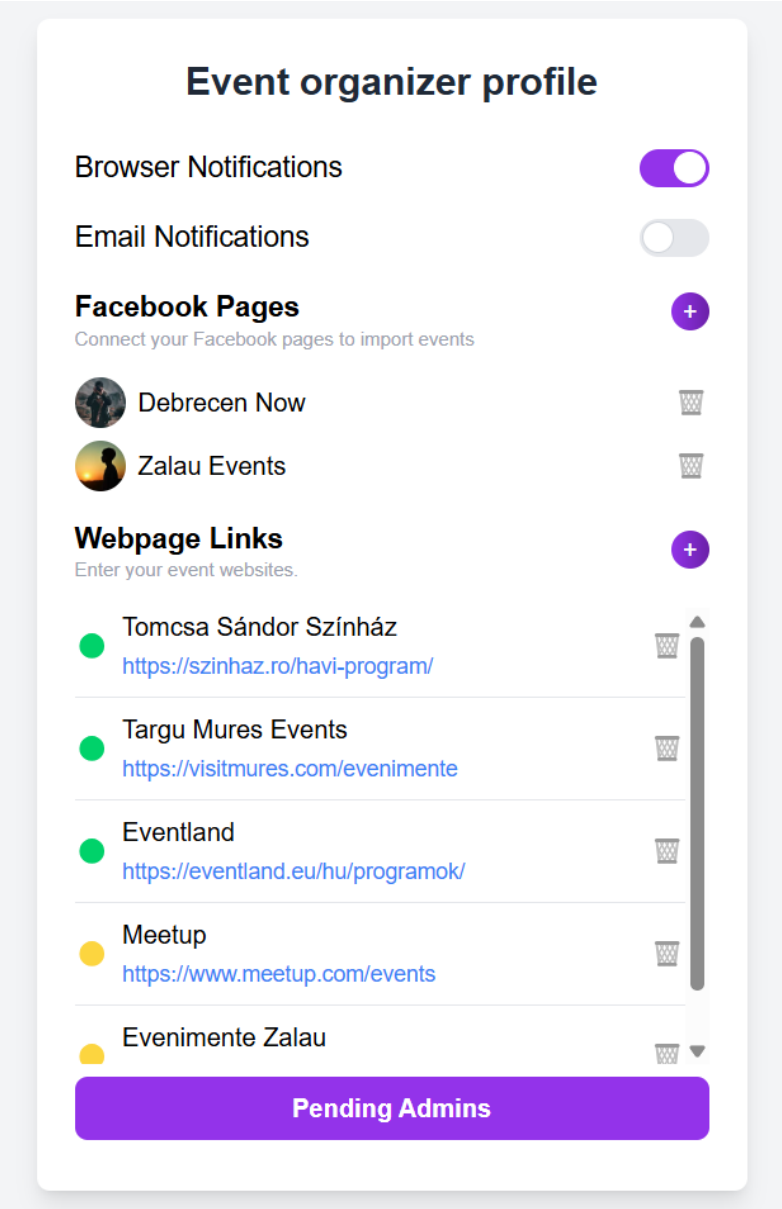
6.8. ábra. Felhasználó profil beállításai

- **Event organizer profil:** Ez az oldal lehetőséget ad, hogy az eseményszervező hozzáadja a Facebook oldalát az EventHub-hoz. Ha az oldalra Superadmin-ként vagyunk bejelentkezve, akkor megjelenik számára a „peding event organizers” és a „pending links” gomb, amely átirányítja a Superadmint azokra az oldalakra (lásd 6.9. ábra).



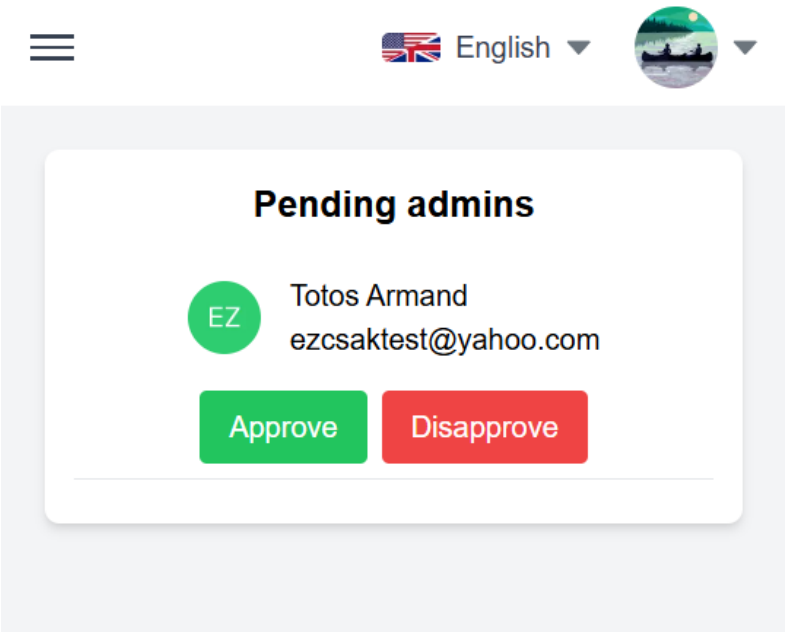
6.9. ábra. Eseményszervező profil oldala

- **Superadmin profil:** Ez az oldal a Superadmin profil oldala, ahol megjelenik a peding event organizers és a pending links gomb, amely átirányítja a superadmint azokra az oldalakra (lásd 6.10 ábra).



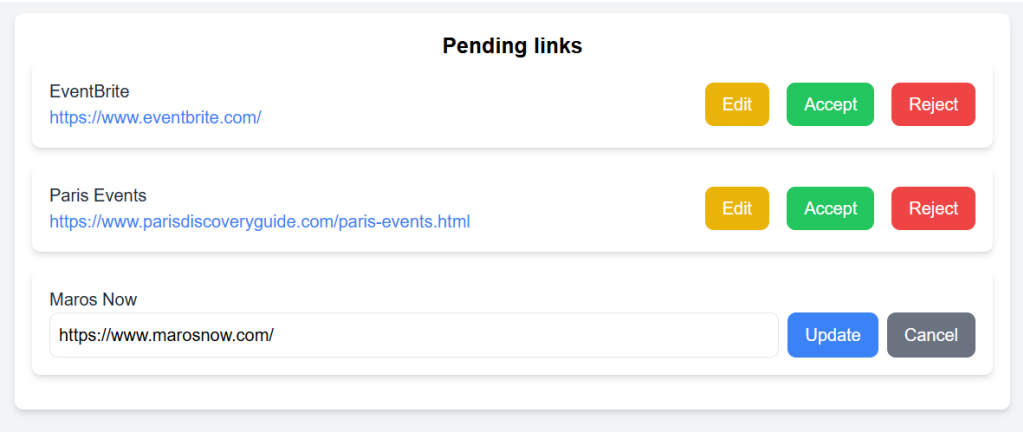
6.10. ábra. Superadmin profile

- **Pending Admins:** Itt a Superadmin elfogadhatja vagy elutasíthatja az adminkérelmeket (lásd 6.11. ábra).



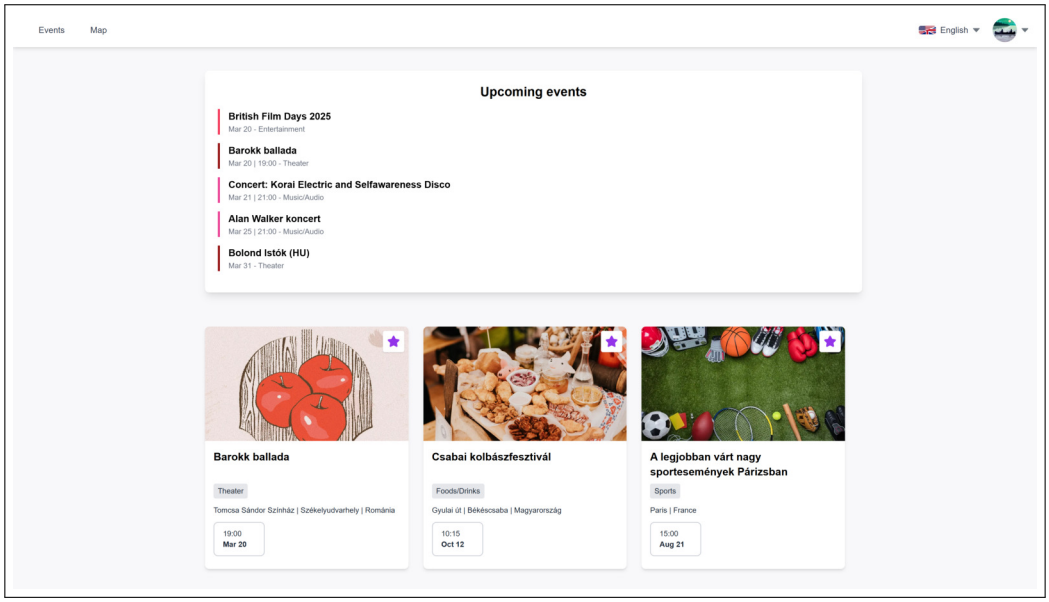
6.11. ábra. Elbírálás alatt lévő eseményszervezők (mobil nézetben)

- **Pending Links:** Itt a superAdmin elfogadhatja vagy elutasíthatja a weboldal hozzáadásokat, illetve felül is írhatja azokat, abban az esetben, ha az eseményszervező véletlenül helytelenül adja meg a weboldal linkjét (lásd 6.12. ábra).



6.12. ábra. Elbírálás alatt lévő weboldal linkek

6.2.6. Kedvelt események



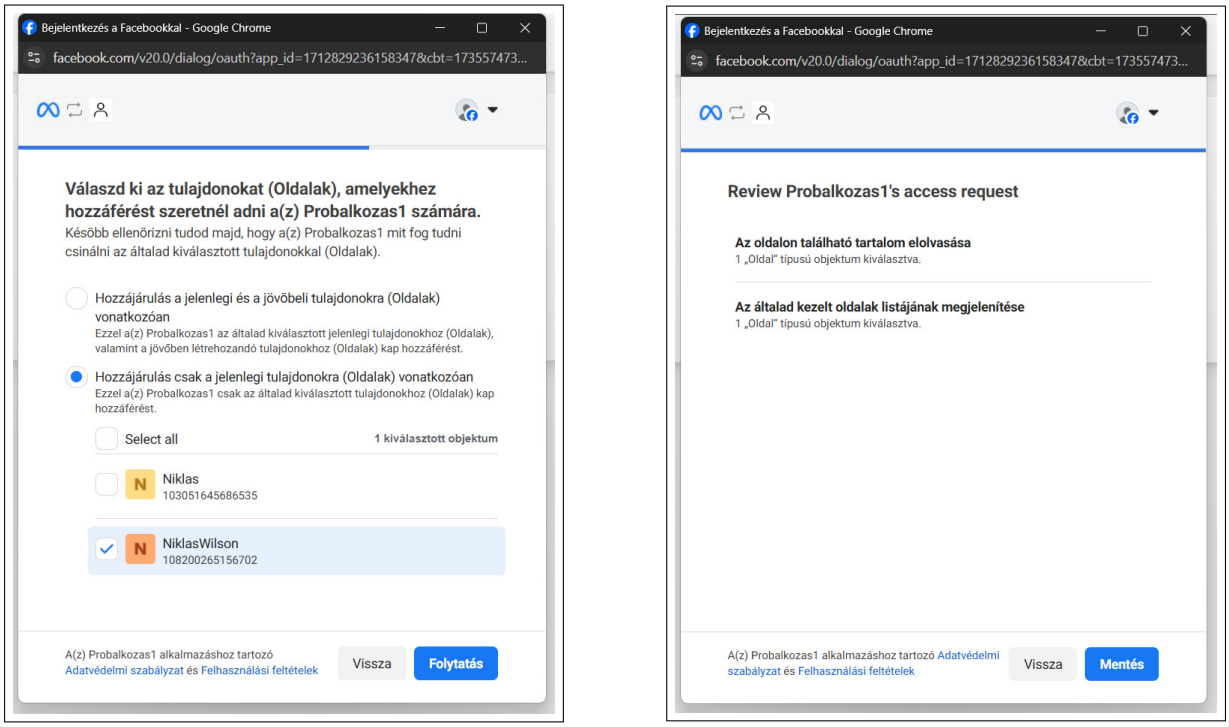
6.13. ábra. Kedvelt események oldal

A bejelentkezett felhasználóknak lehetőségük van kedvelni/követni eseményeket, amelyeket a kedvelt események oldalon tekinthetnek meg, ahol egy idővonal is a rendelkezésükre áll. Az idővonal segíthet az események időbeli előrehaladásának nyomon követésében (lásd 6.13. ábra).

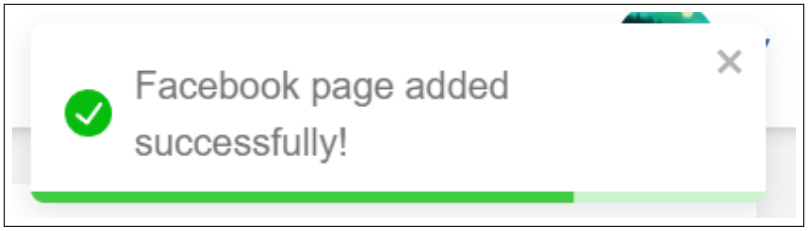
6.3. Facebook oldalakkal való összekapcsolás

Az EventHub alkalmazás lehetőséget biztosít az eseményszervezők számára, hogy egyszerűen hozzáadhassák Facebook oldalaikat, amelyekről az események kerülnek majd begyűjtésre. Ehhez a Facebook SDK-t [Fac24b] használjuk, amely lehetővé teszi a zökkenőmentes integrációt és hitelesítést.

Az eseményszervező a profiloldalán keresztül bejelentkezik a Facebook fiókjába. Ez a bejelentkezés automatikusan lehetővé teszi az alkalmazás számára, hogy hozzáférést nyerjen a Facebook oldalaihoz. Az eseményszervező egyszerűen kiválasztja azokat az oldalakat, amelyeket hozzá szeretné adni az EventHub-hoz, majd elfogadja az általunk kért engedélyeket (lásd 6.14. ábra). Ez a folyamat biztosítja, hogy az események automatikusan frissüljenek az alkalmazásban, anélkül, hogy manuálisan kellene kezelni az adatokat.



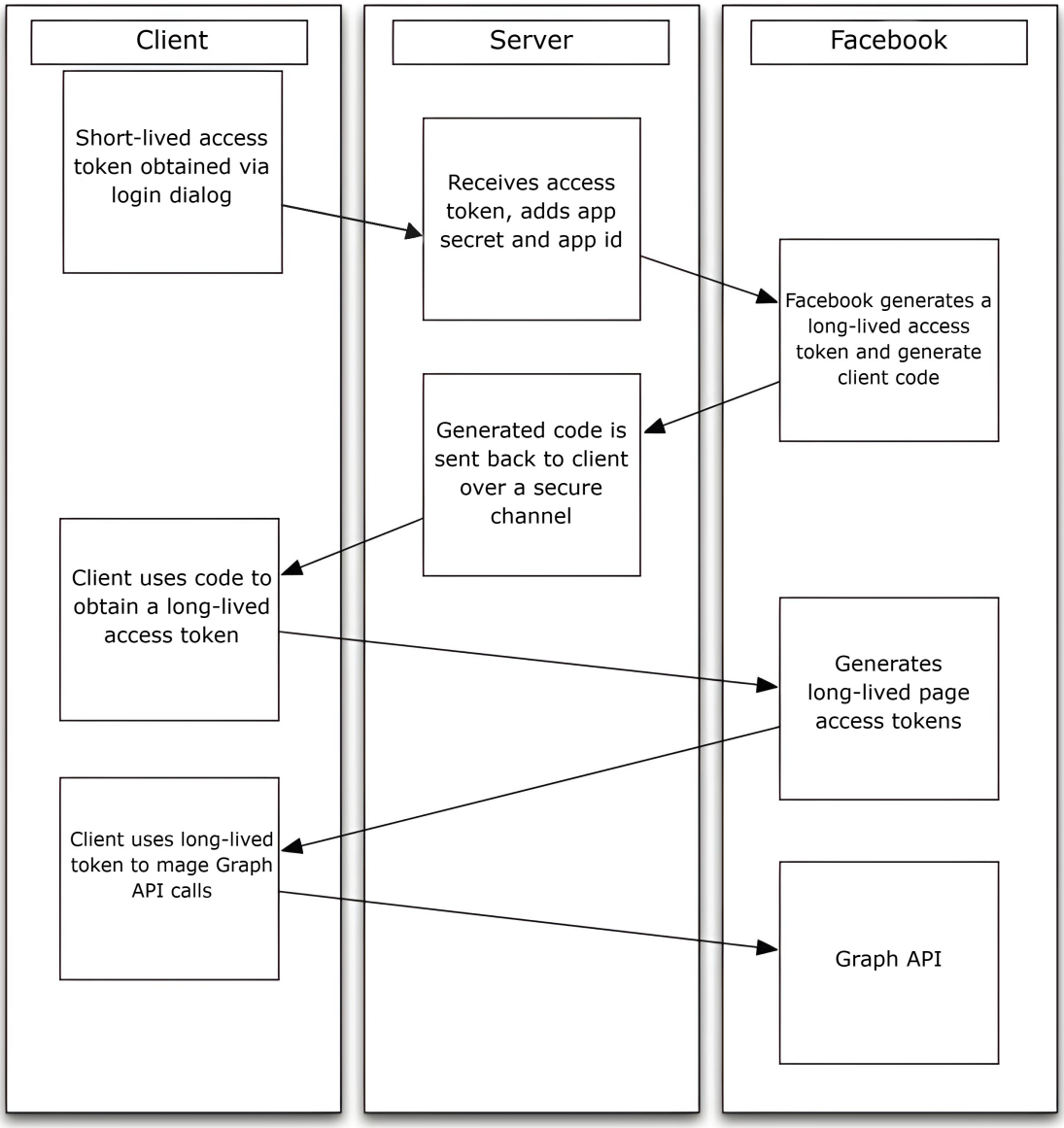
6.14. ábra. Facebook oldalak megadása és engedélyek kérelme



6.15. ábra. Facebook oldalak sikeresen hozzáadva

Az 6.15. ábrához hasonló felugró *„Toast”* üzenetek tekinthetőek meg máshol is a weboldalon a cselekvések megerősítése érdekében.

Az általunk használt Facebook tokeneket az előbb említett bejelentkezést követően kapjuk meg az SDK segítségével. A tokenek átmennek egy többlépcsős hitelesítésen és kibővítésen (lásd 6.16. ábra), hogy az eseményszervezők csak egyszer kelljen összekössék a facebook oldalukat az alkalmazásunkkal.



6.16. ábra. Tokenszerzés

7. fejezet

Továbbfejlesztési lehetőségek

- **Böngészős és e-mail értesítések**: Ez a funkció lehetővé tenné, hogy a felhasználók értesítéseket kapjanak a böngészőjükben vagy e-mailben az általuk követett eseményekről, így nem maradnának le róluk.
- **Manuális eseményfeltöltés**: Az adminisztrátorok számára lenne lehetőség, hogy manuálisan is feltölthessék eseményeiket, ha nem rendelkeznek Facebook oldallal vagy saját weboldallal, ahonnan az eseményeket lehetne importálni.
- **AI scraper fejlesztése**: Az AI-alapú adatgyűjtő eszköz továbbfejlesztése hatékonyabbá tenné az információkinyerést weboldalakról, gyorsabb feldolgozási sebességet, nagyobb pontosságot és a lapozási (pagination) problémák jobb kezelését biztosítaná.
- **Kollaboratív szűrés gépi tanulás által, a felhasználói kedvelések és interakciók alapján**: Ez a módszer gépi tanulási algoritmusokat használna arra, hogy a felhasználók korábbi kedvelései, interakciói (pl. kattintások) és preferenciái alapján személyre szabott eseményajánlásokat készítsen, javítva a felhasználói élményt és a releváns tartalom elérését.

8. fejezet

Összefoglaló

Az EventHub egy eseménykereső webalkalmazás, amely megkönnyíti az események felfedezését és nyomon követését. Gyakran elveszünk az információk forgatagában, és észrevétlenül elsuhannak mellettünk az érdekes események, akár otthon pihenünk, akár egy új városban járunk. Az EventHub ezt a gondot orvosolja: egy barátságos, átlátható felületen összegyűjti az eseményeket, mégpedig úgy, hogy automatikusan begyűjti az adatokat a Facebookról és más weboldalakról. Ez a megoldás élményeket szerezhet a felhasználóknak, és a szervezők dolgát is megkönnyíti, hiszen nem kell külön fáradozniuk az események népszerűsítésével.

Az EventHub egy modern technológiákra épülő alkalmazás, amely a Next.js/React és TypeScript segítségével biztosít letisztult és gyors felhasználói élményt, míg a háttérben a .NET/C# és a PostgreSQL gondoskodik az adatok megbízható kezeléséről. Az információkat a Facebook SDK, a Puppeteer és egy intelligens AI-Scraper gyűjti össze, amely pontos és hatékony működésével garantálja az adatok naprakészségét.

Az alkalmazás számos hasznos funkcióval rendelkezik, beleértve a térképes megjelenítést, a szűrőket, a naptárat és az útvonaltervezést, így segítve a felhasználókat az események könnyű felfedezésében és tervezésében. A különböző szerepkörök – vendég, felhasználó, szervező és Superadmin – révén mindenki számára átlátható és testre szabott élményt nyújt.

A fejlesztés során kiemelt szempont volt a gördülékeny működés és a jövőbeli bővíthetőség, amit a GitHub CI/CD és a Docker támogat. Az EventHub nem csupán egy alkalmazás, hanem egy megbízható útmutató, amely közelebb hozza a környezetünkben zajló eseményeket.

Ábrák jegyzéke

2.1. Mizu.ro kezdőlapja és eseménykategóriák . . . 8
2.2. Mizu.ro szerény eseménykínálata . . . 9
2.3. Visitmures.com eseménykínálata . . . 10
2.4. Meetup.com eseménykínálata. . . . 11

3.1. Használati eset diagram . . . 13

5.1. Architektúra diagram . . . 20
5.2. Autentikációs folyamat . . . 28
5.3. Az EventHub alkalmazás hitelesítési folyamata . . . 29
5.4. Függőségek . . . 34
5.5. Client Credentials Flow folyamata . . . 41
5.6. Az adatbázis sémája és a táblák közötti kapcsolatok . . . 44
5.7. AI Scraper diagram . . . 49
5.8. Adatok pontossága . . . 51
5.9. Több eltérő kezdődátum . . . 51
5.10. Kitelepítési Diagram . . . 61
5.11. Standard SSL és Wildcard SSL összehasonlítása . . . 62

6.1. A weboldal kezdeti tervei . . . 64
6.2. Főoldal . . . 65
6.3. Bejelentkezés . . . 65
6.4. Események oldal . . . 66
6.5. Esemény oldal . . . 67
6.6. Térkép nézet . . . 68
6.7. Útvonaltervezés . . . 68
6.8. Felhasználó profil beállításai . . . 69
6.9. Eseményszervező profil oldala . . . 69
6.10. Superadmin profile . . . 70
6.11. Elbírálás alatt lévő eseményszervezők (mobil nézetben) . . . 71
6.12. Elbírálás alatt lévő weboldal linkek . . . 71
6.13. Kedvelt események oldal . . . 72
6.14. Facebook oldalak megadása és engedélyek kérelme . . . 73
6.15. Facebook oldalak sikeresen hozzáadva . . . 73
6.16. Tokenszerzés . . . 74

Táblázatok jegyzéke

2.1. Hasonló platformok összehasonlítása 11

Kódrészletek jegyzéke

5.1. Paginator részlet 22
5.2. Események lekérdezése szűrési paraméterekkel 23
5.3. QueryParamsBuilder osztály 24
5.4. Fetch wrapper 25
5.5. Metódusok az API hívásokhoz 26
5.6. Importer main függvény 46
5.7. Scraper interface 47
5.8. Facebook impotert események lekérése 47
5.9. AI-scraper által kinyert dátum 50
5.10. AI scraper kinyerési stratégia 52
5.11. AI-scraper ütemezése 53
5.12. Docker Compose konfiguráció 59

Irodalomjegyzék és hivatkozások

[Aut24a] Auth0. Documentation. https://auth0.com/docs, 2024.

[Aut24b] Auth0. Machine-to-Machine Authentication. https://auth0.com/docs/get-started/auth0-overview/create-applications/machine-to-machine-apps, 2024.

[Boy12] Ryan Boyd. *Getting started with OAuth 2.0.* " O'Reilly Media, Inc.", 2012.

[Cra24] Crawl4AI. Open-Source LLM-Friendly Web Crawler & Scraper. https://docs.crawl4ai.com/, 2024.

[DM24] Namiot Dmitry and Sneps-Sneppe Manfred. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9):24–27, 2024.

[Doc24] Docker, Inc. *Docker Image Specification*, 2024.

[Fac24a] Facebook Graph API. Documentation. https://developers.facebook.com/docs/graph-api/, 2024.

[Fac24b] Facebook SDK. Documentation. https://developers.facebook.com/docs/javascript/, 2024.

[Fie24] Fielding, Roy T. Architectural Styles and the Design of Network-based Software Architectures. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2024.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Pearson Deutschland GmbH, 1995.

[Git24a] GitLab Inc. *GitLab CI/CD Documentation*, 2024.

[Git24b] GitLab Inc. *GitLab Runner Documentation*, 2024.

[Mic24a] Microsoft. Entity Framework Core Documentation. https://learn.microsoft.com/en-us/ef/core/, 2024.

[Mic24b] Microsoft. .NET Documentation. https://learn.microsoft.com/en-us/dotnet/, 2024.

[Nex24a] Next.js. Documentation. https://nextjs.org/docs, 2024.

[Nex24b] Next.js. Supported Browsers. https://nextjs.org/docs/architecture/supported-browsers, 2024.

[Ope25] OpenStreetMap . Documentation. https://wiki.openstreetmap.org/wiki/API, 2025.

[Pos24] PostgreSQL Global Development Group. PostgreSQL Documentation. https://www.postgresql.org/docs/, 2024.

[Pup24] Puppeteer. Documentation. https://pptr.dev/, 2024.

[Sin84] Sinnott, Roger W. Virtues of the Haversine. *Sky and Telescope*, 68(2):159, 1984. Original publication of the formula's astronomical applications.

[SM09] Juha Savolainen and Varvana Myllarniemi. Layered architecture revisited—comparison of research and practice. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, pages 317–320. IEEE, 2009.

[TA24] Janemary Thirusanku and Lo Poh Ai. Technology innovation in event management. *Journal of Advanced Research in Technology and Innovation Management*, 10(1):1–13, 2024.

[xUn24] xUnit.net. xUnit.net Documentation. https://xunit.net/, 2024.